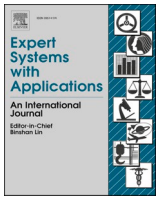
Expert Systems With Applications 241 (2024) 122678

Contents lists available at ScienceDirect

Expert Systems With Applications

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



MIGAN: GAN for facilitating malware image synthesis with improved malware classification on novel dataset

Osho Sharma [a], Akashdeep Sharma [b,*], Arvind Kalia [a]

[a] Department of Computer Science, Himachal Pradesh University, Shimla, India
[b] Department of Computer Science and Engineering, UIET, Panjab University, Chandigarh, India

ARTICLE INFO

Keywords:
Deep learning
Convolutional neural networks
Generative adversarial networks
Malware classification
Cybersecurity

ABSTRACT

Malware visualization is a technique wherein malware binaries are represented as grayscale or color images in order to identify and extract discriminating features for classification. This technique is effectively better than classic machine learning based malware recognition techniques that require significant domain expertise or time-consuming behavioral analysis to identify discriminating features. In this manuscript, a Generative Adversarial Network (GAN) architecture is introduced for facilitating malware image synthesis called 'MIGAN', that can quickly produce high-quality synthetic malware images and then classify malware samples into families. The proposed framework consists of a generator and discriminator network paired with a classification module. The novelty exists in the GAN network structure, hybrid loss function, new dataset and classification network structure. The MIGAN generated images manage to achieve better Inception Score than original malware images (2.81 vs 1.90, respectively) along with better Fréchet Inception Distance score and Kernel Inception Distance score. The synthetic malware images primarily serve two purposes: firstly, it solves the class imbalance problem in custom built and public 'Malimg' datasets. Secondly, since these images resemble existing malware images, it is assessed to be fairly similar to upcoming 'zero-day' or 'previously unseen' malware that can be eventually discovered in the future. The two classification networks (custom classification network with traditional learning approach and pretrained Resnet50v2 network with transfer learning approach) were supplemented and trained with nearly 50,000 synthetic malware images. The proposed framework achieved promising scores of 99.2 % Area Under the Curve (AUC), 99.3 % F1-score and 99.5 % Accuracy. The comprehensive evaluation and excellent results demonstrate the effectiveness of the proposed framework. This framework can also be applied to image synthesis with several other types of images.

1. Introduction

Malware, short for 'malicious software(s)'- are the software(s) responsible for security breaches, damaging computational systems, leakage of private data, spying on user activities without their consent and a majority of other cyber-crimes. Recent times have witnessed a heavy increase in malware attacks due to tremendous growth in internet and communication technologies, along with other developmental factors. Security firms that monitor the growth of malware have shown some alarming statistics. The reports of AV-TEST reveal that the volume of fresh malware programs along with PUAs (Potentially Unwanted Applications) which are being registered in their database each day are over 4,50,000 (AV-TEST Malware Statistics, 2022). According to VirusTotal malware statistics, their system registers nearly 1.6 million distinct hits on new malware every day in 2022 (VirusTotal Statistics, 2022). The extreme growth of malware necessitates actions from the security experts and academicians; therefore, malware research is a burning topic of concern for the safety and security of people and organizations.

In general, malware recognition and classification is a challenging research topic due to reasons like lack of updated malware datasets (as malware are prone to becoming obsolete) and the complications introduced due to 'concept-drift', where malware creators consistently push newer/evolved malware variants to make them adjust according to changing computational environments (Gibert et al., 2020). A majority of malware thus created, often utilize the code structures of previous malware variants (Gibert et al., 2020). In the area of malware analysis, one of the most common strategies is to store the signatures of all

* Corresponding author.
E-mail addresses: oshosharma.cs@gmail.com (O. Sharma), akashdeep@pu.ac.in (A. Sharma), arvkalia@gmail.com (A. Kalia).

https://doi.org/10.1016/j.eswa.2023.122678
Received 9 November 2022; Received in revised form 9 November 2023; Accepted 18 November 2023
Available online 25 November 2023

malware instances in a database (usually maintained by a security firm) and comparing the malware signatures with the entries in the database. However, this strategy is known to fail against formerly unknown malware samples since their signatures are lacking in the database (Moti et al., 2021). Furthermore, keeping track of all possible malware signatures is computationally inefficient and costly (Nahmias et al., 2020). Therefore, the 'zero-day' and 'previously-unseen' malware makes malware detection much more difficult.

To get over the shortcomings of signature-based malware detection, classic machine learning methods have been developed. These methods rely on identification of discriminating malware features. The important features are selected and used to train a model that can recognize and classify malware variants effectively. However, identification of highly discriminating features and their extraction is computationally expensive (Darem et al., 2021). Several studies in the past have tried to address the problem of identification of discriminating features for malware detection. Their approach involves conversion of malware binaries into other types like audio (Farrokhmanesh & Hamzeh, 2019), image (Vasan et al., 2020a), signal (Pei et al., 2020) etc. Out of these methods, malware to image conversion (also known as malware visualization) based approaches have shown very promising results. However, a prominent issue in the domain of malware image-based classification methods still remains unaddressed - the problem of class imbalance. Several datasets in related works face the issues related to undersampled classes, hampering the learning process and affecting the overall performance of machine learning/ deep learning algorithms. To attain optimal learning, each class in a dataset should contain a sufficient number of samples. However, malware datasets used in several prior studies are affected with class imbalance for e.g. the two most popular public benchmark malware datasets used extensively in related works: Microsoft Malware Challenge dataset (Ronen et al., 2018) and 'Malimg' dataset (Nataraj et al., 2011) are heavily imbalanced. To reduce the problem of class imbalance, researchers often duplicate/repeat the samples in dataset classes. However, sample duplication/repetition may cause the fitted model to be biased i.e., the model may become overfitted (Buda et al., 2018). Therefore, the problem of class imbalance needs to be addressed which indicates a dire need for a framework capable of handling this issue. The current work focuses on resolving this issue by creating previously unseen malware image samples using GANs (Generative Adversarial Networks) to resample the classes in datasets.

GANs are a prominent generative deep structure which are primarily used to create synthetic data. Several previous works have utilized GAN variants to generate and create images, videos, sound, natural languages, sequences etc. (Y. Zhang et al., 2021). In the purview of malware research, GANs have two main implications: firstly, it is known that malware developers try to evolve malware samples by making slight alterations to the code which is similar to the ability of GANs to produce new malware instances by addition of noise in existing samples (Rezaei et al., 2021). Secondly, GANs have been used for developing adversarial malware instances that can bypass or attack machine learning models (Fang et al., 2021). Prior studies have shown successful utilization of variants of GANs like BGAN (Boundary Seeking GAN) (Moti et al., 2021) to generate signatures of previously unseen malware samples, where BGAN was utilized to augment existing IoT and packed malware datasets to improve the classification performance of the proposed malware detection framework to nearly 99 %. Similarly, Peng et al. (2021) demonstrated adversarial sample creation using GAN based models, which improved the overall detection accuracy of their proposed system to nearly 96.35 % using their proposed GAN variant.

To address some of the critical issues in the sphere of image-based malware family recognition and classification, a framework is suggested which utilizes the ability of GANs to reduce the problem of class imbalance by generating previously unseen malware images. To accomplish this task, a variant of GAN- MIGAN (Generative Adversarial Network for facilitating Malware Image synthesis) is proposed which can generate class specific samples, thereby achieving better results in multiclass classification tasks. Since the created malware images resemble existing malware images, it is assessed to be fairly similar to upcoming zero-day malware that can be eventually discovered in the future. Additionally, both traditional and transfer learning approaches are leveraged to classify the resulting resampled/augmented datasets. To carry out traditional learning, a custom-built deep CNN (Convolutional Neural Network) is utilized and compared to the transfer learning approach, which employed a pretrained Resnet50v2 CNN trained on 1.5 million images from ImageNet dataset (Deng et al., 2009), which is fine-tuned to suit the task of classifying malware images into families. Since the networks are tested with a multitude of samples, the robustness and dependability of the overall framework increases. The overall performance of the system is additionally improved by utilizing a hybrid loss function that is shown to improve the discriminative power of deeply learned features. Finally, through extensive experiments on a new Windows malware image dataset and a popular benchmark Windows malware image dataset-'Malimg' (Nataraj et al., 2011), the suggested system is shown to successfully recognize and classify malware samples with high accuracy.

The scientific contributions of the manuscript are presented in the following points:

1. A framework for malware recognition and classification that integrates malware visualization, malware image sample generation, automatic feature extraction and classification is proposed. To visualize the malware instances, grayscale and color image-based conversion strategies are utilized.
2. Proposal of a new variant of GAN- MIGAN (Generative Adversarial Network for facilitating Malware Image synthesis), for the generation of synthetic malicious and benign malware images followed by augmentation of self-created and public benchmark datasets with synthetic images in order to improve the classification performance, also reducing the problem of class-imbalance. Additionally, another GAN variant: DCGAN (Deep Convolutional GAN) is used to compare the sample quality with respect to MIGAN samples using quantitative GAN metrics.
3. Proposal of a traditional learning-based custom deep CNN structure which is compared with transfer learning-based Resnet50v2 reference CNN, fine-tuned for the task of classifying malware images into families.
4. Proposal of a hybrid loss function which is shown to improve the discriminative power of deeply learned features based on the idea that the distance between intra-class samples should be as small as possible and the distance between inter-class samples should be as large as possible, consequently increasing the classification accuracy.

For researchers and security professionals, the trained synthetic data and generation model is made publicly available. The study contributes novel prospects for developing effective, low cost and highly reliable malware recognition and classification systems for uncontrolled environments. The proposed framework has good implications and usability in the area of cyber-security and can be deployed using cloud computing solutions for the masses. Additionally, MIGAN reduces the problem of insufficient data samples, imbalanced data, and missing data labels as well as to lessen the impact of samples that are not distributed evenly.

The remainder of this manuscript is divided into these sections: the second section goes through some of the well-known malware analysis techniques that utilize generative learning and malware visualization. The suggested framework is outlined and its components are explained in the third section. The fourth section depicts the study's custom built and benchmark datasets and the evaluation metrics. The outcomes of this work and performance analysis is summarized in the fifth section. Afterwards, the work comes to a closure with a conclusion and future plans in the sixth section.

2. Related works

The published works in the area of malware recognition and classification can be categorized based on the underlying methods used to accomplish this task: methods relying on static features (like signatures, checksums) (Amin et al., 2020), methods utilizing dynamic features (api calls, opcodes etc.) (Amer & Zelinka, 2020; Escudero García & DeCastro-García, 2021), methods combining both static and dynamic features (also called hybrid methods) (Darabian et al., 2020), traditional machine learning based techniques (Bai et al., 2021; Dehkordy & Rasoolzadegan, 2021), singular or multimodal deep learning based methods (Gibert et al., 2019; Gibert et al., 2020), methods based on conversion of malware binaries into other formats like audio (Farrokhmanesh & Hamzeh, 2019), signal (Pei et al., 2020) or images (Naeem et al., 2020; Vasan et al., 2020a) to isolate discriminating features. The current work primarily addresses issues in two areas: malware visualization (i.e. conversion of malware binaries into grayscale and color images) and GAN based sample generation followed by data augmentation (Moti et al., 2021). We highlight some of the most important works in malware visualization and generative learning in the current section.

2.1. Image based malware recognition and classification

The method of conversion of malware binaries into grayscale images is used by several prior studies. The grayscale malware images of the same malware family share several visual characteristics that can be used to quickly and easily identify malware family types. Bakour & Ünver (2021) demonstrated a framework to classify Android malware images consisting of machine learning classifiers and deep neural networks. The researchers utilized five grayscale image data sets and utilized six machine learning classifiers. Additionally, the authors employed Resnet and inceptionv3 reference CNNs. Their proposed model was shown to achieve nearly 98.2 % detection accuracy. On a similar note, Gibert et al. (2019) proposed a grayscale malware image classification framework that consisted of a CNN structure having four convolutional and fully connected layers. The authors utilized two public benchmark datasets: Microsoft malware challenge dataset and Malimg dataset and their model was shown to achieve nearly 98 % classification accuracy, however, their method completely ignored the class imbalance issue prevalent in both public malware datasets. Darem et al. (2021) proposed an obfuscated malware detection and classification framework where the authors converted the opcode sequences from Microsoft malware challenge dataset into grayscale images. The authors further proposed an ensemble technique to aggregate the performance of two classification models: XGBoost classifier and a custom CNN model to achieve an aggregate classification performance score of nearly 99.12 % in the two public datasets: Microsoft challenge dataset and Malimg, however class imbalance issue still remained unaddressed in their work. Jain et al. (2020) suggested a method for classifying malware images using grayscale pictures of Windows malware. To perform the classification task, CNN and ELM (Extreme Learning Machine) based models were used. With a good overall accuracy of nearly 96 %, the ELM-based model was shown to outperform the CNN model on the public dataset: Malimg.

The task of reducing the computational requirement can be achieved by utilizing transfer learning approach wherein a trained model is fine tuned to suit another similar problem at hand. Sudhakar and Kumar (2021) presented a framework wherein both traditional learning and transfer learning approaches were utilized to classify grayscale malware images in Malimg and Microsoft dataset. A custom deep CNN was utilized for traditional learning approach and a pretrained Resnet50 model was used for transfer learning. The aggregate accuracy of their framework reached nearly 99 % however, their method completely ignored the class imbalance issue prevalent in both of the public malware datasets. Cui et al. (2018) utilized a customized convolutional neural network architecture to perform image based malware classification using Malimg dataset (Nataraj et al., 2011). Their approach involved utilizing a novel swarm intelligence algorithm called bat algorithm which was used to solve the problem of class imbalance. Additionally, the authors utilized image-based operations (rotation, shearing, flipping, rescaling) operations to reduce the class imbalance problem. Their approach achieved nearly 94.5 % classification accuracy in Malimg (Nataraj et al., 2011) dataset. Tuncer et al. (2021) proposed a malware visualization-based framework wherein the authors utilized Local Binary Pattern, Singular Value Decomposition and Local Ternary Pattern based hybrid features. In total, the authors extracted 5162 features and reduced them to 512 using Principal Component Analysis. Finally, Linear Discriminant Analysis was used as a classifier and the framework achieved nearly 88.08 % accuracy. On the Android platform, Mercaldo and Santone (2020) attempted to implement the similar malware image visualization methodology. Their suggested method employed a three-layer deep structure that made use of grayscale histogram values to identify and categorize malicious Android applications. Their method showed an accuracy of 96.6 % when tested on 50,000 benign and malicious Android apps. Similar method was proposed by Ding et al. (2020), suggesting a straightforward yet efficient CNN-based malware detection system to identify and categorize Android malware. The authors suggested a method to extract and transform the bytecode of Android applications into grayscale images and demonstrated decent results. To test their hypothesis that malware memory dumps can be utilized to accurately detect and categorize malware samples, Dai et al. (2018) employed a backdoor malware dataset. This is demonstrated by feeding malware memory dumps into Multilayer Perceptron (MLP) neurons as grayscale images. Their method yielded good accuracy results for the MLP model of around 95 %. A malware image classification method using co-occurrence matrix-based second order attributes on grayscale malware pictures was used by Verma et al. (2020). Their method showed a promising accuracy on Microsoft malware dataset of around 99 %.

Previous research has also shown the significance of using color malware pictures in comparison to grayscale pictures when identifying and categorizing malware samples. A deep CNN model and color image visualization method were suggested by Naeem et al. (2020) to identify and categorize Industrial Internet of Things (IIoT) and Android malware samples. Their suggested technique included transforming Android malware into color photos, which were then filtered through a customized deep CNN model to extract visual features. On datasets for the IIoT and Android malware, the model exhibited accuracy scores of 99 % and 98 %, respectively. Vasan et al. (2020a) used a technique to synthesize color images from malicious programs by converting grayscale images and using a color filter. In addition, the authors utilized VGG16, Resnet50, and Inceptionv3 as well as a deep CNN for image categorization. On color malware picture classification, their proposed approach IMCFN was demonstrated to attain an accuracy of 97.35 %. Another paper by Vasan et al. (2020b) showed the working of an ensemble technique that employed pretrained Resnet50 and VGG16 CNNs leveraging transfer learning approach that worked on Windows malware images. On the Malimg malware dataset, their model IMCEC showed 97.59 % accuracy.

The works in the development of neural networks for classification problems includes the work by Wang et al. (2016) where the authors combined self-adaptive mechanism with extreme learning machine (ELM) algorithm and proposed ‘SaLEM’ algorithm, which demonstrated the ability to construct optimal networks by always choosing the optimal number of hidden layer neurons. The efficacy of the algorithm was demonstrated using two classification problems, wherein the algorithm achieved excellent results of nearly 98 percent accuracy. More developments in the structure of CNNs for image classification were discussed in another work by Wang et al. (2022) where the researchers tried to integrate an evolutionary algorithm called Monarch Butterfly Optimization (MBO) algorithm to boost the neuron population using encoding based optimizations. The authors demonstrated nearly 99 %

accuracy for eight different public benchmark datasets for the multiclass image classification problem. For the purpose of classifying malware families, Li et al. (2022) proposed a method that relied on multimodal fusion and weight self-learning. The malware's byte, format, statistical, and semantic modalities were all fused in different ways to generate useful features for their system. Their classification model was then refitted with a weight self-learning mechanism that continually calculated log-loss in light of the class label and the probabilities indicated by each feature. The results showed that their method efficiently achieved high-quality classification results even when applied to datasets containing malware families whose distribution was highly imbalanced. Raff et al. (2017) proposed MalConv, a convolutional neural network capable of taking byte sequences of raw malware binaries as input and performing binary malware classification. The authors concluded that classification tasks are more reliant on header byte data of raw malware binaries, also indicating the ineffectiveness of batch normalization in byte sequence based neural learning. Ma et al. (2021) performed an empirical study comparing three popular approaches of malware classification relying on input formats: image based approach, binary sequence based approach and disassembly based approach. Based on the experiments conducted on four public malware datasets, the study pointed out that there was no concrete evidence that an individual class of methods is outperforming the other class of methods. Li et al. (2022) proposed a machine learning based method relying on n-gram and TF-IDF approach to extract and utilize multi-dimensional features from crypto-malware. Their work also incorporated static analysis based and statistical analysis based methods to extract multi-dimensional features from crypto malware binaries. Their detection model was able to achieve a good detection score of nearly 98 percent. However, the study did not effectively address the class imbalance issue prevalent in malware datasets.

