Journal of King Saud University – Computer and Information Sciences 34 (2022) 1968–1983

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

Mal-Detect: An intelligent visualization approach for malware detection

Olorunjube James Falana [a,*], Adesina Simon Sodiya [a], Saidat Adebukola Onashoga [a], Biodun Surajudeen Badmus [b]

[a] Department of Computer Science, Federal University of Agriculture Abeokuta, Ogun State, Nigeria
[b] Department of Physics, Federal University of Agriculture, Abeokuta, Ogun State, Nigeria

ARTICLE INFO

Article history:
Received 29 November 2021
Revised 24 February 2022
Accepted 26 February 2022
Available online 12 March 2022

Keywords:
Malware
Convolutional neural network
Generative adversarial network
Attack
Visualization

ABSTRACT

Recent outbreaks of pandemics have deepened the adoption and use of IT-based systems. This development has led to an exponential increase in cyberattacks caused by malware. Current approaches (static, dynamic and hybrid) for detecting malware still exhibit low efficiency when subjected to sophisticated malware. This work used an ensemble technique consisting of Deep Convolutional Neural Network and Deep Generative Adversarial Neural Network (Mal-Detect) to analyse, detect, and categorise malware. The proposed Mal-Detect first converts both malware and benign file binaries into RGB binary images. New malware images are then generated using a deep generative adversarial neural network from original malware samples. The generated malware images with original malware and benign files images are pre-processed and trained with Deep Convolutional Neural Networks to extract important features from the dataset. The effectiveness of Mal-Detect was evaluated against three benchmark datasets; MaleVis, Mallmg and Virushare. The results of the evaluation showed that Mal-Detect outperforms other state of art techniques with an accuracy of 99.8% and an average accuracy of 96.77% on all malware datasets tested. These results showed that Mal-Detect can be deployed for detecting all categories of malware.

1. Introduction

Malware is a computer program that is written to harm a computer system without the owner's knowledge (Potter & Day, 2009; Santos et al., 2013; Vasan et al., 2020a, Vasan et al., 2020b). Numerous types of malware harm computer systems, including computer viruses, trojan horses, worms, spyware, adware, rootkits, botnets, and others (Santos et al., 2013; Sodiya et al., 2014; Symantec, 2021). According to Symantec's 2021 security threat report, the biggest threat in the year 2020 was not COVID but rather malware. More than 431 million new malware variants, ransomware and zero attacks were noticed by Symantec in the year 2020 and the figure is predicted to be double at the end of the year 2021.

Malware has become more prevalent given the complexity of today's network and the attack landscape. Computer users and business owners are finding it very difficult to keep pace with cybercriminals. According to a survey by Flirch (2021), out of 582 information security professionals interviewed, 50% of the respondents believed their organization is ill-prepared to tackle ransomware as a form of malware attack. 75% of organizations attacked by ransomware in 2020 were running up to date endpoint protection (Firch, 2021).

The technique for identifying malware is based on extracting the malware's features vector using static and dynamic analysis approaches (Ni et al., 2018). Static analysis examines a malicious program in a controlled environment without executing it,

* Corresponding author.
E-mail addresses: falanaoj@funaab.edu.ng (O.J. Falana), Sodiyaas@funaab.edu.ng (A.S. Sodiya), Onashogasa@funaab.edu.ng (S.A. Onashoga), badmusbs@funaab.edu.ng (B.S. Badmus).

Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.jksuci.2022.02.026
1319-1578/

whereas dynamic analysis monitors it under inspection. The process for analysing a given malware usually involves unpacking, disassembling and extracting important features (Alam et al., 2015; Ding et al., 2018; Kang et al., 2016). The static analysis, which is one of the most widely used malware detection techniques is faced with low detection rates, packing, code obfuscation, and is easily evaded by code modifications. In contrast, dynamic analysis has a higher detection rate, however, most dynamic analysis tools, such as Dynamic TaintAPI Call monitoring (Ding et al., 2018), system call monitoring (Win et al. (2015), Control Flow Graph (ACFG) (Alam et al., 2015) can only observe a partial executable's behaviour. Also, if malware chooses to mask its behaviour when running in a monitored environment, the dynamic analysis may be rendered ineffective.

Machine learning approaches have recently been used to distinguish between malicious and benign applications using a combination of dynamic and static analysis. Studies have shown that machine learning techniques can be rendered ineffective by the use of adversarial data (Grosse et al., 2017; Liu et al., 2020; Yuan et al., 2019). By applying perturbation or noise to the original malware dataset, adversarial samples are created. Malware detection has also benefited from the use of deep learning and reinforcement learning. However, most deep learning approaches necessitate huge datasets and a significant training cost.

A visualization-based detection technique capable of analysing malware behaviour is proposed to overcome the challenges posed by other detection techniques. Visualizing malware as a coloured image offers the benefit of differentiating different components of the malware binary as malware programmers only tamper with a small section of the malware codes to generate a new mutant. Malware executables and benign applications are converted into RGB malware images. The RGB image pixels are encoded as matrices, which can be thought of as app activity signatures. The detection is based on fine-tuned deep convolutional neural networks. Novel malware samples are generated and added to the dataset using the DGAN to give a robust detection that can withstand adversarial attacks.

The contributions of this work include.

- A visualization model for understanding and categorising malware behaviours
- A model for generating a novel malware from an original malware.
- A computed threshold for determining the level of the variant in the generated malware set
- A visualization model for learning using a small size of the dataset
- Implementation of binary and multiclass classification on a wide variety of datasets

The rest of this paper is structured as follows: Section 2 provides a review of related work Section 3 presents the methodology of the proposed techniques. In Section 4, experimental results and discussion are presented and Section 5 concludes the research study.

1.1. Literature review

Recently, a few non-visual ways for detecting malware have been proposed, each of which uses explicit features retrieved from executable files to identify malware. These features, which are often the underlying components of executable documents, have demonstrated that they can accurately distinguish malware.

Ding et al. (2018) used a dynamic taint analysis technique to capture the dependencies between system calls by looking at how tainted data was propagated between system calls. Using contaminated output arguments as API call return values and Windows API calls as taint sources, the approach traced taint tags of executables to determine whether any input argument of an API call is contaminated. A dependency graph between API calls was built by analysing the taint information in the recording file. The proposed method had the limitation of being able to observe only a portion of an executable's behaviour, and it could be made obsolete if malware choose to mask its destructive behaviour when executing in a monitored environment.

In Alam et al. (2015), an Annotated Control Flow Graph (ACFG) of a sliding window of difference and Control Flow Weight (SWOD-CFWeight) was employed to detect metamorphic malware in real-time. The ACFG was used to capture a program's semantic flow as well as to offer faster control flow graph (CFG) matching, while SOD-CFWeight was used to capture changes in opcodes across various compilers. The authors employed this strategy to counteract the obfuscation effects of both metamorphic and polymorphic malware. The techniques were evaluated and found to have a detection rate ranging from 94% to 99.6%. However, most applications are larger than the amount of dataset tested, the proposed technique cannot be employed successfully for programs larger than 10 MB. In addition, the technique ran into an NP-complete difficulty, which is a major issue in call graph matching.