Markov pictures, which excel in preserving statistical information of malware bytes, are used in several studies instead of grayscale and color images. In their proposed model named 'MDMC', Yuan et al. (2020) used markov pictures of malware samples and reported excellent malware recognition results of around 97 % on two open benchmark datasets: the Drebin (Android) dataset and the Microsoft malware challenge dataset. Sharma et al. (2022) used a Gabor filter-based technique to extract textures after rendering the malware samples as grayscale, color and Markov pictures. The classification of images in two public datasets and two custom built datasets was then performed using a custom deep CNN and Xception CNN based models. The suggested model was able to achieve classification accuracy scores of almost 99 %. It can be noticed that a majority of prior works completely ignore the issue of class imbalance and most of the works utilized imbalanced malware datasets to evaluate their frameworks.

2.2. Generative learning-based approaches for malware recognition and classification

Researchers in the past have used GANs to generate synthetic malware examples which resemble real world malware. Y. Zhang et al. (2021) improved the classification accuracy of their proposed deep learning-based malware classifier using GAN-based adversarial filtering. The malware detection was carried out using adversarial samples created by GANs utilizing the malware's byte sequence, and it was shown to achieve nearly 88 percent detection accuracy. The generated samples were also utilized to improve the resilience of the proposed malware classifier network. Hu and Tan (2017) introduced the 'MalGAN' technique, where the researchers utilized GAN variant to create adversarial malware instances utilizing the byte sequence of malware samples to bypass black-box machine learning-based malware detection approaches. Their suggested technique further utilized machine learning classifiers for classification and was tested on a large dataset of 180 thousand programs. In their analysis, the random forest and decision tree classifiers demonstrated a detection accuracy score of 97 percent. However, in their paper, the feature extraction and model training processes required a substantial duration. Y. Li et al. (2020) demonstrated the implications of GANs for attacking a neural network-based PDF malware classifier. The proposed fvGAN model utilized the 'mimicus' framework to collect and transform PDF feature vectors into fake PDF malware files. This task was followed by the assessment of the proposed system on the state-of-the-art PDF malware classifier- 'PDF rate'. The evaluation was carried out with three datasets: 'contagio', 'surrogate' and 'attack' and the suggested system was shown to achieve nearly 100 % evasion rate. H. Li et al. (2020) used bi-objective GAN for Android-based adversarial example creation. With a success rate of over 95 %, the adversarial instances created by the suggested technique were able to defeat firewall-equipped Android malware detection systems. Nagaraju and Stamp (2021) employed the Auxiliary-Classifier GAN model for image-based malware analysis. They tested a range of image sizes, including 32×32, 64×64, up to 512×512. Malicious binaries were used to extract and truncate grayscale photos to the required sizes. CNN and ELM models were also taken into consideration in addition to AC-GAN's discriminator, with CNN showing remarkable performance in recognizing synthetic images. However, their CNN based implementation showed less than 80 percent accuracy on Malimg dataset, which demonstrated scope of improvement. In a similar work by Nguyen et al. (2022), the researchers demonstrated four machine learning techniques: SVM, KNN, MLP, RBM and two ensemble techniques: Random Forest and XGBoost in the purview of malware classification. Additionally, the authors compared the performance of DCGAN, ACGAN and Resnet152 for image-based malware classification on 'malexe' dataset consisting of 26,412 malware binaries from 20 families. The authors utilized ACGAN discriminator to achieve classification accuracy of nearly 99 percent on 'malexe' dataset.

GANs are preferred for training of synthetic samples to raise the robustness and reliability of detection methods. Taheri et al. (2020) developed a GAN-based attack scenario to produce poisoned malicious Android apps to mask the machine learning-based detection model to counter adversarial malware threats in Android. The authors present an attack scenario based on Jacobian matrix using three benchmark datasets: drebin, contagio and genome. The Random Forest classifier demonstrated the highest accuracy score in their research (nearly 85 percent). In a subsequent work by Taheri et al. (2021), the researchers proposed two defense methods: Robust-NN and C4N (CNN + 1 nearest neighbor) based methods to modify the training class data contaminated by adversarial attacks. The authors utilized three datasets: drebin, contagio and genome and extract two types of features: API features and permission features. The proposed approach demonstrated promising detection scores of nearly 95 % in both of the proposed approaches.

To improve malware detection structures, GANs are usually coupled with other deep models such as Auto-encoders and reinforcement learning methods. Kim et al. (2018) incorporated transfer learning in a popular variation of GAN known as DCGAN and presented a deep autoencoder-based malware detection model. The authors depicted the usage of deep autoencoders for feature extraction and further improve GAN learning with the suggested tDCGAN model, which could create synthetic malware samples. On Kaggle's Microsoft malware challenge dataset, the suggested tDCGAN achieved a respectable average classification accuracy score of 95.74 percent. In a similar study by Fang et al. (2021), the researchers planned a GAN model A3CMAL (Actor Critic Algorithm) which could cheat or deceive the target malware classifiers using a supervised reinforcement learning driven agent that could automatically produce adversarial malware instances to bypass the malware classifiers. It was shown that the generated malware was bypassing the malware classifier causing non targeted misclassification. The scheme was tested with custom built Windows malware dataset containing nearly 21,203 files and demonstrated promising results. In another similar work, Li and Li (2021) proposed a reinforcement learning framework to generate Windows adversarial examples. Their proposed approach used malware executable samples as input and

utilized a feature extractor paired with a detection engine that generated malware executables capable of evading detection. Their method also involved a procedure for the creation of reward functions automatically without needing any manual input. Their method was tested on three classes of malware: backdoor, worms and trojan and their approach demonstrated a high accuracy rate of detection.

It is interesting to note that previous works suffered from several issues like: (a) Several authors have developed highly complex models that require a lot of processing time/resources. (b) Most previous works utilize old/obsolete malware datasets and there is a notable scarcity of updated malware datasets to evaluate the proposed methods. (c) Several works in the past utilized computationally intensive feature engineering techniques that required domain knowledge and time. (d) several previous works face the issue of class imbalance in malware datasets but completely ignore it. The manuscript addresses these issues by (a) proposing a traditional learning based custom deep CNN which is compared with a transfer learning based Resnet50v2 CNN, fine-tuned to suit the task of classifying malware images. (b) Apart from one popularly used public malware image dataset 'Malimg', we construct our own malware image dataset by downloading latest malware samples from the internet and converting these samples to images. (c) The proposal of MIGAN for malware image generation, reducing the problem of class imbalance and also the newly generated malware images resemble previously unseen malware samples, which further improve the robustness of the model. Also, the resilience of the model in recognizing unseen or zero-day malware increases.

3. Proposed framework

The GAN variant- MIGAN is developed for the synthesis of previously unseen malware samples in the form of 224×224 images and to improve the performance of semi-supervised multiclass image classification task by reducing class imbalance and augmenting existing malware image datasets. The network structures underwent extensive training from scratch. Although there are several GAN architectures that have been utilized in the past for picture synthesis, very few of these are effective for the creation of malware images, due to the complexity of malware images. Furthermore, out of several GAN variants, very few are known to successfully generate class specific samples, which is a critical requirement for the task of generating family specific malware images. Therefore, we develop a new GAN variant for malware image synthesis and augment the datasets. Fig. 1 depicts the contents of the overall framework. The proposed framework involves (a) collecting benign and malicious samples from the internet, (b) conversion of malware instances into grayscale and color images, (c) utilization of MIGAN to generate synthetic malware images to augment the datasets and addressing the problem of class imbalance, (d) performing the task of multiclass malware classification using custom-built deep CNN which is later compared with a fine-tuned Resnet50v2 reference CNN. Additionally, we focus on keeping the framework's complexity and computing requirements as low as possible.

3.1. Grayscale and color malware images

As a part of the malware visualization strategy employed in the study, malware files are converted into grayscale and color images to extract distinguishing features. To generate grayscale malware images, the method employed by Anandhi et al. (2021) is utilized where firstly, the binary code of the malware is translated into hexadecimal format which is subsequently represented into binary code. This step enables a more granular representation of malware executable. The binary code is further segmented into bytes by reading 8 bits at once. Each byte is thereafter mapped to a grayscale value in the range 0–255 (following the natural mapping of 8 bit integers). Here 0 represents black and 255 represents white (and the intermediate values represent various shades of grey). The grayscale values are organized in a from of 2D 256×256

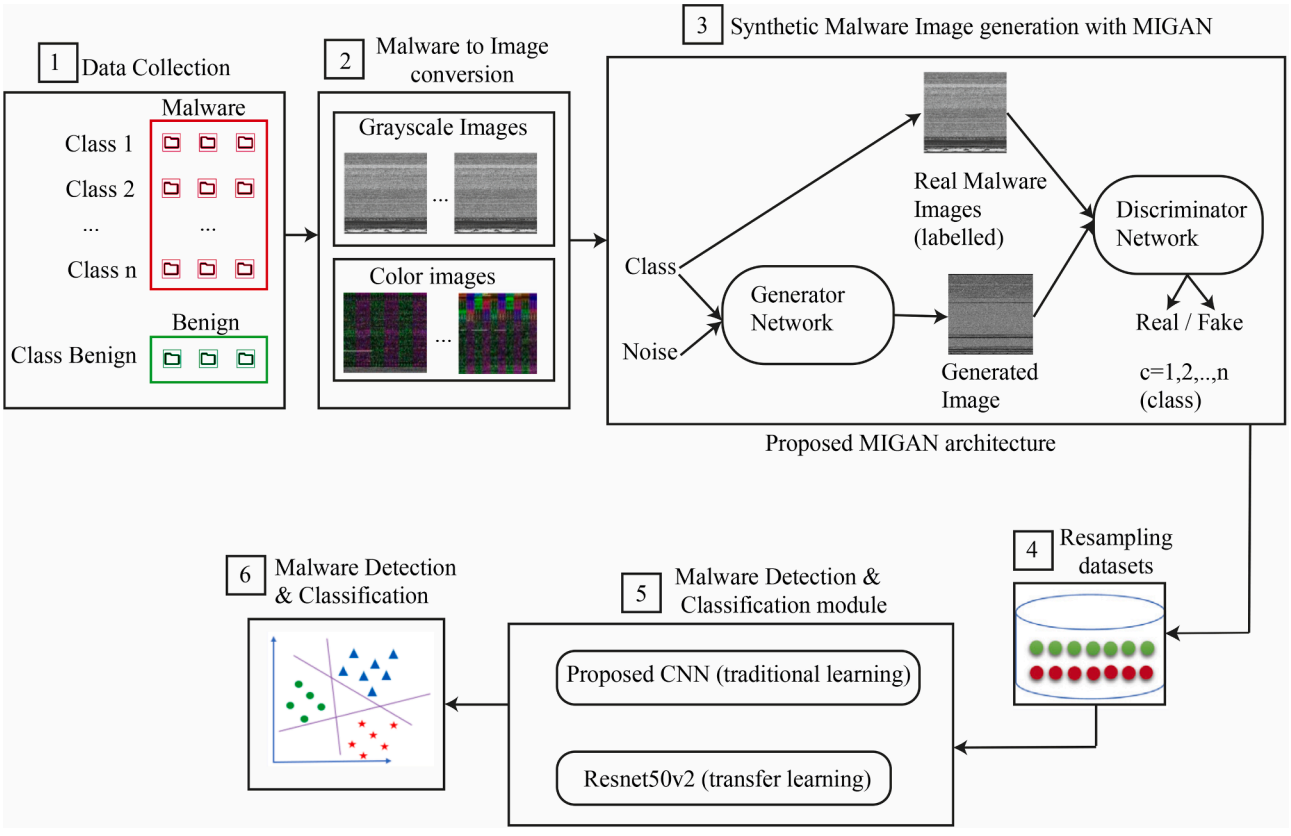


Fig. 1. Components of the proposed framework.

matrix. This size is chosen to create an image which has rich structural information. Fig. 2 shows the process of malware to grayscale image transformation employed in the study. Fig. 3 showcases a selection of malware pictures from the Malimg (Nataraj et al., 2011) dataset that illustrate the visual resemblances across samples of the identical malware families.

The process of grayscale malware image generation is followed by color malware image generation. To perform this task, the method used by Vasan et al. (2020a) is utilized, wherein the malware binary is firstly read and the data is segmented into bytes which contain the raw information to construct the color image. Thereafter the data is normalized by converting the raw byte values into a range suitable for color representation such as 0 to 255. The normalized bytes are subsequently arranged into a 3D matrix form where each coordinate in the matrix is represented by three bytes. These bytes correspond to the height, width and RGB color channel respectively. This 3D structure captures the color and spatial information of the malware in a uniform manner. If the total number of bytes does not form a perfect cube then padding bytes may be added to the data. The 3D matrix is thereafter treated as a color image with the three dimensions representing the height, width and RGB color channel. The process of malware to color image transformation is shown in Fig. 4. The samples from the VirusShare repository that were utilized to create the dataset of colored malware images are shown in Fig. 5. After the generation of samples, these samples are adjusted to fit the input dimensions of the neural networks utilized in the current study.

3.2. Generating malware images and resampling

Previous studies indicate the usage of GANs in scenarios including image synthesis, video formation, picture style transfer and image completion etc. It can be noted that GANs operate by learning the density function and assessing the distribution of the training set. A GAN structure typically consists of a Generator and a Discriminator, which simultaneously trains with a min–max objective function. While the discriminator seeks to correctly identify synthetic and real samples, the generator aims to generate realistic samples to trick the discriminator. A random vector with noise from a particular differential function is first provided as input to the generator, which strives to make the vector more like the real samples with each iteration. A neural network is used as the discriminator, and it receives input samples from both original and synthesized samples. Then it returns a probability that is equal to one for real data and near to zero for synthetic data. The cross-entropy function is used to determine the discriminator network's error rate, which is then back-propagated to the generator and discriminator structures to adjust the weights. The GAN objective function is depicted in (1) where G and D are the generator and discriminator respectively. Also, p_r and p_g refers to the probability distribution of real and generated data. The discriminator seeks to maximize the objective function, whereas the generator seeks to minimize it. According to intuition, this competition ought to strengthen both models more than if they had each been trained independently using only the training data that were available.

$$\min_G \max_D V(D,G) = E_{x\ p_r}[\log D(x)] + E_{\tilde{x}\ p_g}[\log(1 - D(G(z)))] \quad (1)$$

However, multiclass data cannot be processed by a regular GAN. To develop MIGAN for the task of malware image synthesis, inspiration from auxiliary-classifier GAN (AC-GAN) (Odena et al., 2017) was taken which is an improved kind of GAN that incorporates a class label in the generator model, as the current study deals with multiclass data. In order to clearly explain the working of ACGAN, we must specify a few details about yet another GAN variant- CGAN (Conditional Generative Adversarial Network) because ACGAN is a modified variant of CGAN. The CGAN is a form of GAN that uses a generator model to conditionally generate images. If a class label is available, image production can be dependent on it, enabling the targeted generation of images of a specific type. The Auxiliary Classifier GAN is a modification of the conditional GAN that modifies the discriminator to estimate the class label of an image rather than accept it as input. As a result, the training process is stabilized, larger, high-quality images may be produced, and a depiction in the latent space that is unrelated to the class label can be learned. However, a regular ACGAN may face issues while dealing with complex malware images therefore, we develop our own GAN variant MIGAN, to specifically deal with malware images which also inherits the ability of ACGAN to generate class specific samples. The structure of MIGAN is illustrated in Fig. 6 and the specific details of generator and discriminator are given in Table 1 and Table 2 respectively.