Almarri and Sant (2014) used an API call graph creation approach to translate malware API into a call graph utilizing an integrated operating system APIs. The approach helps to avoid computation complexity owing to the NP-Complete problem. The technique was evaluated with benign samples against Trojans, Viruses, and Worms, with a 98% accuracy and 0% false-positive rate on the three samples. During the query, the authors, on the other hand, assumed that the best subgraph corresponds to the same malware family. This is not always the case, because malware developers can utilize a powerful tool to construct a sophisticated obfuscation strategy, resulting in a huge gap between the malware input sample and the malware family. Also, most dynamic analysis tools are unable to fully explore all execution routes in malware samples, and as a result, certain critical API calls may be missed. Furthermore, because an attacker can use polymorphic techniques to load malware samples with worthless APIs, the extracted API call list and parameters may be wrong.

It is increasingly difficult for traditional machine learning methods to learn high-level features from data (Xin et al., 2020). In recent times, numerous visualization techniques for malware analysis have also been proposed. To efficiently detect different types of malware and their variants, Liu et al. (2020) presented a data visualization and adversarial training based on Machine Learning. Naeem et al. (2020) devised a system for detecting malware attacks on the Industrial Internet of Things (IIoT) (MD-IIOT). The authors created a sniffer gateway to monitor and collect data on incoming and outgoing traffic. MD-IIOT integrated a deep convolutional neural network in the detection module and had a detection accuracy of 97.81% on the IIOT dataset and 98.47% on the Windows dataset, respectively. However, the method cannot counteract an adversarial form of attack. Using image processing techniques, Nataraj et al. (2011) proposed a method for visualizing and classifying malware.

In another related work, Vasan et al. (2020a), Vasan et al. (2020b) extracted malware binaries into an 8-bit unsigned integer vector. The vector was grouped into a 2D array, which was then shown using a colour map. The model was tested with a total of 26,654 samples drawn from the Mallmg malware dataset, the IoT android mobile dataset, and benign samples. The technique has an accuracy of 98.82% on the Mallmg malware dataset and 97.37% on the IoT android mobile dataset, according to the results. Kumar (2020) highlighted the problems that have arisen as a result of the rise of fileless malware. To avoid detection by antivirus soft-

ware, fileless malware attacks do not download dangerous files or write any material to the disk, causing major concern for enterprises.

2. Methodology

2.1. Visualizing of malware

Visualization is the process of graphically expressing data and engaging with these representations to acquire insight into the data. As illustrated in Fig. 1, this paper proposes a simple yet effective method for viewing and identifying malware using image processing techniques and a deep learning methodology called Deep Convolutional Neural Network (DCNN).

The key advantage of displaying malware as an image is the ease with which different components of a binary may be distinguished. Furthermore, because malware authors simply modify a small portion of the code to create new variations, images are excellent for detecting small changes while maintaining the overall structure. As a result, malware variants belonging to the same family may be distinguished.

The process of visualizing a malware using Deep Convolutional Neural Network is divided into three phases:.

i. Malware and Benign app Transformation
ii. Generating Adversarial samples
iii. Malware Classification

2.1.1. Malware Transformation

Every image is made up of a series of dots (pixels) that are organized in a specific order. If the colour order of a pixel is modified, the image will also change.

Let $D_1, D_2, \ldots D_n$ be a set of benign binary defined in R^n space and $M_1, M_2, \ldots M_n$ also be a set of malware binary defined in R^n space which can be represented as.

$$D : \Omega \rightarrow R^n \tag{1}$$

This implies D which maps Ω to R^n is a real function defined by $D_1, D_2, D_3, \ldots, D_n \in R^n$ where$\Omega = \{1, 2, 3, \ldots, n\}$.

Clearly, D_Ω is an index set which is a collection of a set of benign binaries in R^n.

that is.

$$D_\Omega = \{D_1, D_2, D_3, \ldots, D_n\} \tag{2}$$

Furthermore,

Let D_Ω be a finite sequence of terms of eight benign bits with 1,2,3 …, n representing the number of terms of each subset D_1, D_2, D_3 …. D_n.

Hence, we have the Cartesian product map.

$$X : D_\Omega \times G \rightarrow R^n \tag{3}$$

as

$$D_\Omega \times G \rightarrow \{(D_1, g_1), (D_2, g_2), \ldots, (D_n, g_n)\} \tag{4}$$

Clarification:

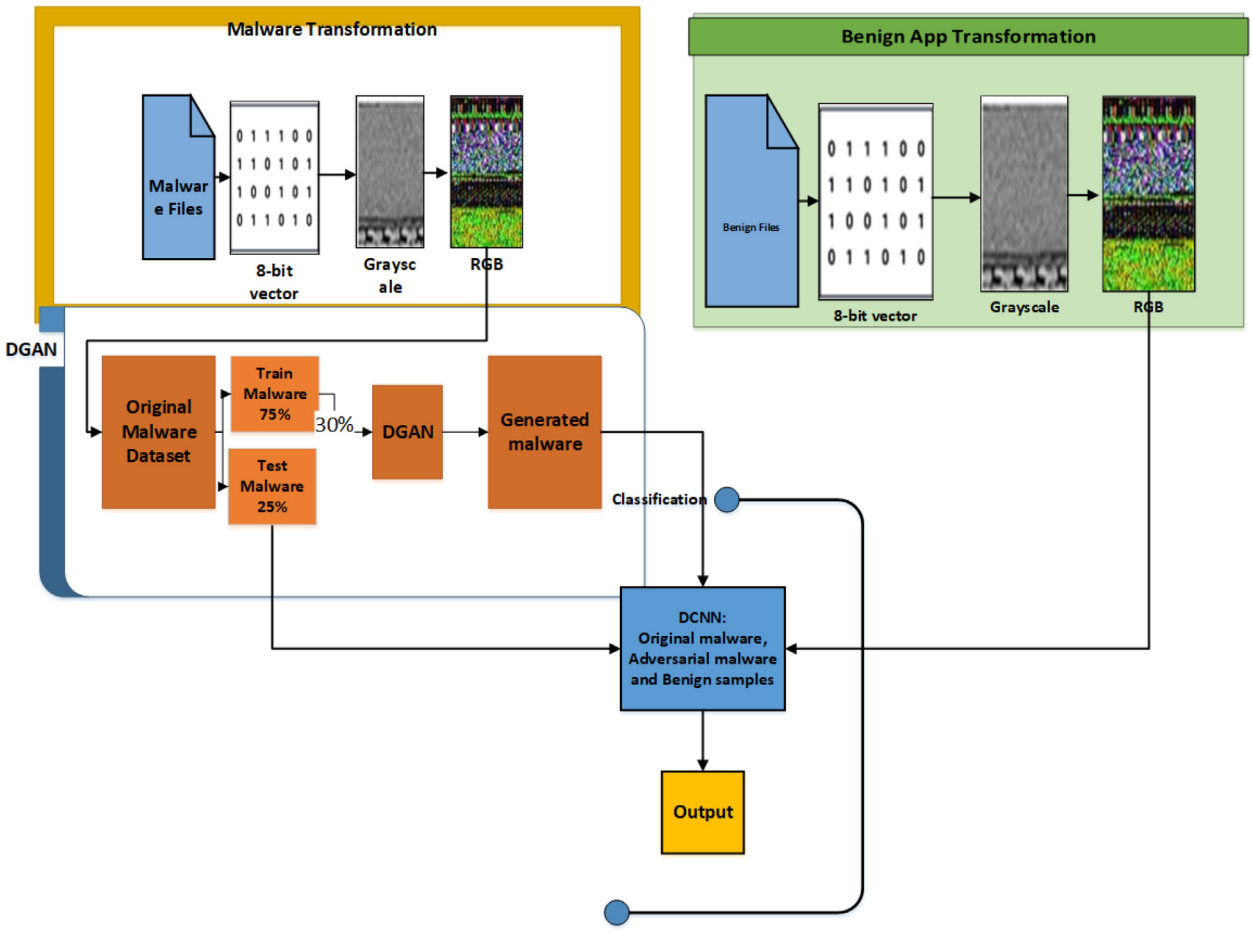

Fig. 1. General Architecture of Mal-Detect.