Each created sample is given the corresponding class label c and noise z by the MIGAN. The generator G utilizes both of these to generate $X_{fake} = G(c,z)$ images. A probability distribution over the class labels and sources is provided by the discriminator D as given in (2).

$$P(S|X),\ P(C|X) = D(X) \quad (2)$$

The objective function is formed using the log likelihood of source class L_S and correct class L_C where L_S and L_C are given in (3) and (4).

$$L_C = E[\log P(C = c|\ X_{real})] + E\left[\log P\left(C = c|X_{fake}\right)\right] \quad (3)$$

$$L_S = E[\log P(S = \text{real}|\ X_{real})] + E\left[\log P\left(S = \text{fake}|X_{fake}\right)\right] \quad (4)$$

The MIGAN generator tries to maximize $L_C - L_S$ and discriminator tries to minimize $L_C + L_S$. Through the distribution of random points on a latent space, the MIGAN generator creates images. The latent space basically consists of noise derived from a Gaussian distribution. The model further takes the class label as a parameter. The MIGAN architecture incorporates two sequential models for generator and discriminator respectively. In the generator model, several transpose convolutional layers are utilized which are also called up-conv layers. These layers are interspersed with batch normalization layers. These layers work together to transform the input noise vector into a 3-channel image having 224×224 dimensions. Similarly, the discriminator structure comprises of conventional convolutional layers that serve the purpose of diminishing the height and width of the feature maps. This is followed by two final layers (which are inspired from ACGAN structure): the adversarial layer, which discriminates between real and fake samples and the auxiliary layer which classifies the image into 26 distinct classes (which is based on the initial number of classes). This layout made it easier and more efficient for the model to grasp features from real malware images. Experiments with different learning rates were performed for the generator and discriminator structures. By adjusting the dropout layers and adding more data, the overfitting problem was also tracked. We experimented with multiple dropout values and settled with a dropout value of 0.5 during training. After each convolution layer, the ReLU- rectified linear unit activation function was applied.

The discriminator model is layered on top of the generator model and the discriminator's layers are initially set to be untrainable. As a result,

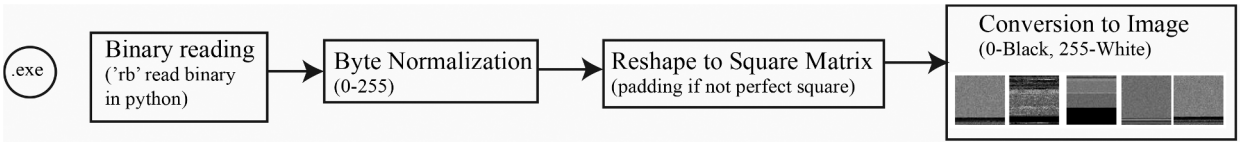


Fig. 2. Process of malware exe to grayscale image transformation.

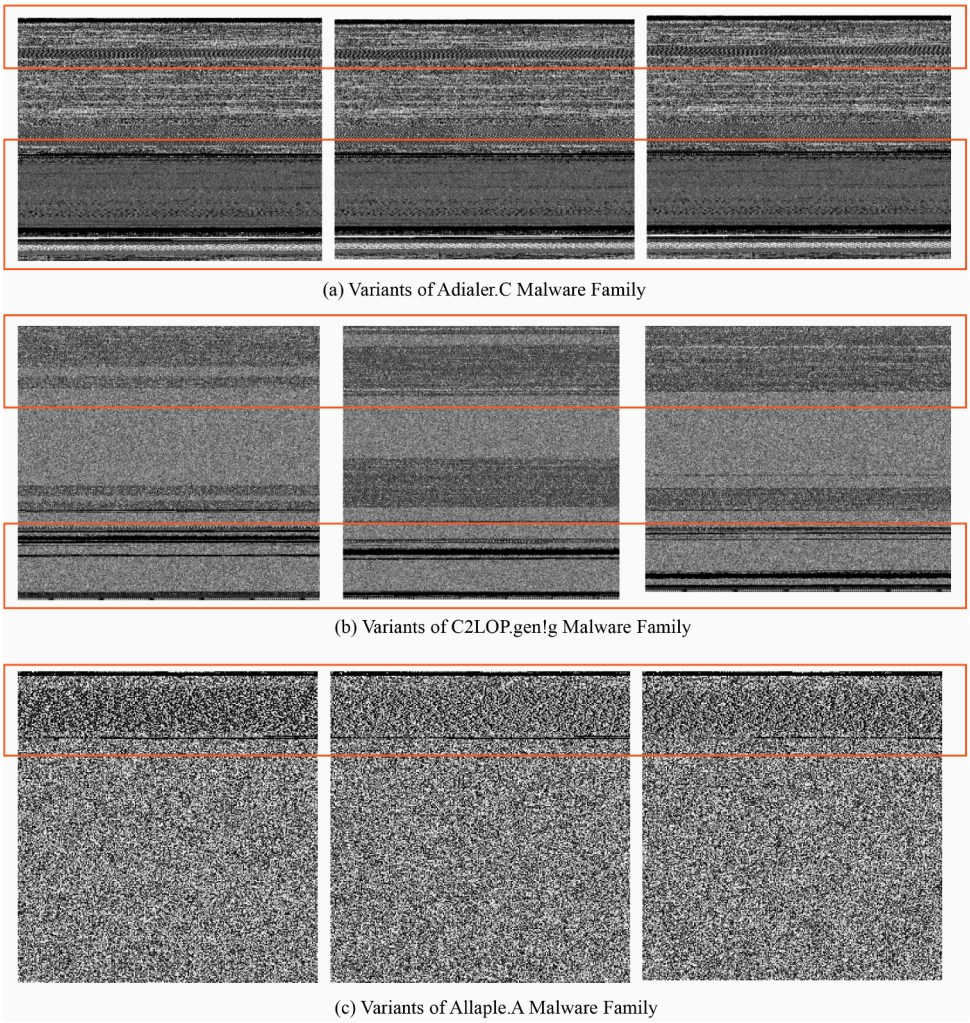


Fig. 3. Similarity in structures between variants of the same grayscale malware families in Malimg (Nataraj et al., 2011) dataset. Orange rectangles represents similarity in the structure of malware samples belonging to same malware family.

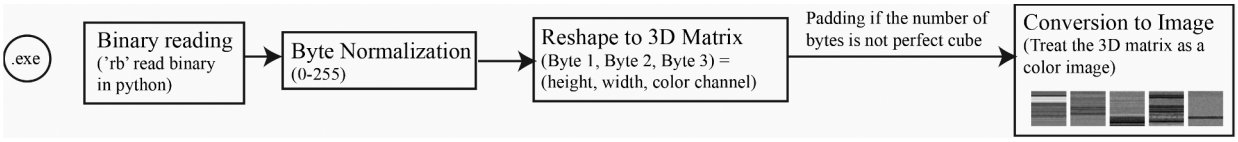


Fig. 4. Process of malware to color image conversion employed in the study.

the discriminator updates only the generator. The images were resized and normalized as part of the image preparation procedure. (Normalization is a procedure that alters the range of pixel values. Its goal is to change an input image into a selection of pixel values that are more recognizable.) The optimization function used in the manuscript is Adam which is straightforward to use, uses sparse gradients, uses little memory and is computationally less intensive. Adam is therefore the ideal candidate for the model's optimization. Batch size = 64, learning rate = 0.01, beta = 0.5 (beta is the momentum of the Adam optimizer) and number of epochs = 1000 are the parameters utilized for training. The model was trained for about five hours. It can be noted that the newly generated malware images resembled original malware images and therefore these are potentially similar to previously unseen malware samples that may get discovered in the near future. Furthermore, the incorporation of previously unseen samples in training set results into a more robust framework as the network is trained on a wide variety of samples. Following the task of synthetic malware image synthesis, the resampling operation was performed to balance the datasets.

Resampling is a fundamental technique used frequently in data preprocessing tasks in order to address class imbalance and sample distribution issues. In the context of this study, the goal of resampling operation (Fig. 1) is to align class distribution aiming for a balanced distribution closely approximating to a single value which is large enough to make data reasonably diverse, but not too large to make the training overly expensive. After initially starting with a smaller number of samples in each class (500) and observing that the number can further be increased to increase the diversity in malware images in each class, the number of samples in each class was increased to 1000 (keeping in mind the training time and computational cost) as shown in Table 4. For the datasets utilized in the manuscript, the resampling process involved oversampling and undersampling operations. In the case of oversampling, the synthetic malware images generated by MIGAN were added to the minority classes, thereby increasing the number to 1000 samples in each class and similarly in the case of

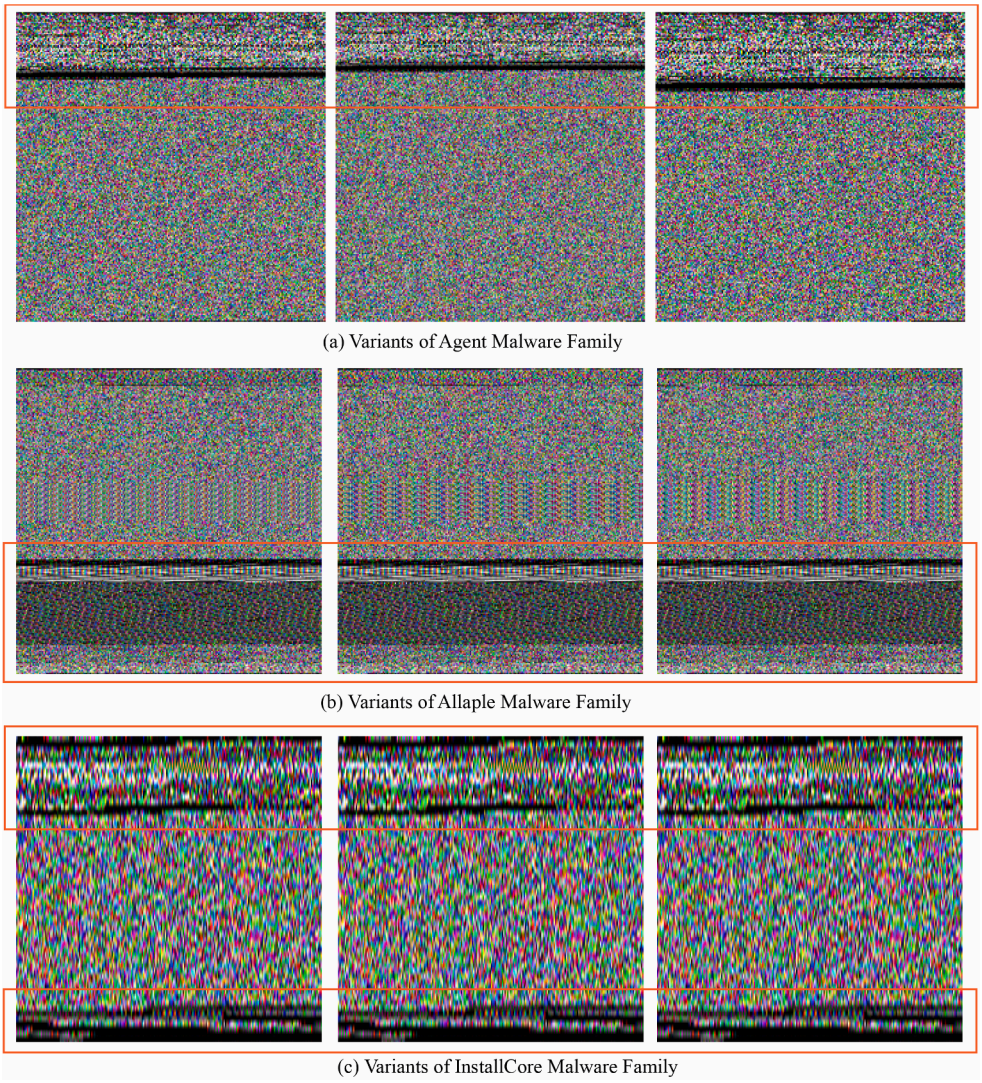


Fig. 5. Examples of color-mapped malware images from the VirusShare malware collection. Orange rectangles represents similarity in the structure of malware samples belonging to same malware family.

undersampling, randomly selected samples in each majority class is brought to 1000. This results in the creation of balanced datasets and providing a more equitable learning ground for the classifiers which may potentially improve generalizability and predictive accuracy (Buda et al., 2018).

3.3. Custom deep CNN

Building upon the MIGAN architecture described in the previous section, which is used to reduce class imbalance by resampling malware image datasets, this section delves into the details of the proposed deep CNN for the task of multiclass malware image classification. Convolutional Neural Networks are a specialized class of deep neural networks inspired by the biological vision system of animals. CNNs have found widespread success in various visual recognition tasks such as image categorization and object detection due to its inherent ability to effectively isolate discriminating features from images.

Fig. 7 illustrates the proposed CNN's architecture diagram and Table 3 presents the details of each layer in the deep CNN. The proposed deep CNN architecture consists of two fully connected layers and 13 convolutional layers with the aggregate number of layers totaling to 33. Low-level features are detected by the initial layers and high-level features are detected by the subsequent layers. The fully connected layers are positioned toward the end of a CNN to combine all the distinguishing features discovered by the earlier layers and produce a particular output. Previous research by Gibert et al. (2020) and Sudhakar and Kumar (2021) imply that a broader receptive field can be produced by layering multiple smaller convolution kernels. This enhances the depth of the network and the quantity of nonlinear transformations in comparison to a deeper convolutional network, enabling more complicated patterns to be learned with fewer network parameters. The CNN structure is designed to process images of the size 128×128 and incorporates Conv2D layers with gradually increasing filter numbers in order to effectively capture malware specific hierarchical features. The batch normalization layers are used to stabilize the learning process by normalizing the activations. On the other hand, the MaxPool layers enhance the robustness of the model by providing downsampling capabilities. The inclusion of dropout layers at different intervals serves as a mechanism to mitigate the overfitting issues. The output layer is intrinsically linked to the last fully connected layers. Malware sample classification into the relevant classes is performed using the final output layer. According to the number of classes that need to be classified, the output layer's neuron count is appropriately set. Twenty five output neurons make up the convolutional neural network trained to recognize malware samples from the Malimg dataset (Nataraj et al., 2011) which consists of 25 classes. The custom Windows malware dataset utilize 26

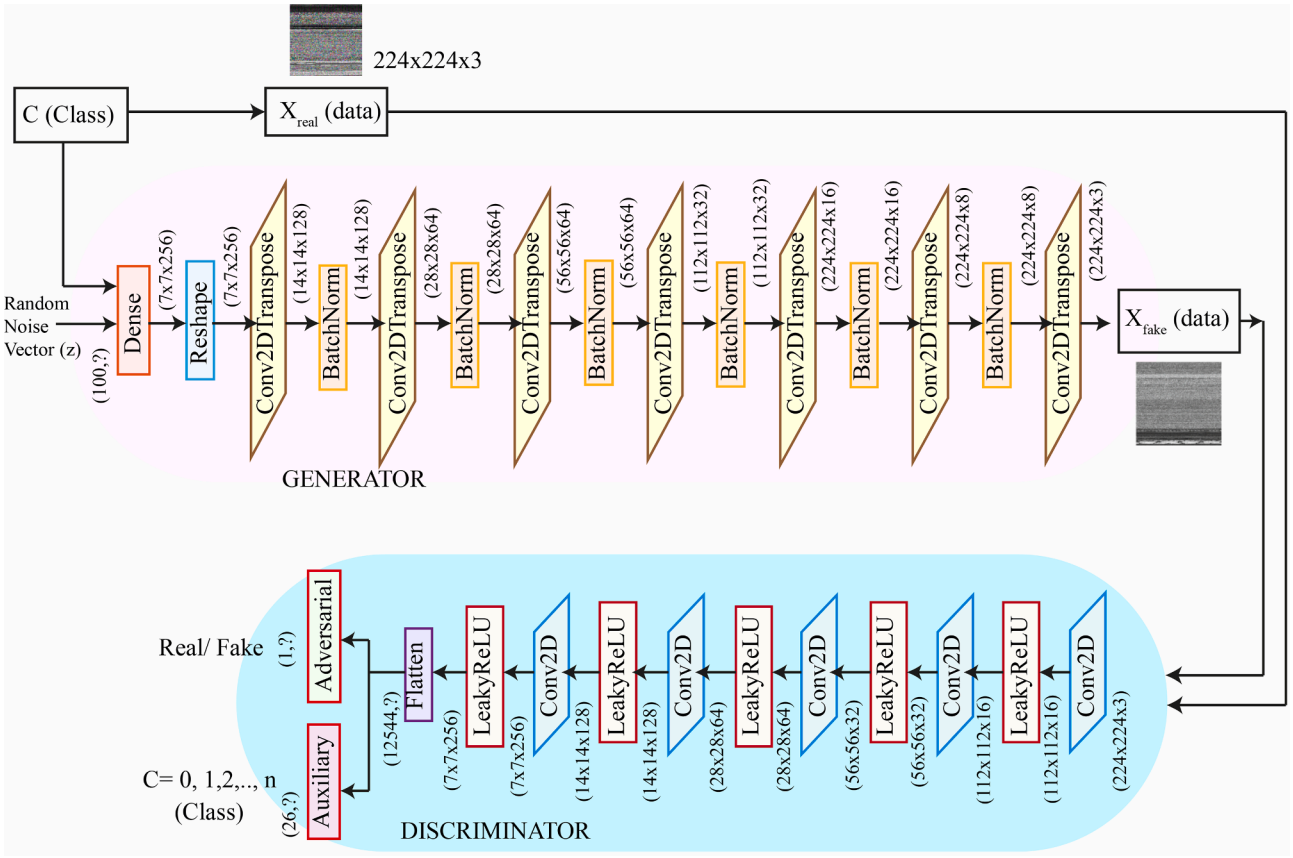


Fig. 6. Structure of the Generator and Discriminator network (MIGAN) for the task of malware image synthesis.

Table 1
Proposed generator structure used for the task of malware image synthesis.