$$D_{1k} = \bigcup_{k=1}^{8} D_{1k} i.e. D_{11} \cup D_{12} \cup D_{13} \cdots \cup D_{18}$$

$$D_{2k} = \bigcup_{k=1}^{8} D_{2k} i.e. D_{21} \cup D_{22} \cup D_{23} \cdots \cup D_{28}$$

$$D_{1k} = \bigcup_{k=1}^{8} D_{1k} i.e. D_{31} \cup D_{32} \cup D_{33} \cdots \cup D_{38}$$

$$D_{nk} = \bigcup_{k=1}^{8} D_{nk} i.e. D_{n1} \cup D_{n2} \cup D_{n3} \cdots \cup D_{n8}$$

$$D_{\Omega} = \{D_{1k},\ D_{2k},\ D_{3k}, \ldots, D_{nk}\} \quad (5)$$

where k = {1, 2, 3 …, 8}

Equations (1) and (4) can be represented diagrammatically in Fig. 1.

Similarly, M_1, M_2 … M_n will have the same relationship as shown in equations (1) to (5) since each set is a subset in R^n.

Each malicious or benign binary file is organized into a two-dimensional array as a one-byte vector of unsigned integers, as indicated in Equation (5). This array is represented as a grayscale image in the range [0, 255] as illustrated in Algorithm 1, the result of the grayscale image is further converted to a binary image as shown in Algorithm 2.

Algorithm 1: Conversion of Malware Binary to Grayscale

```
i. Input: D = [D1, D2, … DN]
ii. Output: Gray Scale Image: g = [g1, g2, g3 … gn]
// Reading 8-bit unsigned integer of malicious code
iii. EightBit_array [0] = 0;
iv. Pixel_array[0][0] = 0
v. Image_width = 2^8
vi. Image_height = size(Di/2^11)
vii. for i in range(1, len(D)) do
viii.   for c in D do
ix.       if c = 0 or c = 1
x.          EightBit_array [i] = c
xi.       if i % 8 == 0
xii.        continue
xiii.     endif
xiv.      endif
xv. endfor
    // Setting the width and converting it into a vector
xvi.  for j = 1 to 256
xvii.     X[j] = EightBit_array[j]
xviii.    for k = 1 to EightBit_array.length()
        // converting to pixel array or grayscale
xix.        Pixel_array[j][k] = EightBit_array [k]
xx.       endfor
xxi.  endfor
xxii. endfor
```

The grayscale obtained from Algorithm 1 is further converted into a binary image using Otsu's binarization algorithm (Otsu, 1979) as shown in Algorithm 2.

Algorithm 2: Conversion of Grayscale Image to Binary Image using Otsu's binarization

```
i. Input: Grayscale image: img
ii. Output: Binary image
iii. Read image
 #check grayscale
   if(len(image.shape) == 1) do
     img = cv.imread('img', 0)
   else do
("Image must be gray")
iv. Calculate each intensity level's histogram and probabilities.
# Get the histogram of the image
bins_numumber = 256
histogram, binEdges = np.histogram(img,
bins = bins_number)
v. Calculate a threshold value, T
// global thresholding, make a list of all conceivable threshold values that we'll iterate over
bin_midpoint = (binEdges[: −1] + binEdges[1:]) / 2
weight_1 = np.cumsum(histogram)
weight_2 = np.cumsum(histogram[::-1])[::-1]# Otsu's thresholding after Gaussian
# iterate over the threshold until the one with the least in the class is found.
Mean_1 = np.cumsum(histogram * bin_midpoint) / weight_1
Mean_2 = (np.cumsum((histogram * bin_midpoint)[::-1]) / weight_2[::-1])[::-1]
Variance_in_class = weight_1[:-1] * weight_2[1:] * (mean1_[:-1] – mean_2 [1:])
vi. Generate maximum threshold of class variance
Max_val = np.argmax(Variance_in_class)
Max_threshold = bin_midpoint [:-1][ Max_val]
vii. Make a new Image Array of the same amount of rows and columns as the old image.
[x, y, z] = size(img) # for grayscale z will be 1
Binary_image = np.zeros(x, y);
viii. Allocate 1 to new binary image, if grayscale pixel is greater than or equal the threshold value; otherwise assign 0 to the new binary image
for i in range(1, x) do
  for j in range (1, y) do
      if img(i, j) >= Max_threshold
    Binary_image (i, j) = 1;
  else
  Binary_image (i, j) = 0;
 endfor
endfor
```

2.1.2. Generating adversarial samples

To develop a robust detection system against adversarial attacks, an adversarial perturbation is introduced on malicious samples through Deep Generative Adversarial Network (DGAN). DGAN is a generative model for describing the conflict between two adversaries known as the generator and the discriminator. The generator introduces perturbations into observations from samples of the original dataset while the discriminator attempts to predict whether the observations are from the original dataset or one of the generator's forgeries as shown in Fig. 3.

a. Generator

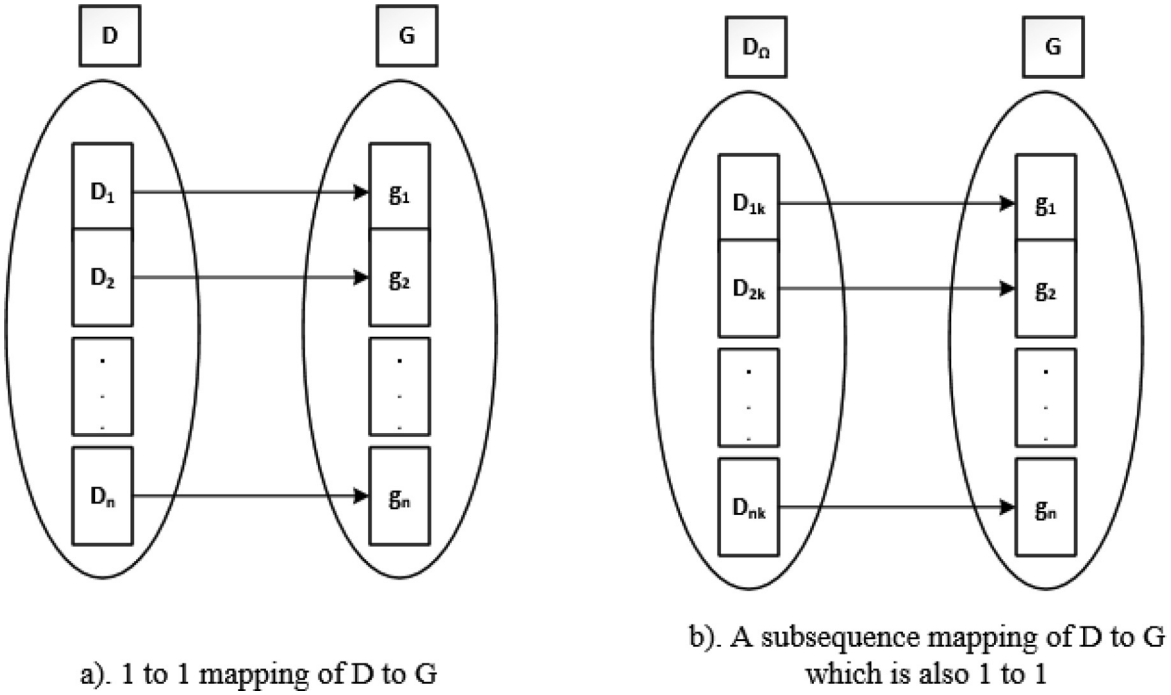

Fig. 2. An illustrative example for 1 to 1 mapping from the given: (a) benign binary D to grayscale image G, (b) the introduction of an index set DΩ also produces a new 1 to 1 mapping.

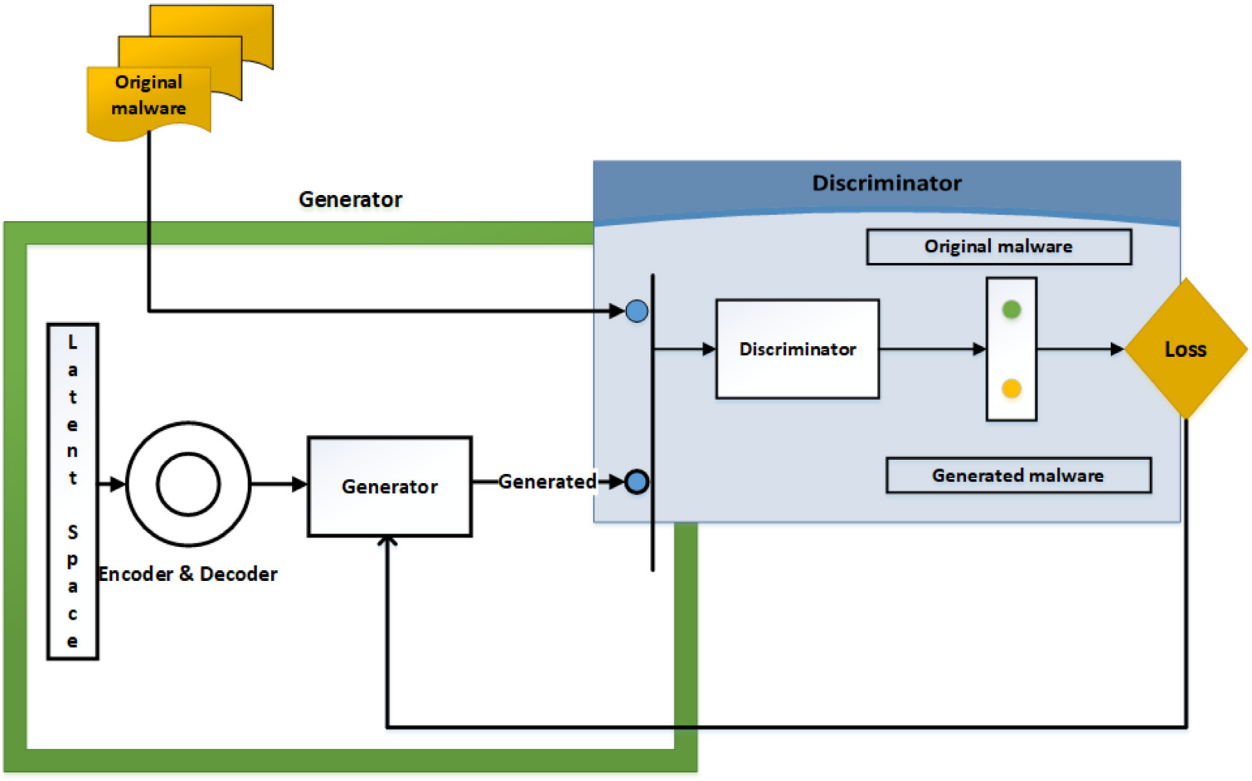

Fig. 3. Deep Generative Adversarial Network Framework.

The generative portion is in charge of creating phony malware and benign from N-dimensional uniform random variables (noise). The probability P(X) is captured by the generator where X is the input.

b. Discriminator

The discriminator model predicts a binary class label for the actual or created image based on a domain of generated images.