Layer Number	Layer Name	Input Dimensions	Output Dimensions	Filter Size	Stride	Padding	Number of Filters	Total Parameters	Trainable Parameters	Non-Trainable Parameters
1	Dense	(100,?)	(7×7×256)	–	–	–	–	1,254,400	1,254,400	0
2	Reshape	(7×7×256)	(7×7×256)	–	–	–	–	0	0	0
3	Conv2DTranspose	(7×7×256)	(14×14×128)	(5,5)	(2,2)	'same'	128	819,200	819,200	0
4	BatchNorm	(14×14×128)	(14×14×128)	–	–	–	–	512	512	0
5	Conv2DTranspose	(14×14×128)	(28×28×64)	(5,5)	(2,2)	'same'	64	204,800	204,800	0
6	BatchNorm	(28×28×64)	(28×28×64)	–	–	–	–	256	256	0
7	Conv2DTranspose	(28×28×64)	(56×56×64)	(3,3)	(2,2)	'same'	64	36,864	36,864	0
8	BatchNorm	(56×56×64)	(56×56×64)	–	–	–	–	256	256	0
9	Conv2DTranspose	(56×56×64)	(112×112×32)	(3,3)	(2,2)	'same'	32	18,432	18,432	0
10	BatchNorm	(112×112×32)	(112×112×32)	–	–	–	–	128	128	0
11	Conv2DTranspose	(112×112×32)	(224×224×16)	(3,3)	(2,2)	'same'	16	4624	4624	0
12	BatchNorm	(224×224×16)	(224×224×16)	–	–	–	–	64	64	0
13	Conv2DTranspose	(224×224×16)	(224×224×8)	(3,3)	(1,1)	'same'	8	1160	1160	0
14	BatchNorm	(224×224×8)	(224×224×8)	–	–	–	–	32	32	0
15	Conv2DTranspose	(224×224×8)	(224×224×3)	(3,3)	(1,1)	'same'	3	219	219	0

neurons (25 malicious classes and 1 benign class) in the final layer.

The significance of the proposed custom deep CNN lies in its capability to provide a feature rich and hierarchical representation specifically suited for multiclass malware image classification task. The inclusion of Conv2D layers enables the network to learn at several levels of abstraction, whereas the Maxpool and Dropout layers ensure increased robustness and generalization. The significance of batch normalization lies in its ability for more rapid and reliable network convergence. The custom deep CNN is efficient and presents a reasonably balanced design across layers. This design allows easy modifications like addition of more layers or filters to suit diverse datasets and tasks.

3.4. Hybrid loss function

Malware images possess distinct types of textural, color and morphological features, which are linked with the underlying malware family type. Hence, it is essential to handle these types of images separately, unlike other image data. Malware image classification task can be tricky since the deep features learned by the neural network can sometimes lack sufficient discriminative power. Therefore, a hybrid loss function is proposed along with the Softmax loss function which is generally utilized in classification tasks. The idea of the hybrid loss function is to improve the stability and discriminative power of deeply learned features and combining the benefits of feature vector normalization, margin regulation followed by decreasing intraclass distance and

Table 2
Proposed discriminator structure used in the study.

Layer Number	Layer Name	Input Dimensions	Output Dimensions	Filter Size	Stride	Padding	Number of Filters	Total Parameters	Trainable Parameters	Non-Trainable Parameters
1	Conv2D	(224×224×3)	(112×112×16)	(3,3)	(2,2)	'same'	16	448	448	0
2	LeakyReLU	(112×112×16)	(112×112×16)	–	–	–	–	0	0	0
3	Conv2D	(112×112×16)	(56×56×32)	(3,3)	(2,2)	'same'	32	4640	4640	0
4	LeakyReLU	(56×56×32)	(56×56×32)	–	–	–	–	0	0	0
5	Conv2D	(56×56×32)	(28×28×64)	(3,3)	(2,2)	'same'	64	18,496	18,496	0
6	LeakyReLU	(28×28×64)	(28×28×64)	–	–	–	–	0	0	0
7	Conv2D	(28×28×64)	(14×14×128)	(3,3)	(2,2)	'same'	128	73,856	73,856	0
8	LeakyReLU	(14×14×128)	(14×14×128)	–	–	–	–	0	0	0
9	Conv2D	(14×14×128)	(7×7×256)	(3,3)	(2,2)	'same'	256	295,168	295,168	0
10	LeakyReLU	(7×7×256)	(7×7×256)	–	–	–	–	0	0	0
11	Flatten	(7×7×256)	(12544,?)	–	–	–	–	0	0	0
12	Dense (Adversarial Layer)	(12544,?)	(1,?)	–	–	–	–	12,545	12,545	0
13	Dense (Auxiliary Layer)	(12544,?)	(26,?)	–	–	–	–	326,170	326,170	0

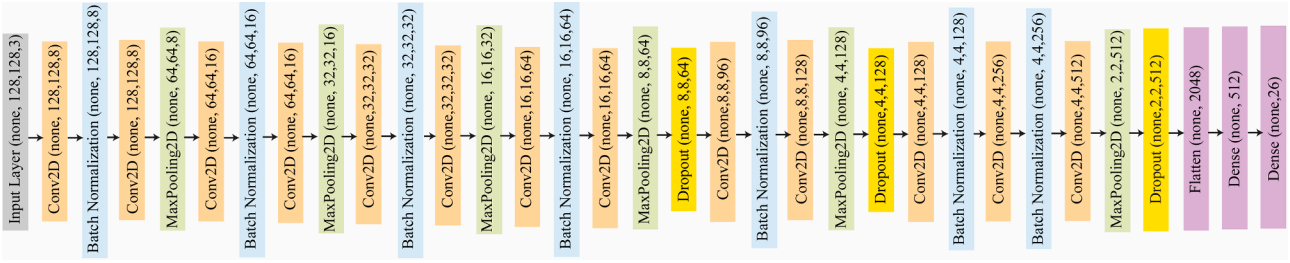


Fig. 7. Proposed custom deep CNN for malware image classification.

increasing interclass distance. The following notation is utilized:

(f_i): the feature vector of sample (x_i) before normalization

$(f_i^{'} = \frac{f_i}{||f_i||_2})$: the feature vector of sample (x_i) after normalization

(c_{y_i}): the centroid of the class of sample.(x_i)

(y^*): the class label of the sample.

(m): the margin term.

(L_f): the loss calculated by the hybrid loss function

(L_{final}): the final loss function that combines the softmax loss $(L_{softmax})$ and the hybrid loss.(L_f)

(λ): the weight factor for combining the softmax loss and the hybrid loss.

(y_j): the class label of sample.(x_j)

The most common loss function typically employed in a C-class classification problem in a deep learning framework is SoftMax. In general, if the training set is represented by $\{(x_i, y_i)\}_{i=1}^{N}$ where $y_i \in \{1, 2, 3, \cdots, C\}$ is the label of x_i where x_i represents any raw input sample, softmax loss is presented in (5).

$$L_{softmax} = \frac{1}{N}\sum_{i=1}^{N} \log \frac{e^{W_{y_i}^T f_i + b_{y_i}}}{\sum_{j=1}^{C} e^{W_j^T f_i + b_j}} \tag{5}$$

In the function, $f_i \in \mathbb{R}^d$ is a feature vector for x_i learned by the neural network. The weight vector for the jth category is represented by $W_j \in \mathbb{R}^d$ while the associated bias term is represented by b_j. The probability distribution of each example falling into each category is determined by the SoftMax loss function. Although the SoftMax loss function can compensate the classification well, it overlooks the relationships pertaining to inter-class and intra-class. Therefore, a hybrid loss function is utilized to decrease the distances of intra-class samples and increase the distances of inter-class samples in order to increase the distinguishing ability of the deeply learned features. The starting point in the hybrid loss function is to normalize the feature vectors. This task ensures that all feature vectors have same scale. Firstly, each feature vector (f_i) is normalized by its L2 norm. The resulting feature vector $(f_i^{'})$ (presented in (6)) has the same direction but a scale of 1.

$$f_i^{'} = \frac{f_i}{||f_i||_2} \tag{6}$$

As discussed earlier, the aim is to minimize the maximum intraclass distance, which is a cosine distance (shown in (7)) where $(f_i^T c_{y_i})$ is the dot product of (f_i) and (c_{y_i}) which is similar to the cosine of the angle between (f_i) and (c_{y_i}) in feature space.

$$\max_i \left(1 - f_i^T c_{y_i}\right) \tag{7}$$

Furthermore, the study aimed to maximize the minimum distance between a sample from another class and the class center, referred to as the interclass distance (shown in (8)).

$$\min_{j:y_j \neq y^*} \left(1 - f_j^T c_{y^*}\right) \tag{8}$$

The margin term (m) is introduced which ensures that the intraclass distance is lesser than the interclass distance by atleast a value (m). Thereafter, the intraclass distance and interclass distance is combined with the margin term to form the loss function (L_f) as shown in (9).

$$L_f = \left(\max_i \left(1 - f_i^T c_{y_i}\right) + m - \min_{j:y_j \neq y^*} \left(1 - f_j^T c_{y^*}\right)\right)_+ \tag{9}$$

In (9), the $((\,)_+)$ denotes the positive part of the expression, i.e., $((x)_+ = \max(0, x))$. Thus, a non-negative constraint is introduced to the loss. Subsequently, the Softmax (which handles the classification task) is combined with the loss obtained by the hybrid loss function (L_f), weighted by a factor (λ), shown in (10).

Table 3
Details of each layer in proposed custom deep CNN for malware image classification.

Layer No'	Layer Name	Input Dim	Output Dim	Filter Size	Stride	Padding	No of Filters	Total Params	Trainable Params	Non-Trainable Params
1	Input	(?, 128, 128, 3)	(?, 128, 128, 3)	–	–	–	3	0	0	0
2	Conv2D	(?, 128, 128, 3)	(?, 128, 128, 8)	3	1	Same	8	224	224	0
3	Batch Normalization	(?, 128, 128, 8)	(?, 128, 128, 8)	–	–	–	8	32	32	0
4	Conv2D	(?, 128, 128, 8)	(?, 128, 128, 8)	3	1	Same	8	584	584	0
5	MaxPooling2D	(?, 128, 128, 8)	(?, 64, 64, 8)	2	2	Valid	8	0	0	0
6	Conv2D	(?, 64, 64, 8)	(?, 64, 64, 16)	3	1	Same	16	1168	1168	0
7	Batch Normalization	(?, 64, 64, 16)	(?, 64, 64, 16)	–	–	–	16	64	64	0
8	Conv2D	(?, 64, 64, 16)	(?, 64, 64, 16)	3	1	Same	16	2320	2320	0
9	MaxPooling2D	(?, 64, 64, 16)	(?, 32, 32, 16)	2	2	Valid	16	0	0	0
10	Conv2D	(?, 32, 32, 16)	(?, 32, 32, 32)	3	1	Same	32	4640	4640	0
11	Batch Normalization	(?, 32, 32, 32)	(?, 32, 32, 32)	–	–	–	32	128	128	0
12	Conv2D	(?, 32, 32, 32)	(?, 32, 32, 32)	3	1	Same	32	9248	9248	0
13	MaxPooling2D	(?, 32, 32, 32)	(?, 16, 16, 32)	2	2	Valid	32	0	0	0
14	Conv2D	(?, 16, 16, 32)	(?, 16, 16, 64)	3	1	Same	64	18,496	18,496	0
15	Batch Normalization	(?, 16, 16, 64)	(?, 16, 16, 64)	–	–	–	64	256	256	0
16	Conv2D	(?, 16, 16, 64)	(?, 16, 16, 64)	3	1	Same	64	36,928	36,928	0
17	MaxPooling2D	(?, 16, 16, 64)	(?, 8, 8, 64)	2	2	Valid	64	0	0	0
18	Dropout	(?, 8, 8, 64)	(?, 8, 8, 64)	–	–	–	64	0	0	0
19	Conv2D	(?, 8, 8, 64)	(?, 8, 8, 96)	3	1	Same	96	55,392	55,392	0
20	Batch Normalization	(?, 8, 8, 96)	(?, 8, 8, 96)	–	–	–	96	384	384	0
21	Conv2D	(?, 8, 8, 96)	(?, 8, 8, 128)	3	1	Same	128	110,720	110,720	0
22	MaxPooling2D	(?, 8, 8, 128)	(?, 4, 4, 128)	2	2	Valid	128	0	0	0
23	Dropout	(?, 4, 4, 128)	(?, 4, 4, 128)	–	–	–	128	0	0	0
24	Conv2D	(?, 4, 4, 128)	(?, 4, 4, 128)	3	1	Same	128	147,584	147,584	0
25	Batch Normalization	(?, 4, 4, 128)	(?, 4, 4, 128)	–	–	–	128	512	512	0
26	Conv2D	(?, 4, 4, 128)	(?, 4, 4, 256)	3	1	Same	256	295,168	295,168	0
27	Batch Normalization	(?, 4, 4, 256)	(?, 4, 4, 256)	–	–	–	256	1024	1024	0
28	Conv2D	(?, 4, 4, 256)	(?, 4, 4, 512)	3	1	Same	512	1,180,160	1,180,160	0
29	MaxPooling2D	(?, 4, 4, 512)	(?, 2, 2, 512)	2	2	Valid	512	0	0	0
30	Dropout	(?, 2, 2, 512)	(?, 2, 2, 512)	–	–	–	512	0	0	0
31	Flatten	(?, 2, 2, 512)	(?, 2048)	–	–	–	512	0	0	0
32	Dense	(?, 2048)	(?, 512)	–	–	–	512	1,049,088	1,049,088	0
33	Dense	(?, 512)	(?, 26)	–	–	–	26	13,338	13,338	0

$$L_{final} = L_{softmax} + \lambda L_f \tag{10}$$

(λ) is a hyperparameter that controls the balance between the standard Softmax loss and the hybrid loss function. For the backpropagation algorithm during neural network training, the gradients of the loss function needs to be computed. In (11), the gradients of the hybrid function with respect to the original feature vector (f_i) are calculated.

$$\frac{\partial L_f}{\partial f_i} = \frac{\partial L_f}{\partial f_i^{'}} \frac{\partial f_i^{'}}{\partial f_i} \tag{11}$$

Here, $(f_i^{'} = \frac{f_i}{||f_i||_2})$ is the normalized feature vector. The gradient of (L_f) with respect to $(f_i^{'})$ can be computed in a similar way as in the standard triplet loss but with the cosine distance instead of the Euclidean distance. The gradient of $(f_i^{'})$ with respect to (f_i) can be computed using the formula for the derivative of a function divided by its L2 norm. For (λ) and (m), different values (using grid search technique by testing on a validation set) were experimented and $(\lambda) = 0.1$ and $(m) = 0.6$ were the finally selected values for the current work. To incorporate this task, a narrow selection of parameter values were first listed. The values for each of these parameter sets were represented by a group of Cartesian sets. Each batch of parameter values were used to train the model and choose the parameter pair that has the lowest error in the validation set to obtain the values for m and λ. In order to restrict the features with SoftMax, the learning process of (10) is created and the highest intra-class distance among all classes is taken into account from a sample to its class center and the smallest distance among other classes to that class center for each iteration is chosen. By calculating the distance between samples and class centers, we employ the hard sample mining technique to select the largest intra-class distance and the smallest inter-class distance (Fig. 8).

Table 4
Details of datasets used in the study.

Custom malware dataset					'Malimg' public malware dataset (Nataraj et al., 2011)				
Family	**Class ID**	**Original Imbalanced**	**MIGAN Resampled**	**DCGAN Resampled**	**Family**	**Class ID**	**Original Imbalanced**	**MIGAN Resampled**	**DCGAN Resampled**
Adposhel	1	150	1000	1000	Adialer.C	1	125	1000	1000
Agent-fyi	2	297	1000	1000	Agent.FYI	2	116	1000	1000
Allaple.A	3	184	1000	1000	Allaple.A	3	2949	1000	1000
Amonetize	4	106	1000	1000	Allaple.L	4	1591	1000	1000
Androm	5	248	1000	1000	Allueron.gen!J	5	198	1000	1000
AutoRun-PU	6	135	1000	1000	Autorun.K	6	106	1000	1000
BrowseFox	7	206	1000	1000	C2Lop.P	7	146	1000	1000
Dinwod!rfn	8	261	1000	1000	C2Lop.gen!g	8	200	1000	1000
Elex	9	226	1000	1000	Dialplatform.B	9	177	1000	1000
Expiro-H	10	322	1000	1000	Dontovo.A	10	162	1000	1000
Fasong	11	149	1000	1000	Fakerean	11	381	1000	1000
HackKMS.A	12	141	1000	1000	Instantaccess	12	431	1000	1000
Hlux!IK	13	328	1000	1000	Lolyda.AA1	13	213	1000	1000
Injector	14	113	1000	1000	Lolyda.AA2	14	184	1000	1000
InstallCore.C	15	221	1000	1000	Lolyda.AA3	15	123	1000	1000
MultiPlug	16	307	1000	1000	Lolyda.AT	16	159	1000	1000
Neoreklami	17	114	1000	1000	Malex.gen!J	17	136	1000	1000
Neshta	18	311	1000	1000	Obfuscator.AD	18	142	1000	1000
Regrun	19	155	1000	1000	Rbot!gen	19	158	1000	1000
Sality	20	216	1000	1000	Skintrim.N	20	80	1000	1000
Snarasite.D!tr	21	314	1000	1000	Swizzor.gen!E	21	128	1000	1000
Stantinko	22	154	1000	1000	Swizzor.gen!I	22	132	1000	1000
VBA	23	280	1000	1000	VB.AT	23	408	1000	1000
VBKrypt	24	124	1000	1000	Wintrim.BX	24	97	1000	1000
Vilsel	25	215	1000	1000	Yuner.A	25	800	1000	1000
Benign	26	173	1000	1000					
Total		5450	26,000	26,000	Total		9342	25,000	25,000

Table 5
GAN metrics comparison: Fréchet Inception Distance, Inception Score and Kernel Inception Distance for the datasets used in the study.

Type	Custom malware image dataset			Malimg dataset (Nataraj et al., 2011)		
	Inception Score	**Fréchet Inception Distance**	**Kernel Inception Distance**	**Inception Score**	**Fréchet Inception Distance**	**Kernel Inception Distance**
Original	1.90 ± 0.02	–	–	1.98 ± 0.03	–	–
DCGAN resampled	2.60 ± 0.034	3.07 ± 0.026	0.27 ± 0.02	2.82 ± 0.04	3.15 ± 0.02	0.23 ± 0.002
MIGAN resampled	2.81 ± 0.02	1.81 ± 0.49	0.11 ± 0.032	2.97 ± 0.012	1.93 ± 0.025	0.14 ± 0.02

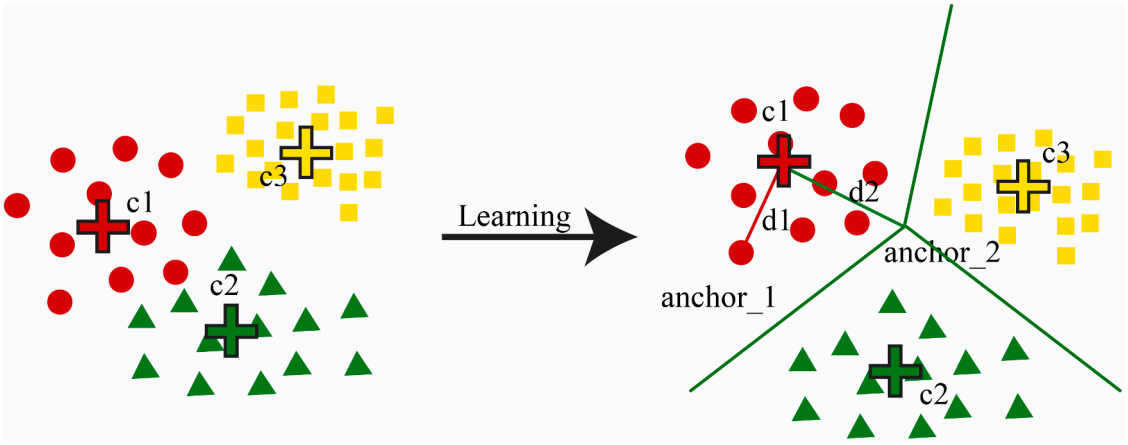


Fig. 8. Illustration of the hybrid loss function. The class center is represented by $\mathbf{c}_i$, i ∈[1,3], the maximum distance between intra-class examples is represented by d1, and the minimum distance between the negative samples and centre c1 is represented by d2.

4. Dataset and experiments

This section presents the details of experimental setup, datasets and evaluation metrics used in the study. An online-offline setup is utilized to carry out the experiments. The offline workstation employs an Intel 11th Generation core i7-1165G7 @2.80 GHz CPU, 16 GB RAM and an intel Iris XE 4 GB graphics card and the online setup consists of Kaggle compute notebook with k80 GPU, 2xvCPU and 12 GB RAM. The data collection and malware to image conversion was carried out with offline machine and the training was carried out in Kaggle compute notebook. Python 3 language was utilized with several libraries: Keras2, Tensorflow2, sklearn and matplotlib. The training networks were assessed and tweaked for optimal number of epochs. The accuracy increased sharply throughout the first 10 epochs before stabilizing and displaying minimal variations by the 50th epoch. This indicated that training can be halted near 60th epoch, which cuts down the training period. Another important parameter is batch size, which indicates how much data a neural network handles at a time. A larger batch size causes each cycle to learn more data, increasing throughput. The prediction accuracy of the neural networks was tested using various batches of input vectors. The accuracy varied as the batch sizes went from 16 to 256. It can be noted that the accuracy value steadily rose from 16 to 64, the batch size after which the accuracy values varied the least. A batch size adjustment of 64 for Malimg dataset and 32 for self-created Windows dataset was therefore selected. Table 7 shows the optimization settings required for experimental reproducibility.

4.1. Dataset details

While choosing datasets to evaluate the proposed framework, it was observed that there is no shortage of malware samples on the internet, however, updated and properly labelled malware datasets were very few. This was confirmed by observing related works in the domain of malware analysis and subsequently it was deduced that most of the recent studies still utilized extremely old malware datasets and therefore we aimed to create a new malware dataset from scratch and make it usable for the research community. To address this problem, malware samples from three prominent websites on the internet were collected: theZoo (theZoo Github, 2022), VirusShare (VirusShare.Com, 2022) and VXHeaven (Vx-Underground, 2022). After downloading the files, it was observed that several samples are duplicate due to identical filenames. To eliminate duplicates, the MD5 checksums of the malware samples were examined. Subsequently, the VirusTotal api was utilized to generate the class label and the malware samples were correctly labelled using a majority vote technique where 75 % of the antimalware in the VirusTotal api concurred on the label of the malicious sample belonging to a particular family type. This resulted in a self-created malware dataset having 5450 samples, divided into 26 different classes. It should be noted that the dataset also consists of benign samples which were collected from benign software repositories (Softonic, CNET) on the internet. Additionally, we also wanted to compare the performance of our framework against other prominent works in malware research, therefore we had to choose between two popular benchmark Windows malware datasets: Microsoft malware challenge dataset (Ronen et al., 2018) and Malimg dataset (Nataraj et al., 2011). However, the Malimg dataset was selected due to its compact size and diverse representation of samples. We carefully observed the class-composition of Malimg dataset and observed that it contains instances of malware from a number of families, including Yuner.A, VB.AT, Malex.gen!J, Autorun.K, and Rbot!gen. The malware samples are transformed into image format according to the steps discussed in the preceding section. The specifics of the two datasets are displayed in Table 4. A depiction of MIGAN generated image samples from the custom malware image dataset can be seen in Fig. 9.

4.2. Evaluation metrics

The criteria for evaluating the suggested framework's performance are as follows:

- True Positive (TP) denotes the correctly classified positive category instances.
- True Negative (TN) denotes the correctly classified negative category instances.
- False Positive (FP) denotes negative category instances that have been incorrectly classified as positive category instances.
- False Negative (FN) refers to positive category instances that have been incorrectly classified as negative category instances.

Accuracy, Precision, F1, and recall are calculated using these criteria and are stated in the formulas given below.

- The AUC is defined as the likelihood that the classifier would give a randomly selected positive sample a high value compared to a randomly selected negative sample. It has a numerical value between 0 and 1, and the nearer it goes to 1, the higher is the model's performance.
- Accuracy: the percentage of accurately predicted samples among all samples as shown in (12).

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{12}$$

- The percentage of accurately predicted malware to total predicted malware is known as precision as shown in (13).

$$\text{Precision} = \frac{TP}{TP + FP} \tag{13}$$

- The fraction of anticipated malware instances to the total number of malware instances is the recall or sensitivity value of a dataset as shown in (14).

$$\text{Sensitivity} = \frac{TP}{TP + FN} \tag{14}$$

- The weighted average of recall and precision values is called F1 Score as shown in (15).

$$\text{F1 score} = \frac{2*\text{Precision}*\text{Recall}}{\text{Precision} + \text{Recall}} \tag{15}$$

5. Results and analysis

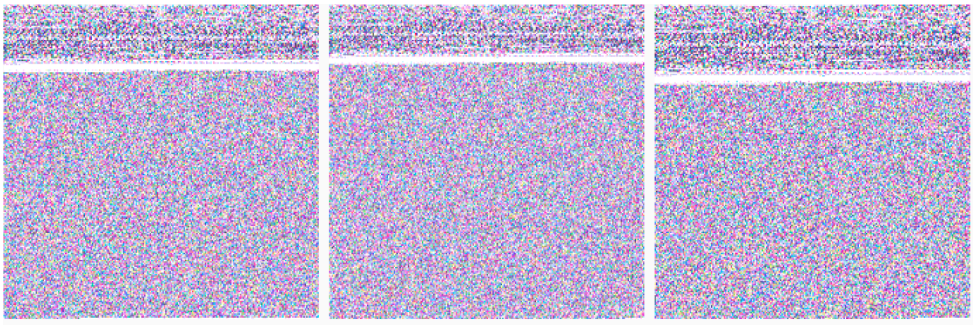
This section contains the details about the evaluative assessment of

Table 6
Comparison: MIGAN vs DCGAN vs SMOTE algorithm on classifier performance.

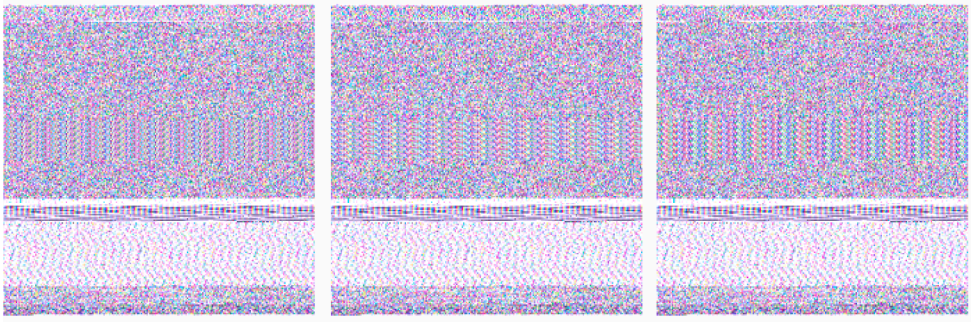
Methods	SMOTE		MIGAN Generated		DCGAN Generated	
	Accuracy	F1 Score	Accuracy	F1 Score	Accuracy	F1 Score
Logistic Regression	93.66	93.25	96.8	97.1	95.9	96.8
KNN	95.67	95.56	97.2	98.6	96.8	97.9
SVM	95.78	95.79	97.1	97.7	96.8	96.9

Table 7
Optimization parameter settings for the two datasets used in the study.

Type	Custom dataset		Malimg dataset (Nataraj et al., 2011)	
	Hyperparameters	Prediction time(s)	Hyperparameters	Prediction time(s)
Imbalanced	lr = 0.01, batch size = 32, epochs = 60	Total time = 11.09, Avg time = 0.6	lr = 0.01, batch size = 64, epochs = 60	Total time = 15.50, Avg time = 0.7
DCGAN resampled	lr = 0.01, batch size = 32, epochs = 60	Total time = 15.55, Avg time = 0.6	lr = 0.01, batch size = 64, epochs = 60	Total time = 16.56, Avg time = 0.7
MIGAN resampled	lr = 0.01, batch size = 32, epochs = 60	Total time = 12.41, Avg time = 0.5	lr = 0.01, batch size = 64, epochs = 60	Total time = 13.22, Avg time = 0.5



(a) MIGAN generated images from Agent malware family



(b) MIGAN generated images from Allaple malware family

Fig. 9. Depiction of MIGAN generated samples from custom malware image dataset used in the study.

the proposed framework in two core areas: malware image generation by MIGAN and malware classification performance. Additionally, the comparison with other related prominent works is performed at the end. As discussed in the previous sections, the malware to image conversion step generates grayscale and color malware images which are then fed into the MIGAN structure for training. The trained MIGAN model is then used to generate malware images which are used to augment and resample the datasets and also reduce class imbalance. The resampling operation results in GAN resampled datasets, where the datasets contain 1000 samples in each class (Table 4). The traditional learning based custom deep CNN is then utilized to perform the task of malware image classification on original imbalanced malware image dataset and resampled datasets, which is later compared with a pretrained Resnet50v2 reference CNN. Moreover, we also utilize a GAN structure which is similar in power to MIGAN, called DCGAN (Deep Convolutional GAN) to compare the performance of these two GANs. DCGAN has long been used for image data generation therefore, intuitively, it makes perfect sense to use it to generate malware images and compare the performance with MIGAN, which can generate class specific samples.

5.1. Validation of malware images generated by GAN

GAN models are assessed depending on the grade of the images produced, in relation to the target problem domain. One of the popular and most natural techniques to assess GANs is through visual examination of samples. However, manual evaluation poses certain limitations for e.g. it is opinionated and may include the reviewer's judgments on the model, its setup and the project goal. This leads to the usage of quantitative evaluation tools which are helpful in assessing the quality of produced images using numerical scores. The Inception Score (IS), Fréchet Inception Distance (FID) and Kernel Inception Distance (KID) are three extensively used quantitative metrics for assessing generated images (Borji, 2018). The Inception Score (IS) (Salimans et al., 2016) is a metric used to assess the standard and variety of images produced by a GAN. The IS evaluates the generated image quality by assessing how well the produced image corresponds to the distribution of actual image. The score is determined by the precision with which an Inception-v3 classifier has been trained to categorize the generated images and the variety of the generated images across multiple classes. A higher IS score denotes more quality and diversity in the generated images. The Inception Score can be calculated by multiplying the conditional entropy of the class given for the created image by the exponential of the class distribution's entropy. A metric called the Fréchet Inception Distance (FID) (Heusel et al., 2018) is used to assess how closely the distribution of generated images resembles that of genuine images. Using the mean and covariance of their Inception-v3 feature representations, the FID determines the separation between two multivariate Gaussian distributions, one for the real images and one for the produced images. A lower FID score means that the distribution of the generated images is more similar to that of the original photos. The Fréchet Inception Distance (FID) can be further elaborated as the squared Euclidean distance between two multivariate Gaussian distributions inside a feature space,

where each distribution is indicated by its mean and covariance. Heusel et al. (2018) demonstrated the improved noise robustness of Fréchet Inception Distance (FID) when compared to Inception Score (IS). Similar to the FID, the Kernel Inception Distance (KID) (Borji, 2018) is a metric used to assess how closely the distribution of real images and the distribution of synthetic images resemble each other. Instead of relying on the mean and covariance of the feature representations of the two distributions, the KID instead uses a kernel-based technique to estimate the separation between the two distributions. For particular types of data and distributions, it has been demonstrated that the KID is more resilient than the FID. The Kernel Inception Distance can be further explained as the squared maximum mean discrepancy (MMD) between the feature depictions of the actual and synthetic images, where the MMD is computed using a kernel function.

A numerical comparison between the datasets is presented in Table 5 indicating the IS, FID and KID scores for GAN generated images. For the custom-built malware image dataset, we observed an Inception Score of 1.90 ± 0.02 (mean ± standard deviation) for the original (imbalanced) images, 2.60 ± 0.034 (mean ± standard deviation) for DCGAN generated images and 2.81 ± 0.02 (mean ± standard deviation) for MIGAN generated images. For the public Malimg dataset, the Inception Score was observed to be 1.98 ± 0.03 (mean ± standard deviation) for original images, 2.82 ± 0.04 (mean ± standard deviation) for DCGAN generated images and 2.97 ± 0.012 (mean ± standard deviation) for MIGAN generated images. This result demonstrates that the MIGAN generated synthetic pictures contain clear visual artifacts and the samples are highly dense and have a wider variety across classes. We used the outputs of the pooling layer of the last hidden layer of the pretrained Inception-v3 network, which is used to generate the Fréchet Inception Distance and Kernel Inception Distance results. For the custom malware image dataset, DCGAN images achieve an FID score of 3.07 ± 0.026 and MIGAN images achieve an FID score of 1.81 ± 0.49. Similarly, for the public dataset Malimg (Nataraj et al., 2011), the DCGAN images achieved an FID score of 3.15 ± 0.02 and MIGAN images achieve an FID score of 1.93 ± 0.025. In case of KID scores, lesser value is considered better (Borji, 2018). The KID scores for both MIGAN resampled custom image dataset and Malimg dataset (Nataraj et al., 2011) are 0.11 ± 0.032 and 0.14 ± 0.02, respectively which is a good score when compared to DCGAN resampled datasets having scores of 0.27 ± 0.02 and 0.23 ± 0.002 for the custom dataset and Malimg datasets, respectively. It should be noted that the FID and KID values of MIGAN are lesser as compared to the FID and KID values of DCGAN resampled datasets. This indicates superior performance of MIGAN resampled datasets compared to DCGAN generated datasets. Evidently, the MIGAN yields the datasets that are higher-quality, more diversified and almost identical to real malware image instances. Fig. 10's t-stochastic neighbour embedding (t-SNE) plot illustrates the distributions of the real and synthetic data in custom malware image dataset. The visual and color similarities between the created synthetic images and the original malware images are clearly illustrated by the plot. The MIGAN generates a wide range of synthetic malware images, some of which are benign. Some green spots are therefore distant from the concentrated population.

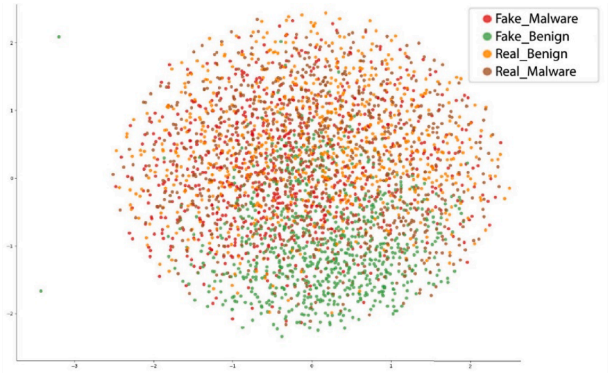


Fig. 10. T-stochastic neighbour embedding (t-sne) plot for custom malware image dataset. a variety of synthetic malware images, some of which are benign, are produced by the migan. as a result, some green areas are far from the densely populated area.