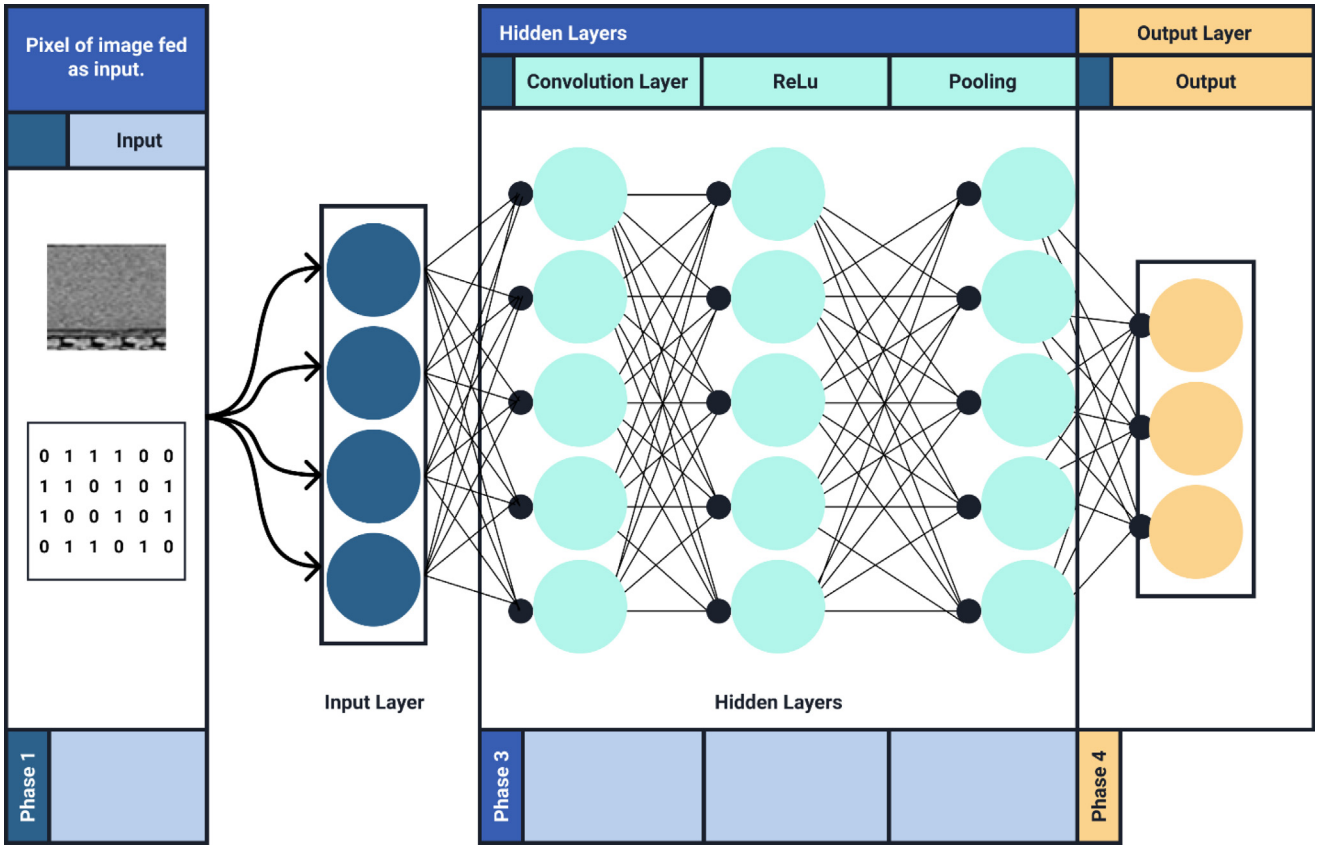

Fig. 4. Visualizing malware using Deep Convolutional Neural Network.

The discriminator records the conditional probability P(Y|X), where X is the input and Y is the label, in other words.

The two models are trained in an adversarial zero-sum game until the discriminator model is tricked roughly half of the time, indicating that the generator model is producing believable examples.

c. Training Deep Convolutional Generative Adversarial Networks

The GAN model consists of two deep neural network models that improve themselves by simultaneously training on both the original and generated malware image dataset and benign image dataset. The first phase involves pre-training the discriminator *D*, a CNN-based image classifier with real malware images x_i and a benign software image t_i while the second phase involves testing the discriminator *D* with generated malware images $G(z)_i$ produced by the generator. The generator *G* aims to trick the discriminator into identifying generated malware samples as real, while the job of the discriminator *D* is to distinguish generated malware from real and the generated benign image from real.

The goal of the GAN model is to minimize the loss function of the generator and maximize that of the discriminator as given by Equation (6). This allows for the creation of photorealistic malware images and benign software images such that generated malware images and generated benign software images are eventually classified by the discriminator as real.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\ p_x(x)}[\log D(x)] + \mathbb{E}_{h\ p_h(z)}[\log(1 - D(G(z)))] \quad (6)$$

where:

D: Discriminator,

G: Generator.

P_x: Probability distribution of real data.

x: Input data.

p_h: Probability distribution of model data.

G_z: generated malware image,

G_{zb}: generated benign software image,

z: Noise vector h = generated data.

p_G: Distribution of generated malware images.

D_x: of the 1st term is 1; i.e., discriminator's evaluation of real data.

$D_{G(z)}$ of the 2nd term is 0; i.e., discriminator's evaluation of generated data.

$V_{(D,G)}$ converges when $P_G \approx P_x$.

Algorithm 3. GAN Dataset generation.

Input: Original Malware Images:$X_1X_2, \ldots, X_n$; Original Benign Images:$T_1T_2, \ldots, T_n$; GAN Generator: G_G, GAN Discriminator: G_D

Output: Generated Images:$G(z)_1, G(z)_2, \ldots, G(z)_n$

```
Def Dataset_generation(input_data: malware_image,
   benign_image):
   while true:
      train G_D with input_data_i
   G(z) ← G_G(z); where z = N[0,1]
   G(z)_i ≈ input_data_i
   y ← G_D (G(z))
   if y is real:
      break;
   else if y is generated:
      train G_G
```

(*continued on next page*)

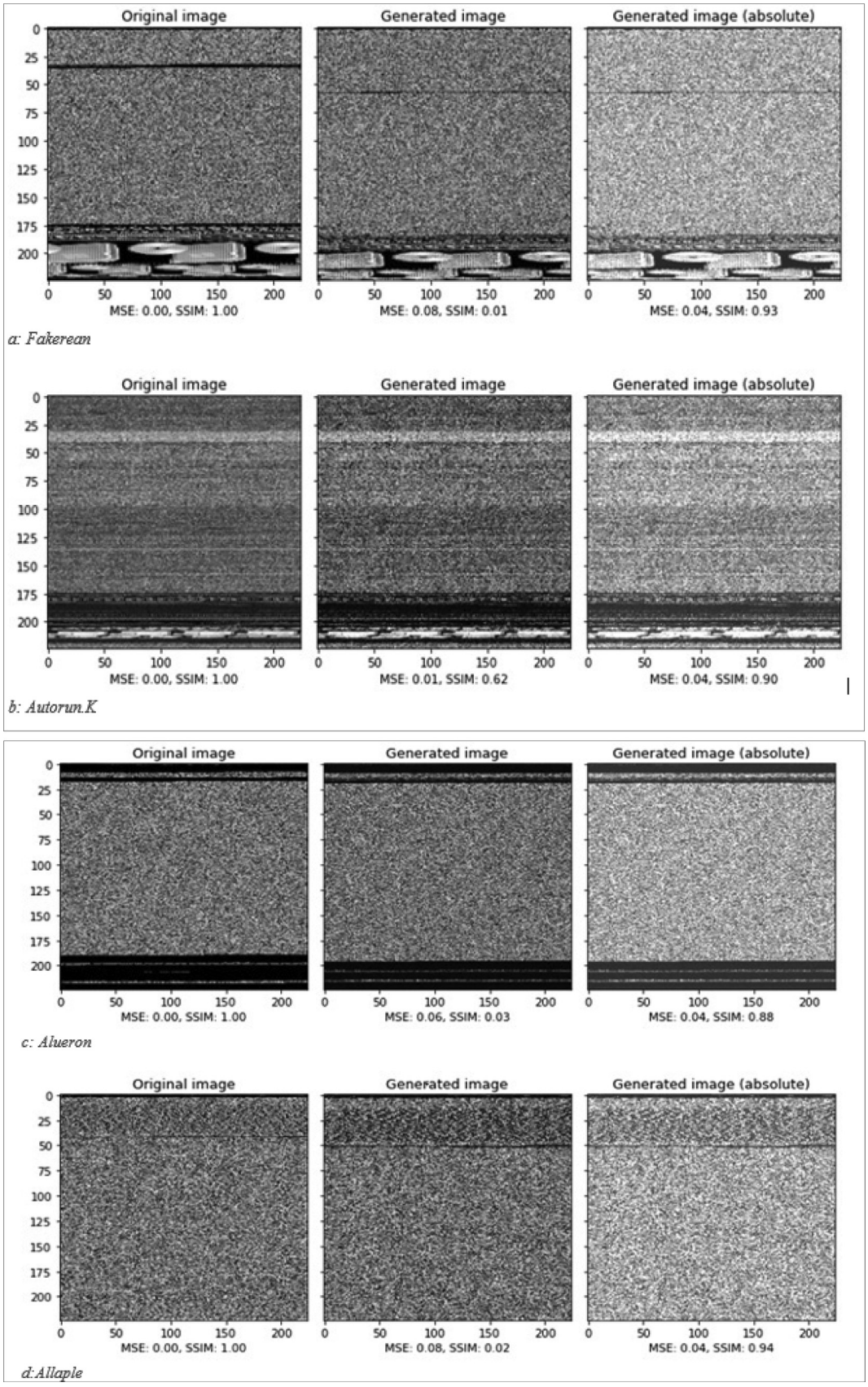

Fig. 5. Generated malware Images.

(*continued*)

Input: Original Malware Images:$X_1X_2,\ldots,X_n$; Original Benign Images:$T_1T_2,\ldots,T_n$; GAN Generator: G_G, GAN Discriminator: G_D
Output: Generated Images:$G(z)_1, G(z)_2,\ldots, G(z)_n$

```
  end if
  return y
  end while
def load_image(input_data_i, outfilename):
  img = Dataset_generation(input_data_i)
  img = Image.fromarray(np.asarray(np.clip(img,0,255),
  dtype="uint8"), "L")
  img.save(outfilename)
```

Algorithm 4. GAN Training.

Input: Populated malware data T = [(x$_1$ y$_1$),. . .,(x$_n$,y$_n$)], the encoder of Generator of GAN:G_G, Discriminator of GAN: G_D

Output: Malware detector D

```
# populating both malware and benign data
  def training(T = (x, y), G_G, G_D):
    #sample x_1, ... x_n; label: y_1,. .., y_n; generator of GAN: G_G;
  discriminator of GAN: G_D
  for i in range(1, m):
    for j in range(1, n):
      for k in range (1, o):
        Perform convolutional operations to generate
  intermediate features
        Perform activation function (ReLU)
        Perform pooling operations (max-pooling)
        Skip connections
        Perform activation function (ReLU)
        end for
      Compute softmax activation of the final layer to output o
    end for
    Compute cross-entropy error E
    Back propagate E to compute the gradient w.r.t output o
    Update model parameters θ using SDG
    end for
    return malware_detector
```

2.1.3. Malware classification

DCCN-DGAN employs a deep convolutional neural network for the detection and categorization of malware. The core of DCNN is shown in Fig. 4.

a. Convolutional layer

Convolutional layers serve as detection filters for specific features in the data. Lower-level features are detected by the first layers using the kernel size, whereas the second layer detects higher-level features. The higher the layer, the more high-level features are detected (Wu, 2017).

The DCCN accepts an image x^i with the dimensions H heights, W widths, and D channels (R, G, B). Assume that the $l-th$ input is an order 3 tensor of size$H^lW^lD^l$. A convolution kernel will also be of a sizeHWD^l. The products of corresponding elements in all the D^l channels are computed when the kernel is overlapped on top of the input tensor at the spatial location (0; 0; 0), and the sum ofHWD^l. gives the convolutional result in the spatial location.

Let f represents all of the kernels in$R^{H\times W\times D^lD}$. Therefore f is an order 4 tensor. The index variables $0 \le l < H; 0 \le j < W; 0 \le d^l < D^l and 0 \le d < D$ are used to locate a specific element in the kernels.

A function will convert the input x^i to output y, which will also be the input to the next layer. As a result, y and x^{i+1} are the same objects.

This can be expressed in Equation (7).

$$y^{l+1}_{i,j^{l+1},d} = \sum_{i=0}^{H}\sum_{j=0}^{W}\sum_{d^l=0}^{D^l} f_{i,j,d^l,d} \times x^l_{i^{l+1}+i,j^{l+1}+j,d^l} \tag{7}$$

In this equation, $x^l_{i^{l+1}+i,j^{l+1}+j,d^l}$ is the element of x^l indexed by the tuple.$\left(i^{i+1}+1, j^{i+1}+1, d^i\right)$

$\forall 0 \le d \le D = D^{i+1}$ and for any spatial location $\left(i^{l+1}, j^{l+1}\right)$ satisfying$0 \le i^{l+1} < H^l - H + 1 = H^{l+1}, 0 \le j^{l+1} < W^l - W + 1 = W^{l+1}$.

b. **Re**LU layer

Rectified Linear Unit (ReLU) helps to improve the nonlinearity of DCNN. It does not change the size of input image x^i and target image y^1. Supposing $x^l_{i,j,d}$ is one of the H^l W^l D^l features extracted

Table 1
Summary of experimental results on binary classification.

Models Baselines			Accuracy	Precision	Recall	F1-score
Deep learning Method	**Mal-Detect**	Benign	0.965	0.96	0.97	0.96
		Malware		0.97	0.96	0.97
	Resnet50	Benign	0.934	0.96	0.90	0.93
		Malware		0.92	0.96	0.94
Ensemble	Extra Trees Classifier	Benign	0.953	0.93	0.96	0.94
		Malware		0.96	0.93	0.85
	Gradient Boosting Classifier	Benign	0.956	0.93	0.96	0.94
		Malware		0.96	0.93	0.95
	Bagging	Benign	0.937	0.88	0.98	0.93
		Malware		0.98	0.89	0.93
Machine Learning	SVM	Benign	0.945	0.94	0.94	0.94
		Malware		0.95	0.94	0.94
	KNN	Benign	0.861	0.97	0.73	0.83
		Malware		0.80	0.98	0.88
	Naïve Bayes	Benign	0.926	0.93	0.88	0.91
		Malware		0.90	0.94	0.92
	Random Forest	Benign	0.941	0.89	0.99	0.94
		Malware		0.99	0.89	0.94
	Decision Tree Classifier	Benign	0.902	0.90	0.88	0.89
		Malware		0.90	0.91	0.90

by DCNN layers, which can be positive or negative. Then Relu will set $x^l_{i,j,d}$ to negative or zero if that region does not contain these patterns as illustrated in Equation (8).

$$y_{i,j,d} = \max\{0, x^l_{i,j,d}\} \tag{8}$$

with $0 \leq i < H^l = H^{i+1}$, $0 \leq j < W^l = W^{l+1}$, and $0 \leq d < D^l = D^{l+1}$ there is no parameter inside a ReLu layer, hence no need for learning in this layer.