5.2. Evaluation with the SMOTE approach

Data created by using GANs is compared to a popular method in literature used for oversampling minority classes, which is called SMOTE- Synthetic Minority Oversampling Technique (Chawla et al., 2002) to assess the proposed method's performance. We compared the influence of both type of created data on a variety of classifiers. Table 6 shows the results on the self-created Windows malware dataset, which suggest that MIGAN's oversampling has a considerable impact on classifier performance improvement.

5.3. Comparison of proposed deep CNN with pretrained Resnet50v2 CNN

Transfer learning typically involves training a neural network with a dataset and then adapting the parameters of the learned network to another comparable problem. Several deep networks that have been trained on real images share a similar pattern: in the starting layers, they learn general characteristics or features that don't seem to be exclusive to one dataset or problem but are appropriate for a variety of datasets or problems. This method is advantageous because it reduces the computational load and learning time associated with learning new datasets from starting because training is costly and takes a lot of time and resources. Consequently, when the target dataset is significantly smaller than the base dataset, as it is in the current study, transfer learning can be a powerful method for training a sophisticated target network without overfitting.

Past studies in malware image research (Naeem et al., 2020; Vasan et al., 2020a) served as an inspiration for the transfer learning approach that was used to classify malware images using pretrained weights and fine-tune the deep CNN. The current work uses a popular reference CNN– Resnet50v2 to classify malware images (Fig. 11). He et al. (2016) first proposed ResNet CNN's initial version which solved the degradation issue brought on by an increase in network depth. The major goal of ResNet was to increase the network's direct connection channel, which allows the preservation of a portion of the output from the preceding network layer. Consequently, ResNet CNN was widely used to address few issues that a conventional convolutional network or a fully connected network may face, like loss of information during transmission and the issue of exploding or vanishing gradient which makes a network with a deeper depth hard to be trained. ResNet works similarly to a highway network in that it allows the initial input data to be transmitted straight to later levels. In order to create the model's pretrained version, approximately 1.5 million images from the ImageNet repository (Deng et al., 2009) were utilized. The ImageNet dataset has 1000 classification categories and the fine-tuned version of Resnetv2 CNN was customized for the task of malware image classification by using a fully connected layer consisting of 25 neurons for 25 classes (for the Malimg dataset (Nataraj et al., 2011)) and 26 neurons for custom-built Windows dataset, instead of a final fully connected layer used to classify 1000 classes of natural images. The initial weights of the pretrained Resnet50v2 CNN that was trained on the ImageNet dataset were optimized using static backpropagation technique.

For the Malimg dataset (Nataraj et al., 2011), we construct three variations of the dataset: learning on original imbalanced Malimg dataset, DCGAN generated (resampled) and MIGAN generated (resampled) Malimg datasets. It is to be noted that both DCGAN resampled and MIGAN resampled Malimg datasets are scaled up versions of the original imbalanced Malimg dataset (Nataraj et al., 2011) which after resampling contains equal number of samples in all 25 classes (i.e. 1000

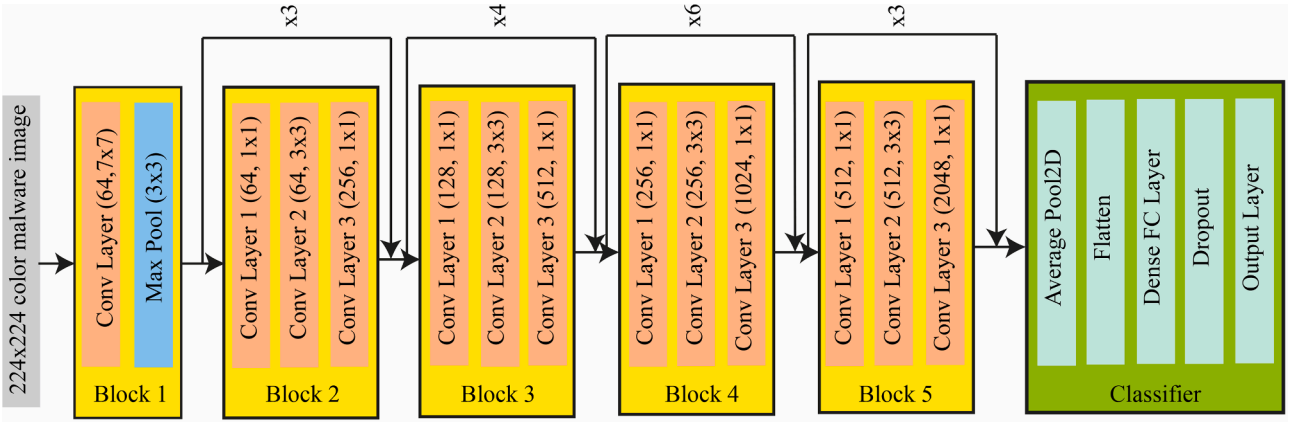


Fig. 11. Structure of Resnet50v2 CNN (He et al., 2016).

samples in each class). The accuracy and loss plots for these datasets using custom CNN and Resnet50v2 reference CNN are presented in Fig. 12(a) and (b). Fig. 12(a) presents the learning accuracy wherein it can be observed that the accuracy begins from 0.1 % and reaches nearly 99 % in the 50th epoch. The network stabilization is achieved near the 50th epoch, thereafter, the network shows least fluctuations. The training loss for these three variations of Malimg dataset can be seen in Fig. 12(b), wherein it can be observed that the loss for original Malimg imbalanced dataset begins from 1.30 % and reduces drastically to 0.50 % in the first 10 epochs. For the MIGAN resampled Malimg dataset, the

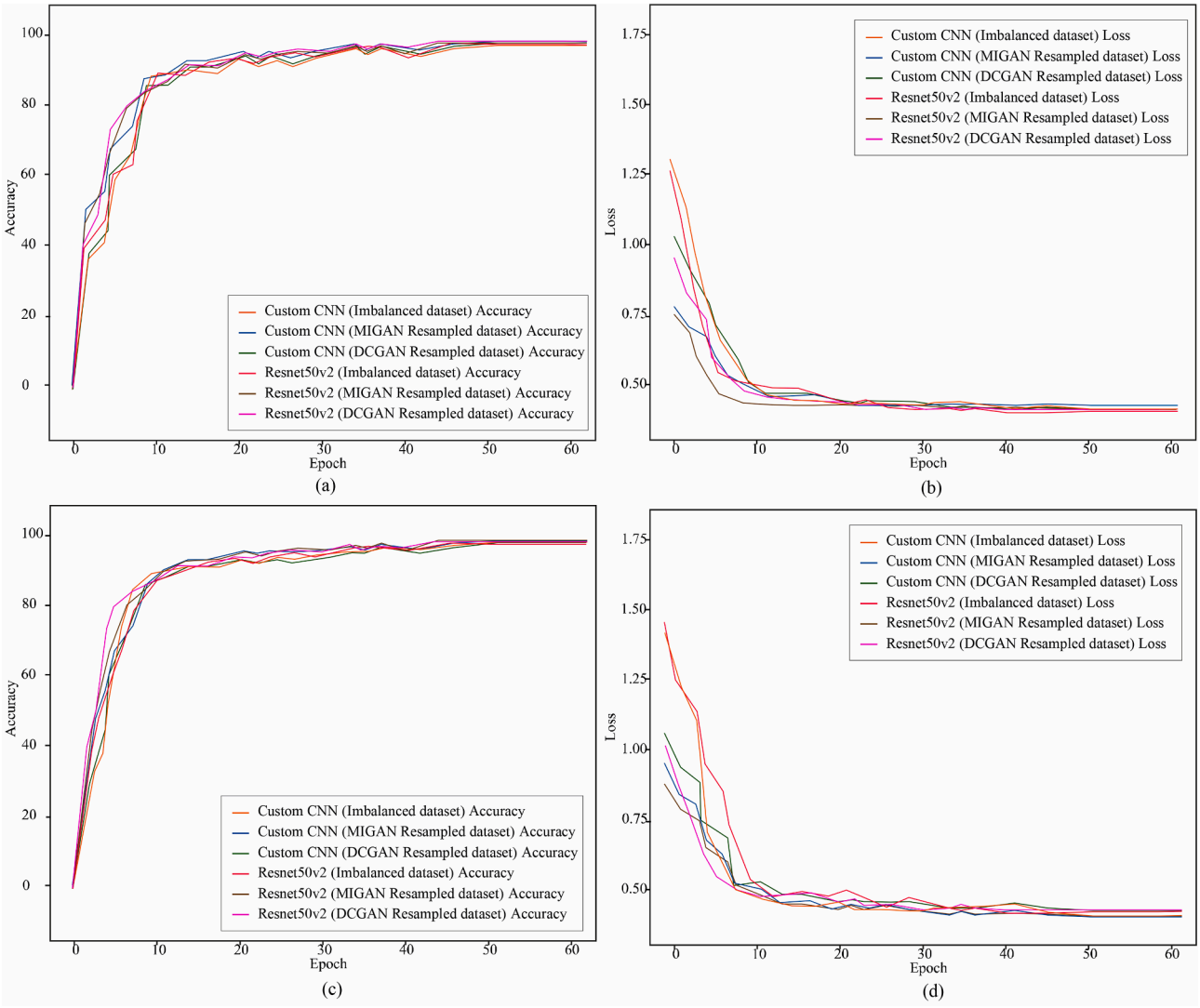


Fig. 12. Training Accuracy and Loss plots: (a) 'Malimg' dataset (Nataraj et al., 2011) training accuracy (b) 'Malimg' dataset (Nataraj et al., 2011) training loss (c) Custom malware image dataset training accuracy (d) Custom malware image dataset loss with custom deep CNN and pretrained Resnet50v2 CNNs.

custom CNN loss begins at 1.04 % and reduces to 0.47 % in the first 10 epochs. The least training loss is observed in Resnet50v2 CNN for the MIGAN resampled dataset. The training networks tends to achieve stabilization near the 49th epoch after which the networks shows least fluctuations. Similarly, for the custom-built malware image dataset, we create three variations: original imbalanced custom malware image dataset, DCGAN generated (resampled) and MIGAN generated (resampled) custom malware image datasets. Similar to the resampling operation for GAN augmented Malimg datasets, we set the MIGAN and DCGAN generated datasets to contain 1000 samples in each of the 26 classes (25 malicious + 1 benign). The accuracy and loss plots for the three datasets are shown in Fig. 12(c) and (d). From Fig. 12(c), it can be observed that the accuracy for Custom deep CNN and Resnet50v2 CNN for the three datasets begins at 0 % in the first epoch, climbs sharply to nearly 85 % in the first 10 epochs and shows stabilization by the 48th epoch. After the 50th epoch, all training networks show least fluctuations. Similarly, from Fig. 12(d) it can be observed that training loss for custom CNN and Resnet50v2 on imbalanced dataset begins from nearly 1.45 % and reduces to 0.48 % in the first 10 epochs. The Resnet50v2 CNN begins from the training loss value of 0.87 % and further reduces to 0.50 % in the first 10 epochs. The network stabilization is achieved near 47th epoch and after 50th epoch, the network shows least fluctuations. Table 8 presents a comparison of accuracy and F1 scores for the datasets used in the study.

5.4. Malware classification performance

This section presents the malware family classification results for the suggested deep CNN and finetuned Resnet50v2 reference CNN on the custom and Malimg malware image datasets. We evaluated the traditional holdout method of train-test splitting and stratified k-fold cross validation approach to determine their relevance for the datasets used in the current study. The stratified k-fold cross validation approach is usually more appropriate if the datasets have imbalanced class distribution (Buda et al., 2018). However, for the current work, the issue of class imbalance was effectively addressed using MIGAN, therefore the traditional holdout method of train test splitting was chosen. This approach is faster and requires less computational power when compared to stratified k-fold cross validation approach. In the holdout method we experimented with split ratios of 75:25 and 80:20 for training and testing and observed that the 75:25 split offered a large enough training set to avoid high bias while maintain a sufficiently large test set to prevent high variance. We additionally, observed that the 75:25 split yielded the most consistent and reliable performance across multiple datasets. This split ratio was inspired from related works in malware research (Ahmed et al., 2023; Chaganti et al., 2023) therefore it was adopted consistently across all datasets. Fig. 13 and Fig. 14 shows the results of this experiment for the custom and Malimg image datasets including classification accuracy, AUC (Area Under ROC Curve), sensitivity and F1 Score. In addition to the fine-tuned Resnet50v2 CNN, the following pretrained deep CNNs were also used in image classification (for comparison): Densenet, Inceptionv3 and VGG16. Fig. 15 displays the computed confusion matrix for Resnet50v2 CNN and custom CNN in the two datasets. Due to its deeper structure and strong connectivity with previous layers, Resnet50v2 CNN yields a very good F1-score and accuracy across all experiments. Each level of the deep network is driven to retrieve malware image patterns that differ from those of the previous layers. Improved layer interconnectivity also makes it possible to reuse feature maps from earlier layers and to create new visual representations that include data from all previous levels as well as the present layer - being used. By doing this, the problems imposed by duplicate layers—which are frequent in complicated deeper systems—are eliminated.

The training process for earlier studies like Amin et al. (2020) and Gupta and Rani (2020) necessitates a large number of malware samples to be trained. Additionally, dynamic analysis used in studies by Amer and Zelinka (2020) and J. Zhang et al. (2021) is time-heavy and computationally-intensive. Because an optimum deep Network layout was utilized, the proposed technique requires less computational power than earlier frameworks. Table 9 and Table 10 show a comparison of the current study with state of the art results. In regard to the evaluation metrics, it can be seen that the proposed methods produce excellent results.

Table 8
Comparison of custom deep CNN and Resnet50v2 CNN on the datasets used in the current study.

Dataset	Type	Architecture	Performance	
			A(%)	F(%)
Custom dataset	Imbalanced	Custom deep CNN	97.5	97.2
		Resnet50v2	97.8	97.2
	DCGAN resampled	Custom deep CNN	98.5	98.3
		Resnet50v2	98.9	98.5
	MIGAN resampled	Custom deep CNN	99.1	98.6
		Resnet50v2	99.4	98.8
Malimg (benchmark)	Imbalanced	Custom deep CNN	97.6	97.2
		Resnet50v2	97.9	98.2
	DCGAN resampled	Custom deep CNN	98.7	98
		Resnet50v2	99.1	99.1
	MIGAN resampled	Custom deep CNN	99.1	98.4
		Resnet50v2	99.5	99.5

5.5. Zero-day malware attack simulation

Malware developers persistently refine the code structure of malware variants to adapt to dynamic computational landscapes - a phenomenon termed as ‘concept-drift’ (Gibert et al., 2020). Zero-day malware, by definition, are not catalogued and hence these are not present in cybersecurity databases because zero-day malware represent newly evolved or previously unseen malware. Zero-day malware can be approximated for research purposes by employing Family-wise Holdout Evaluation (Millar et al., 2021). This method simulates a zero-day environment by withholding known malware families during neural network training and subsequently evaluating the ability of the system to detect these families as zero-day malware threats. In our study, we build upon the foundational zero-day experiment conducted in Drebin dataset original paper (Arp et al., 2014) wherein the 20 most populated malware families (determined by sample count) were sequentially excluded from the training dataset and later tested to simulate a zero-day malware scenario. Extending this method, our experiments were performed using original imbalanced Malimg (Nataraj et al., 2011) and self-created malware image dataset (Table 4), applying the same Family-wise Holdout Evaluation approach. We extend this approach to sequentially withhold all 25 malware families of Malimg (Nataraj et al., 2011) dataset and all 26 families of custom malware image dataset one by one. We chose the classification accuracy metric of proposed deep CNN and pretrained ResNet50v2 reference CNN and take the average for all sequentially excluded cases. The results presented in Table 11 illustrates the classification results for original imbalanced Malimg (Nataraj et al., 2011) and custom malware image dataset. The proposed custom deep CNN achieved an average classification score of 97.4 % whereas the finetuned ResNet50v2 reference CNN achieved slightly higher 97.6 % score on the imbalanced Malimg dataset with family wise holdout method. Similarly, for the original imbalanced custom malware image dataset, the proposed deep CNN achieved 97.6 % and the ResNet50v2 achieved 97.8 % accuracy score with family wise holdout method. The superior performance of the proposed system in the context of zero-day malware classification highlights its ability to capture distinctive features during training while also mitigating overfitting. This yields a better generalization against previously unseen malware in contrast to signature-based systems which are prone to fail in such scenarios.