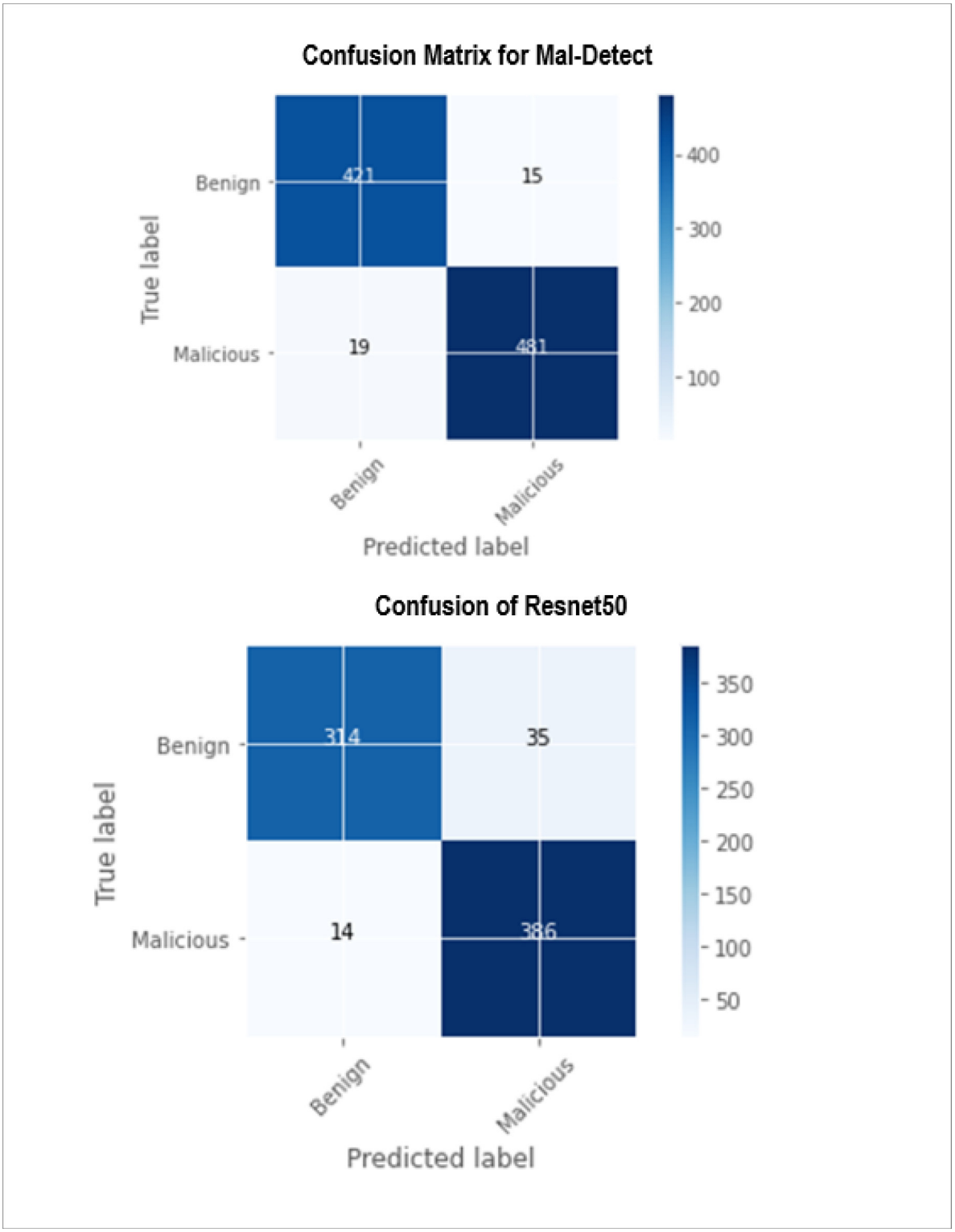

Fig. 6. Confusion Matrix comparison of Mal-Detect and Resnet50.

Based on Equation (8) it can be deduced that $\frac{d_{y_i,j,d}}{x^l_{i,j,d}} = [\![x^l_{i,j,d} > 0]\!]$ where $[\![.]\!]$ is the indicator function, being 1 if its argument is true, and 0 otherwise.

Hence, we have.

$$\left[\frac{\delta z}{\delta x^l}\right] = \begin{cases} \left[\frac{\delta}{\delta x^l}\right] i,j,difx^l_{i,j,d} > 0 \\ 0 \end{cases} \quad (9)$$

where y is an alias for x^{l+1}. The function max (0; x) is not differentiable at × = 0, hence.

c. Pooling layer

Pooling is done exclusively to downsize the image's size. Whenever the images are too big, it's sometimes necessary to limit the

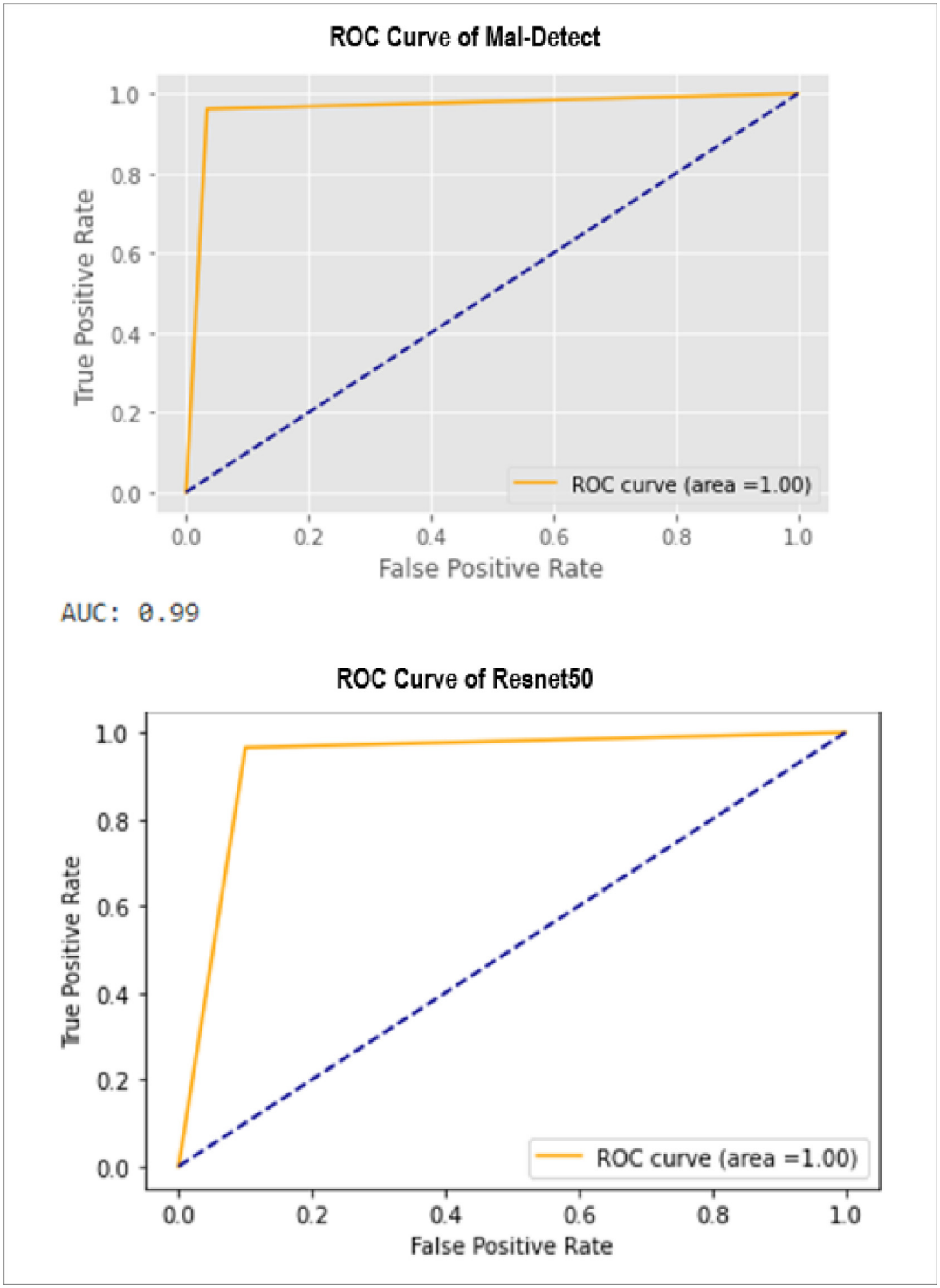

Fig. 7. ROC comparison for Mal-Detect and Resnet50.

number of trainable parameters using the technique of max pooling. Pooling is done separately for each convolution layer.

d. Fully connected layer

Every neuron in one layer is connected to every neuron in another layer in fully connected layers. To classify the image, the fully connected layer receives the flattened matrix from the pooling layer as input. Flattening is the way of transforming all of the max-pooling into a single long continuous linear vector.

e. Output layer

After numerous convolution layers and padding, the output would be a class vector. A completely connected layer is used to provide a final output that is equal to the number of classes necessary. The output layer uses a loss function similar to categorical cross-entropy to compute prediction error. After the forward pass is over, backpropagation starts updating the weights and biases for error and loss reduction.