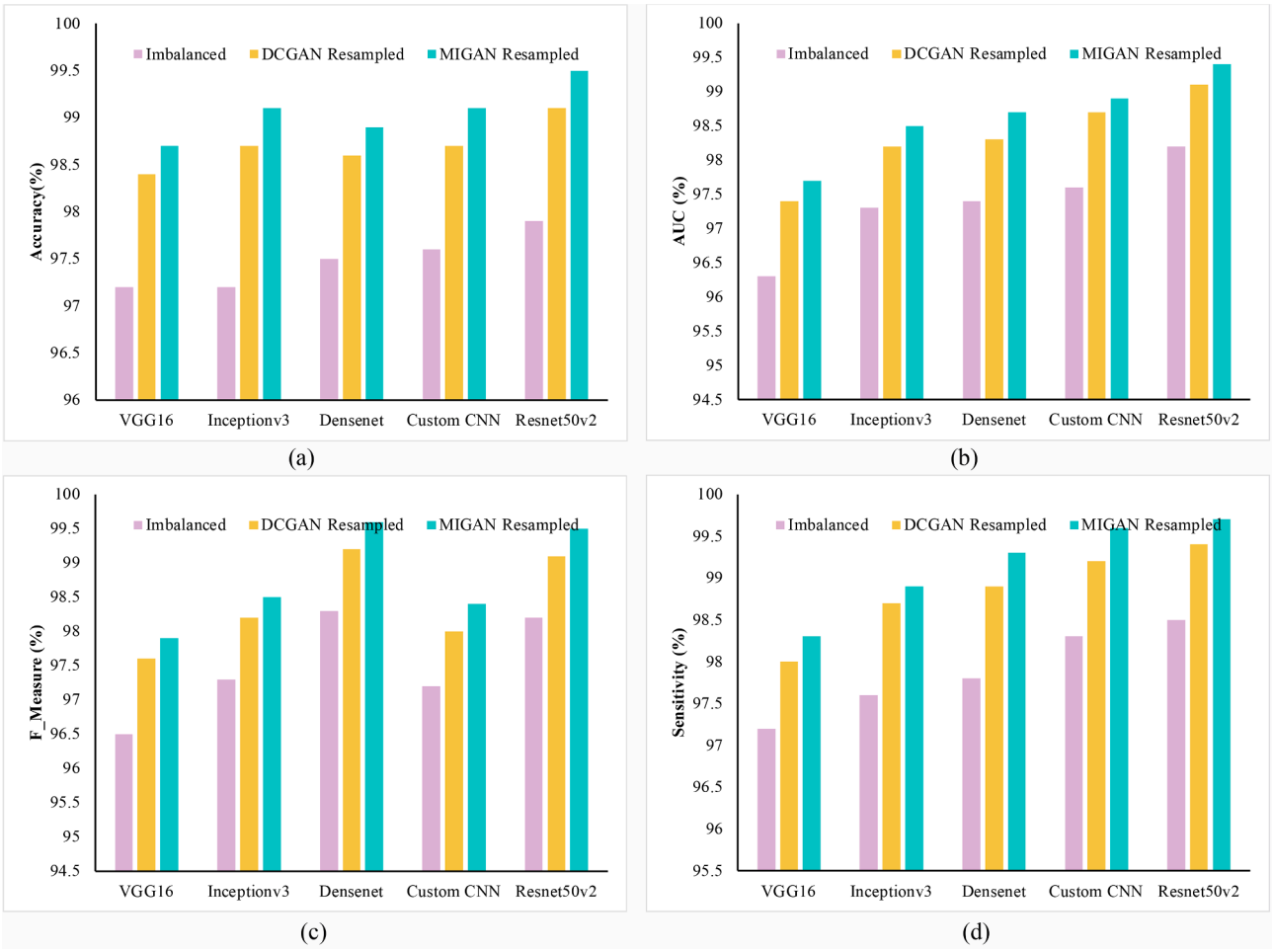


Fig. 13. Experimental scores for custom and pretrained CNNs on the Malimg dataset (Nataraj et al., 2011): (a) accuracy, (b) AUC, (c) F Measure and (d) sensitivity.

5.6. Discussion

The manuscript presents a system for generation, recognition and classification of Windows malware images using generative learning-based approach. To perform the malware image sample generation, existing malware images are fed in MIGAN structure for training. The MIGAN structure is able to produce class specific samples which are used to reduce the problem of class imbalance in Malimg (Nataraj et al., 2011) and custom malware image datasets. To perform the malware family classification task, two methods employing traditional and transfer learning are utilized. Traditional learning is carried out using a custom deep CNN structure and transfer learning is carried out using a Resnet50v2 reference CNN structure which is trained on 1.5 million images on the ImageNet dataset and it is fine-tuned to suit the task of classifying malware images into families. From Table 10, it can be inferred that the Resnet50v2 CNN-based transfer learning strategy yields marginally better results than the traditional learning approach. This seems reasonable given that deeper convnets are able to provide better depictions at each layer and the model learns a more abstract depiction of the input, whereas a shallow network has fewer hidden layers and yields representations that are less accurate when compared to a complex/deeper structure. Nevertheless, the effectiveness of traditional learning oriented custom deep CNN is more implementable while operating on computationally light systems for the reason that a shallow setup is computationally inexpensive than a complex/deeper structure. The accuracy score demonstrated by the custom CNN is 99.1 % on the Malimg (Nataraj et al., 2011) dataset, however the Resnet50v2 achieves higher score of nearly 99.5 %, which is intuitive, given the fact that ResNet50v2 is more deeper than the custom CNN. In comparison, the Inceptionv3 also demonstrates excellent accuracy score of nearly 99.1 % as compared to VGG16 (98.7 %) and Densenet (98.9 %). The highest AUC scores for the Malimg dataset (Nataraj et al., 2011) are demonstrated in the Resnet50v2 model (99.4 %) in comparison to the custom CNN (98.9 %), Densenet (98.7 %) and Inceptionv3 (98.5 %). The F1 scores for all five networks range from 97.9 % to 99.5 % where the highest score is achieved by Resnet50v2 (99.5 %). The networks demonstrate sensitivity scores of above 99 %: Densenet (99.3 %), custom CNN (99.6 %) and Resnet50v2 (99.7 %). Similarly for the self-created Windows malware image dataset, both custom CNN and Resnet50v2 demonstrate accuracy scores above 99 % with Resnet50v2 achieving 99.4 % accuracy. The AUC scores achieved by all five networks range from 98.3 % (for VGG16) to 99.7 %(for Resnet50v2). Furthermore, the highest F1 scores are demonstrated by Resnet50v2 (98.8 %) and custom CNN (98.6 %). The sensitivity score for the five CNNs under consideration is topped by Resnet50v2 (98.7 %) and custom CNN (98.3 %) whereas for VGG16, Inceptionv3 and Densenet, the sensitivity scores are close to 97.5 %.

The paper presents a framework for synthesis, detection and classification of malware samples and reducing class imbalance in malware image datasets using deep learning algorithms. However, a few limitations need to be considered. It can be observed that the proposed model was tested on public 'Malimg' dataset (Nataraj et al., 2011) and self-created Windows malware image dataset, however, the framework's performance on malware belonging to other platforms (like Android, Linux etc.) requires further experimentations. Additionally, the proposed framework showed good results with Windows malware datasets,

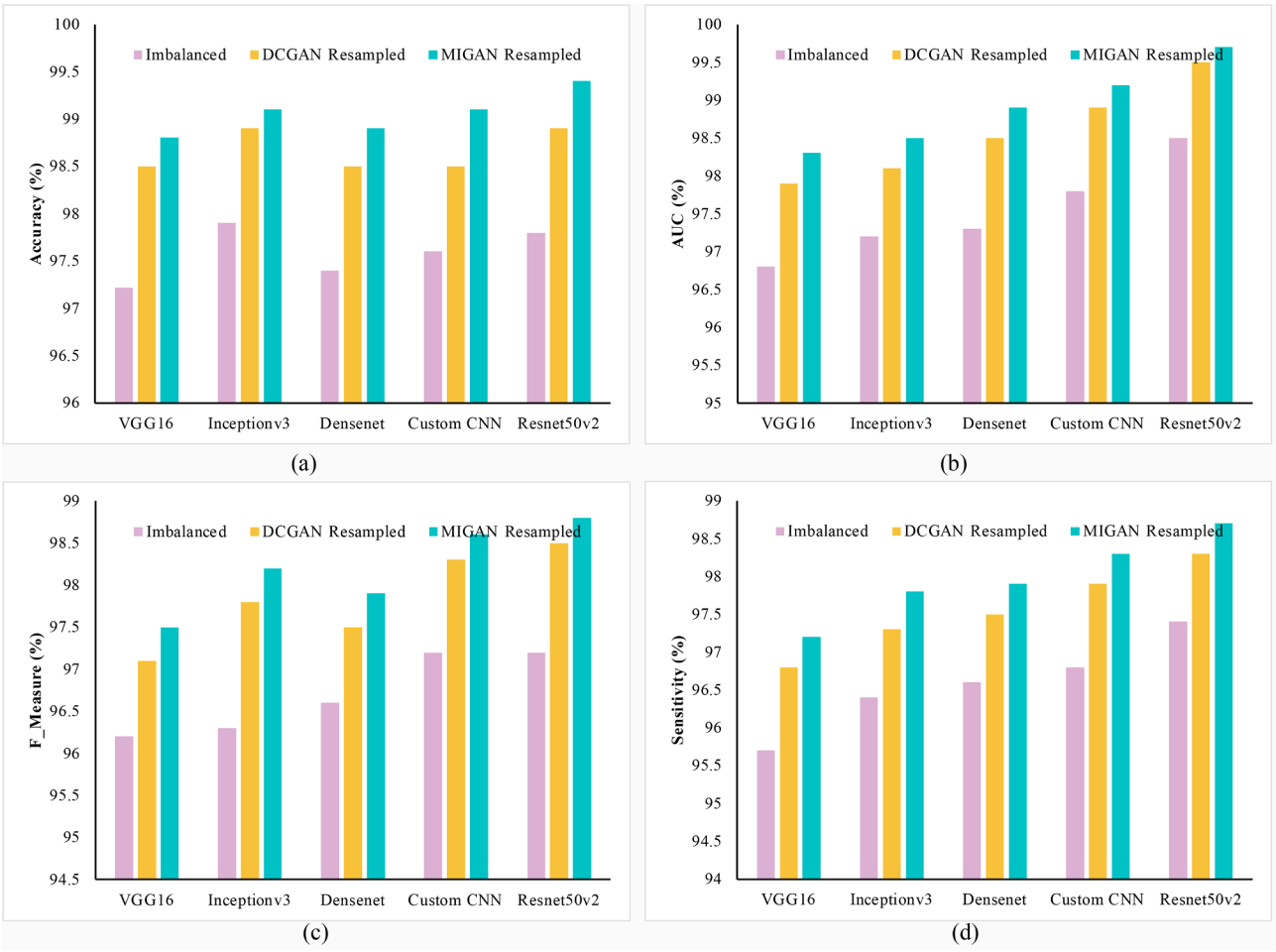


Fig. 14. Experimental scores for custom and pretrained CNNs on custom malware image dataset: (a) accuracy, (b) AUC, (c) F Measure and (d) sensitivity.

however, the ever-changing nature of malware (concept-drift), along with the use of techniques like polymorphism, obfuscation and packing may impact the performance of the framework and further experimentations are required in special cases where evolved malware are under consideration. Moreover, the scalability of the model for real-time deployment on limited computing power environments requires additional research.

6. Conclusions and future scope

In this manuscript, we proposed a GAN variant – MIGAN, for the task of Malware Image Synthesis and to solve multiple challenges in the domain of malware recognition and classification. Firstly, MIGAN solves the problem of class imbalance in self-created and public-benchmark malware image datasets. Secondly, the images generated using MIGAN resemble existing malware images, and therefore these are potentially similar to 'previously unseen' or 'zero-day' malware samples that may get discovered in the near future. As a result of training of the model on a wide range of samples, the overall robustness of our model improves. Additionally, the malware visualization-based approach adopted in the study can easily identify patterns in malware samples and can be used to classify malware samples into families. To carry out the classification task, we utilized both traditional learning and transfer learning approaches. The traditional learning-based approach involved constructing a custom deep CNN from scratch and optimizing it for the task of malware image classification. The transfer learning-based approach, on the other hand, involved utilizing a pretrained Resnet50v2 model which was trained on 1.5 million images of the ImageNet repository and it is fine-tuned to suit the task of classifying malware images. Furthermore, the work involves utilization of a hybrid loss function using a hard sample mining method, which filters readily categorized data and causes the intra-class distance to be less than the mean inter-class distance with a margin to enhance the discriminative power of the deeply learnt features. To evaluate the performance of the framework, one custom malware image dataset is utilized along with one public benchmark dataset 'Malimg'. This work performs exceptionally well when compared to other state-of-the-art research works in terms of accuracy, F1 and other metrices.

The article presents a fusion of GAN based and malware visualization-based method for generating, identifying, and classifying Windows malware images. However, this method can be further developed in the future to evaluate the classification model's defense against adversarial attacks. By inserting bytes from non-malicious executables into malware files, this experiment will change the statistical distribution of malware samples. Neural learning-based malware detectors are severely impacted by adversarial samples. On several other malware datasets, we also intend to apply additional statistical techniques like the Kullback-Leibler deviation (KLD) and Jensen-Shannon deviation (JSD) etc. In the future, we also aim to test the model for malware targeting other platforms like Android and Linux. Finally, since the model surpasses prominent deep learning approaches in the literature, it is the favored method of choice for malware generation, recognition and classification.

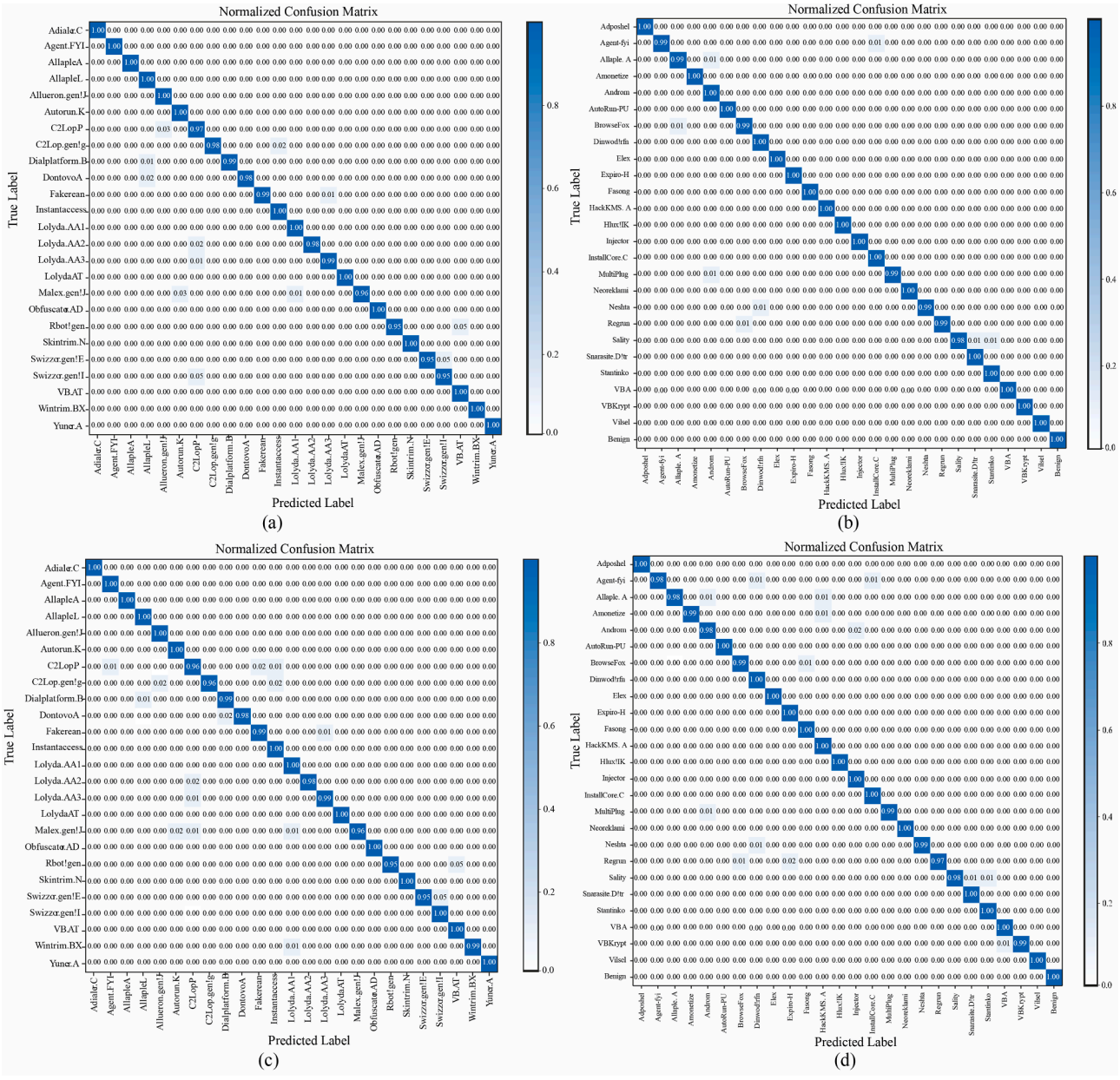


Fig. 15. Confusion matrices of Resnet50v2 CNN on (a) Malimg dataset (Nataraj et al., 2011), (b) custom Windows malware image dataset. Confusion matrices of custom-built CNN on (c) Malimg dataset (Nataraj et al., 2011) and (d) custom Windows malware image dataset.

Table 9
Comparison of different works on Malimg dataset (Nataraj et al., 2011).

Model	Accuracy (%)	F1 (%)	Train time (≈min)	Test time (≈s)
(Anandhi et al., 2021)	98.98	98.89	–	0.06
(Vasan et al., 2020a)	98.89	98.76	–	0.082
(Gibert et al., 2019)	–	94.82	–	–
Current (custom CNN)	99.15	98.42	95	0.6
Current (Resnet50v2 CNN)	99.52	99.51	185	0.6

Table 10
Comparison of different works on custom built Windows malware image dataset.

Model	Accuracy (%)	F1 (%)	Train time (≈min)	Test time (≈s)
(Stamp et al., 2021)	97.52	97.82	152	1
(Xiao et al., 2020)	98.54	98.32	160	1
(Yuan et al., 2020)	98.86	98.51	150	1.5
Current (custom CNN)	99.14	98.60	80	0.6
Current (Resnet50v2 CNN)	99.40	98.82	165	0.6

Table 11
Zero-day malware attack simulation with Family-wise holdout evaluation method (Millar et al., 2021).

Dataset	Accuracy-Proposed custom deep CNN (%)	Accuracy-ResNet50v2 (%)
Malimg (imbalanced) (Nataraj et al., 2011) average	97.4	97.6
Custom malware image dataset (imbalanced) average	97.6	97.8

Statements and declarations

Ethics approval

Not applicable.

Consent to participate

Not applicable.

Human and animal ethics

No animals or humans were harmed for the experiments in this manuscript.

Consent for publication

Not applicable.

Credit author statement

OS contributed to the study design, implementation, results analysis and preparation of the final draft. AS contributed to the design, conceptualization, review-editing and supervision. AK validated and supervised the entire work. All authors reviewed the manuscript.

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.

Acknowledgements

Not applicable.

Funding

Not applicable.

Availability of code

Code download link.

Availability of data

Malimg (Nataraj et al., 2011) dataset download.
Self-created Windows malware image dataset download.
Source malware repositories of self-created malware image dataset.
It is important to exercise caution while downloading malware binaries since it can harm computing setups.

Declaration of Generative AI and AI-assisted technologies in the writing process

No generative AI was used in this manuscript.

References

Ahmed, M., Afreen, N., Ahmed, M., Sameer, M., & Ahamed, J. (2023). An inception V3 approach for malware classification using machine learning and transfer learning. *International Journal of Intelligent Networks, 4*, 11–18. https://doi.org/10.1016/j.ijin.2022.11.005

Amer, E., & Zelinka, I. (2020). A dynamic Windows malware detection and prediction method based on contextual understanding of API call sequence. *Computers & Security, 92*, Article 101760. https://doi.org/10.1016/j.cose.2020.101760

Amin, M., Tanveer, T. A., Tehseen, M., Khan, M., Khan, F. A., & Anwar, S. (2020). Static malware detection and attribution in android byte-code through an end-to-end deep system. *Future Generation Computer Systems, 102*, 112–126. https://doi.org/10.1016/j.future.2019.07.070

Anandhi, V., Vinod, P., & Menon, V. G. (2021). Malware visualization and detection using DenseNets. *Personal and Ubiquitous Computing*. https://doi.org/10.1007/s00779-021-01581-w

Arp, D., Spreitzenbarth, M., Hübner, M., Gascon, H., & Rieck, K. (2014). Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. In *Proceedings 2014 Network and Distributed System Security Symposium. Network and Distributed System Security Symposium*, San Diego, CA. 10.14722/ndss.2014.23247.

AV-TEST Malware Statistics. (2022). Web Page accessed May 14, 2022 https://www.av-test.org/en/statistics/malware/.

Bai, Y., Xing, Z., Ma, D., Li, X., & Feng, Z. (2021). Comparative analysis of feature representations and machine learning methods in Android family classification. *Computer Networks, 184*, Article 107639. https://doi.org/10.1016/j.comnet.2020.107639

Bakour, K., & Ünver, H. M. (2021). VisDroid: Android malware classification based on local and global image features, bag of visual words and machine learning techniques. *Neural Computing and Applications, 33*(8), 3133–3153. https://doi.org/10.1007/s00521-020-05195-w

Borji, A. (2018). *Pros and Cons of GAN Evaluation Measures* (arXiv:1802.03446). arXiv. 10.48550/arXiv.1802.03446.

Buda, M., Maki, A., & Mazurowski, M. A. (2018). A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks, 106*, 249–259. https://doi.org/10.1016/j.neunet.2018.07.011

Chaganti, R., Ravi, V., & Pham, T. D. (2023). A multi-view feature fusion approach for effective malware classification using Deep Learning. *Journal of Information Security and Applications, 72*, Article 103402. https://doi.org/10.1016/j.jisa.2022.103402

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research, 16*, 321–357. https://doi.org/10.1613/jair.953

Cui, Z., Xue, F., Cai, X., Cao, Y., Wang, G., & Chen, J. (2018). Detection of Malicious Code Variants Based on Deep Learning. *IEEE Transactions on Industrial Informatics, 14*(7), 3187–3196. https://doi.org/10.1109/TII.2018.2822680

Dai, Y., Li, H., Qian, Y., & Lu, X. (2018). A malware classification method based on memory dump grayscale image. *Digital Investigation, 27*, 30–37. https://doi.org/10.1016/j.diin.2018.09.006

Darabian, H., Homayounoot, S., Dehghantanha, A., Hashemi, S., Karimipour, H., Parizi, R. M., & Choo, K.-K.-R. (2020). Detecting Cryptomining Malware: A Deep Learning Approach for Static and Dynamic Analysis. *Journal of Grid Computing, 18*(2), 293–303. https://doi.org/10.1007/s10723-020-09510-6

Darem, A., Abawajy, J., Makkar, A., Alhashmi, A., & Alanazi, S. (2021). Visualization and deep-learning-based malware variant detection using OpCode-level features. *Future Generation Computer Systems, 125*, 314–323. https://doi.org/10.1016/j.future.2021.06.032

Dehkordy, D. T., & Rasoolzadegan, A. (2021). A new machine learning-based method for android malware detection on imbalanced dataset. *Multimedia Tools and Applications, 80*(16), 24533–24554. https://doi.org/10.1007/s11042-021-10647-z

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. *IEEE Conference on Computer Vision and Pattern Recognition, 2009*, 248–255. https://doi.org/10.1109/CVPR.2009.5206848

Ding, Y., Zhang, X., Hu, J., & Xu, W. (2020). Android malware detection method based on bytecode image. *Journal of Ambient Intelligence and Humanized Computing*. https://doi.org/10.1007/s12652-020-02196-4

Escudero García, D., & DeCastro-García, N. (2021). Optimal feature configuration for dynamic malware detection. *Computers & Security, 105*, Article 102250. https://doi.org/10.1016/j.cose.2021.102250

Fang, Z., Wang, J., Geng, J., Zhou, Y., & Kan, X. (2021). A3CMal: Generating adversarial samples to force targeted misclassification by reinforcement learning. *Applied Soft Computing, 109*, Article 107505. https://doi.org/10.1016/j.asoc.2021.107505

Farrokhmanesh, M., & Hamzeh, A. (2019). Music classification as a new approach for malware detection. *Journal of Computer Virology and Hacking Techniques, 15*(2), 77–96. https://doi.org/10.1007/s11416-018-0321-2

Gibert, D., Mateu, C., & Planes, J. (2020). HYDRA: A multimodal deep learning framework for malware classification. *Computers & Security, 95*, Article 101873. https://doi.org/10.1016/j.cose.2020.101873

Gibert, D., Mateu, C., Planes, J., & Vicens, R. (2019). Using convolutional neural networks for classification of malware represented as images. *Journal of Computer*

Virology and Hacking Techniques, 15(1), 15–28. https://doi.org/10.1007/s11416-018-0323-0

Gupta, D., & Rani, R. (2020). Improving malware detection using big data and ensemble learning. *Computers & Electrical Engineering, 86*, Article 106729. https://doi.org/10.1016/j.compeleceng.2020.106729

He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep Residual Learning for Image Recognition*. 770–778. https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., & Hochreiter, S. (2018). *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium* (arXiv:1706.08500). arXiv. 10.48550/arXiv.1706.08500.

Hu, W., & Tan, Y. (2017). *Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN*. arXiv:1702.05983 [Cs]. http://arxiv.org/abs/1702.05983.

Jain, M., Andreopoulos, W., & Stamp, M. (2020). Convolutional neural networks and extreme learning machines for malware classification. *Journal of Computer Virology and Hacking Techniques, 16*(3), 229–244. https://doi.org/10.1007/s11416-020-00354-y

Kim, J.-Y., Bu, S.-J., & Cho, S.-B. (2018). Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders. *Information Sciences, 460–461*, 83–102. https://doi.org/10.1016/j.ins.2018.04.092

Li, H., Zhou, S., Yuan, W., Li, J., & Leung, H. (2020). Adversarial-Example Attacks Toward Android Malware Detection System. *IEEE Systems Journal, 14*(1), 653–656. https://doi.org/10.1109/JSYST.2019.2906120

Li, S., Jiang, L., Zhang, Q., Wang, Z., Tian, Z., & Guizani, M. (2022). A Malicious Mining Code Detection Method Based on Multi-Features Fusion. *IEEE Transactions on Network Science and Engineering, 1–1*. https://doi.org/10.1109/TNSE.2022.3155187

Li, S., Li, Y., Wu, X., Otaibi, S. A., & Tian, Z. (2022). Imbalanced Malware Family Classification Using Multimodal Fusion and Weight Self-Learning. *IEEE Transactions on Intelligent Transportation Systems, 1–11*. https://doi.org/10.1109/TITS.2022.3208891

Li, X., & Li, Q. (2021). An IRL-based malware adversarial generation method to evade anti-malware engines. *Computers & Security, 104*, Article 102118. https://doi.org/10.1016/j.cose.2020.102118

Li, Y., Wang, Y., Wang, Y., Ke, L., & Tan, Y. (2020). A feature-vector generative adversarial network for evading PDF malware classifiers. *Information Sciences, 523*, 38–48. https://doi.org/10.1016/j.ins.2020.02.075

Ma, Y., Liu, S., Jiang, J., Chen, G., & Li, K. (2021). A Comprehensive Study on Learning-Based PE Malware Family Classification Methods. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1314–1325. 10.1145/3468264.3473925.

Millar, S., McLaughlin, N., Martinez del Rincon, J., & Miller, P. (2021). Multi-view deep learning for zero-day Android malware detection. *Journal of Information Security and Applications, 58*, Article 102718. https://doi.org/10.1016/j.jisa.2020.102718

Mercaldo, F., & Santone, A. (2020). Deep learning for image-based mobile malware detection. *Journal of Computer Virology and Hacking Techniques, 16*(2), 157–171. https://doi.org/10.1007/s11416-019-00346-7

Moti, Z., Hashemi, S., Karimipour, H., Dehghantanha, A., Jahromi, A. N., Abdi, L., & Alavi, F. (2021). Generative adversarial network to detect unseen Internet of Things malware. *Ad Hoc Networks, 122*, Article 102591. https://doi.org/10.1016/j.adhoc.2021.102591

Naeem, H., Ullah, F., Naeem, M. R., Khalid, S., Vasan, D., Jabbar, S., & Saeed, S. (2020). Malware detection in industrial internet of things based on hybrid image visualization and deep learning model. *Ad Hoc Networks, 105*, Article 102154. https://doi.org/10.1016/j.adhoc.2020.102154

Nagaraju, R., & Stamp, M. (2021). *Auxiliary-Classifier GAN for Malware Analysis* (arXiv:2107.01620). arXiv. http://arxiv.org/abs/2107.01620.

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 Networks, 124*, 243–257. https://doi.org/10.1016/j.neunet.2020.01.003

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 - VizSec '11*, pp. 1–7. 10.1145/2016904.2016908.

Nguyen, H., Di Troia, F., Ishigaki, G., & Stamp, M. (2022). *Generative Adversarial Networks and Image-Based Malware Classification* (arXiv:2207.00421). arXiv. http://arxiv.org/abs/2207.00421.

Odena, A., Olah, C., & Shlens, J. (2017). Conditional Image Synthesis with Auxiliary Classifier GANs. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 2642–2651. https://proceedings.mlr.press/v70/odena17a.html.

Pei, X., Yu, L., & Tian, S. (2020). AMalNet: A deep learning framework based on graph convolutional networks for malware detection. *Computers & Security, 93*, Article 101792. https://doi.org/10.1016/j.cose.2020.101792

Peng, X., Xian, H., Lu, Q., & Lu, X. (2021). Semantics aware adversarial malware examples generation for black-box attacks. *Applied Soft Computing, 109*, Article 107506. https://doi.org/10.1016/j.asoc.2021.107506

Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., & Nicholas, C. (2017). *Malware Detection by Eating a Whole EXE* (arXiv:1710.09435). arXiv. 10.48550/arXiv.1710.09435.

Rezaei, T., Manavi, F., & Hamzeh, A. (2021). A PE header-based method for malware detection using clustering and deep embedding techniques. *Journal of Information Security and Applications, 60*, Article 102876. https://doi.org/10.1016/j.jisa.2021.102876

Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., & Ahmadi, M. (2018). *Microsoft Malware Classification Challenge*. arXiv:1802.10135 [Cs]. http://arxiv.org/abs/1802.10135.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., & Chen, X. (2016). *Improved Techniques for Training GANs* (arXiv:1606.03498). arXiv. 10.48550/arXiv.1606.03498.

Sharma, O., Sharma, A., & Kalia, A. (2022). Windows and IoT malware visualization and classification with deep CNN and Xception CNN using Markov images. *Journal of Intelligent Information Systems*. https://doi.org/10.1007/s10844-022-00734-4

Stamp, M., Chandak, A., Wong, G., & Ye, A. (2021). *On Ensemble Learning*. arXiv:2103.12521 [Cs]. http://arxiv.org/abs/2103.12521.

Sudhakar, & Kumar, S. (2021). MCFT-CNN: Malware classification with fine-tune convolution neural networks using traditional and transfer learning in Internet of Things. *Future Generation Computer Systems, 125*, 334–351. https://doi.org/10.1016/j.future.2021.06.029

Taheri, R., Javidan, R., & Pooranian, Z. (2021). Adversarial android malware detection for mobile multimedia applications in IoT environments. *Multimedia Tools and Applications, 80*(11), 16713–16729. https://doi.org/10.1007/s11042-020-08804-x

Taheri, R., Javidan, R., Shojafar, M., Vinod, P., & Conti, M. (2020). Can machine learning model with static features be fooled: An adversarial machine learning approach. *Cluster Computing, 23*(4), 3233–3253. https://doi.org/10.1007/s10586-020-03083-5

tisf. (2022). *theZoo—A Live Malware Repository* [Web Repository]. https://github.com/ytisf/theZoo.

Tuncer, T., Ertam, F., & Dogan, S. (2021). Automated malware identification method using image descriptors and singular value decomposition. *Multimedia Tools and Applications, 80*(7), 10881–10900. https://doi.org/10.1007/s11042-020-10317-6

Vasan, D., Alazab, M., Wassan, S., Naeem, H., Safaei, B., & Zheng, Q. (2020). IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture. *Computer Networks, 171*, Article 107138. https://doi.org/10.1016/j.comnet.2020.107138

Vasan, D., Alazab, M., Wassan, S., Safaei, B., & Zheng, Q. (2020). Image-Based malware classification using ensemble of CNN architectures (IMCEC). *Computers & Security, 92*, Article 101748. https://doi.org/10.1016/j.cose.2020.101748

Verma, V., Muttoo, S. K., & Singh, V. B. (2020). Multiclass malware classification via first- and second-order texture statistics. *Computers & Security, 97*, Article 101895. https://doi.org/10.1016/j.cose.2020.101895

VirusShare.com. (2022). [Web Repository] https://virusshare.com/ (accessed May 14, 2022).

VirusTotal Statistics. (2022). [Web Page] https://www.virustotal.com/gui/stats (accessed May 14, 2022).

Vx-underground. (2022). [Web Repository] https://www.vx-underground.org/archive/VxHeaven/index.html (accessed May 14, 2022).

Wang, G.-G., Lu, M., Dong, Y.-Q., & Zhao, X.-J. (2016). Self-adaptive extreme learning machine. *Neural Computing and Applications, 27*(2), 291–303. https://doi.org/10.1007/s00521-015-1874-3

Wang, Y., Qiao, X., & Wang, G.-G. (2022). Architecture evolution of convolutional neural network using monarch butterfly optimization. *Journal of Ambient Intelligence and Humanized Computing*. https://doi.org/10.1007/s12652-022-03766-4

Xiao, G., Li, J., Chen, Y., & Li, K. (2020). MalFCS: An effective malware classification framework with automated feature extraction based on deep convolutional neural networks. *Journal of Parallel and Distributed Computing, 141*, 49–58. https://doi.org/10.1016/j.jpdc.2020.03.012

Yuan, B., Wang, J., Liu, D., Guo, W., Wu, P., & Bao, X. (2020). Byte-level malware classification based on markov images and deep learning. *Computers & Security, 92*, Article 101740. https://doi.org/10.1016/j.cose.2020.101740

Zhang, J., Gao, C., Gong, L., Gu, Z., Man, D., Yang, W., & Li, W. (2021). Malware Detection Based on Multi-level and Dynamic Multi-feature Using Ensemble Learning at Hypervisor. *Mobile Networks and Applications, 26*(4), 1668–1685. https://doi.org/10.1007/s11036-019-01503-4

Zhang, Y., Li, H., Zheng, Y., Yao, S., & Jiang, J. (2021). Enhanced DNNs for malware classification with GAN-based adversarial training. *Journal of Computer Virology and Hacking Techniques, 17*(2), 153–163. https://doi.org/10.1007/s11416-021-00378-y