3. Implementation and evaluation

3.1. Description of the dataset

A publicly available malware dataset from the Virushare database was used for the evaluation of this study. The dataset comprised of 2,000 malware samples from different malware families added to the virushare database in the year 2020. The benign programs, which comprised of 1744 programs were collected from different executable programs, namely, game, browser, word processor, business, etc. After the collection, the clean files were scanned on VirusToal using 30 different anti-malware engines to ensure that they are clean files.

In addition to this, Mal-Detect was benchmarked with a dataset comprised of 9339 Malimg malware images belonging to 25 malware families (Nataraj et al., 2011) and the MaleVis dataset comprised of 8,750 malware of 25 malware families (Hemalatha et al., 2021).

The experiment was performed using Python 3.7.12 with Keras-2.6.0 and TensorFlow 2.6.0 in a Google collab environment utilizing its GPU, RAM and Hard disk.

3.2. Evaluation

To evaluate the performance of Mal-Detect, some sets of evaluation criteria were used namely Accuracy, Precision, Recall, F1-score as shown in Equations 10–13. Also, Confusion Matrix, Area Under Curve (AUC) and Receiver Operating Characteristic Curve (ROC) were further used for the performance analysis.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{10}$$

$$Precision = \frac{TP}{TP + FP} \tag{11}$$

$$Recall = \frac{TP}{TP + FN} \tag{12}$$

$$F_{Measure} = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{13}$$

where:

True Positive (TP): indicates malware files correctly classified.

False Positive (FP): represents the benign files misclassified as malware.

False Negative (FN): shows the number of malware misclassified as benign files.

True Negative (TN): indicates the number of benign file cases correctly classified.

3.3. Experimental results

To show the effectiveness of Mal-Detect, the experiment was designed to:.

a) Generate a novel malware sample: Fig. 5 shows the visual representation of malware generated from the original malware image using DGAN. From the visual representation, it was seen that malware images from the same malware families were visually similar compared with images from other malware families. Met-

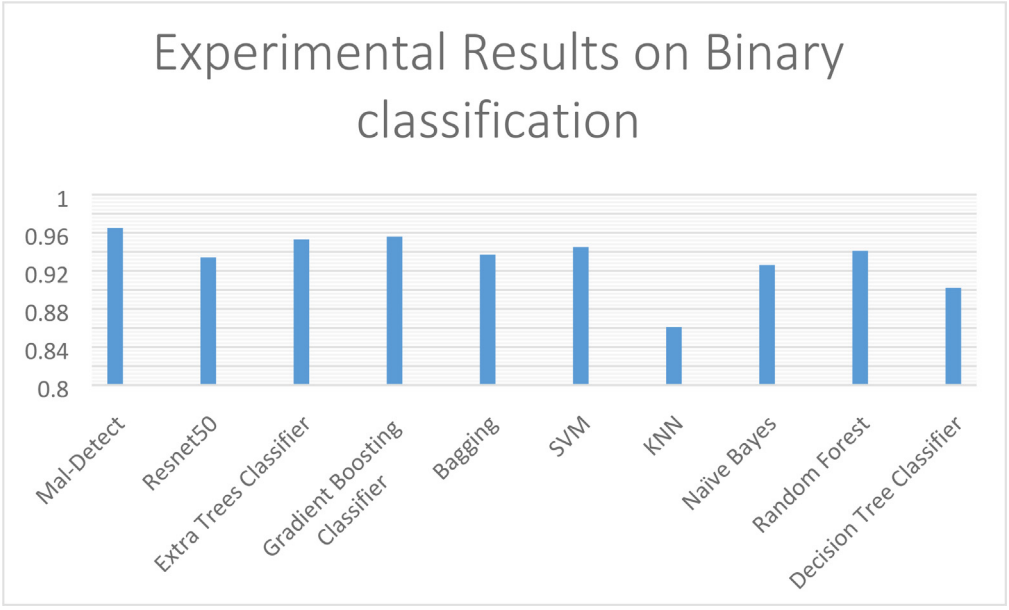

Fig. 8. Binary Classification Comparison.

rics such as Mean Squared Error (MSE), Structural Similarity Index (SSIM) were used to evaluate the generated images.

MSE relays the error value of the generated malware image to the original images. The scores obtained from the evaluation metrics range from 0 to 1; where MSE is best at zero (0). SSIM quantifies image quality degradation caused by processing such as data compression or by losses in data transmission. Just like MSE, SSIM value also ranges from 0 to 1, where 0 is poor and 1 is best.

These generated or novel malware samples were augmented with the original malware dataset to counter adversarial forms of attack,.

b) The Mal-Detect's ability to classify malware and benign labels without requiring a large number of training samples was tested. The experiment was conducted using 2,000 malware samples and 1744 benign samples. The results of the experiment showed that Mal-Detect recorded an accuracy of 96.5% when compared with Resnet50, Extra Trees Classifier, SVM, KNN, Naïve Bayes, Random Forest and Decision Tree classifier as shown in Table 1. Figs. 6 and 7 showed the confusion matrix and the ROC curve of the Mal-Detect compared with Resnet 50. It can be seen from the results that the proposed method had higher accuracy, precision and recall. The result of the binary classification was further presented using a bar chart in Fig. 8 where the proposed Mal-Detect outperformed other machine learning methods with an accuracy of 96.5%. The ensemble classifier Gradient Boosting and Extra Tree Classifier came second and third with an accuracy of 95.6% and 95.3% respectively.

c) A comparison of Mal-Detect with other machine learning techniques was carried out using two benchmark datasets: Mallmg and MaleVis.

i. Mallmg Dataset

To train the Mal-Detect on Mallmg dataset, 10 malware families were selected from a total of 25 malware families. We converted the images from grayscale to RGB by copying the grayscale channels for three slices and downsampled the images to 224 by 224.

Table 2
Accuracy comparison on Mallmg using 224 by 224 pixel.

		Deep Learning Model								Machine Learning								Ensemble			
		Mall-Detect (DCCN-GAN)				Resnet50				SVM				KNN				Bagging			
S/N	Malware Family	**ACC**	P	R	F1	**ACC**	P	R	F1	ACC	P	R	F1	**ACC**	P	R	F1	ACC	P	R	F1
1	Adialer.C	**0.998**	1.00	1.00	1.00	**0.997**	1.00	1.00	1.00	**0.996**	1.00	1.00	1.00	**0.944**	0.66	1.00	0.79	0.987	1.00	1.00	1.00
2	Agent.FYI		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
3	Alueron.gen!J		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00		0.90	1.00	0.95		1.00	1.00	1.00
4	Lolyda.AA1		1.00	1.00	1.00		0.98	1.00	0.99		0.98	1.00	0.99		0.97	0.89	0.93		0.98	1.00	1.00
5	Lolyda.AA2		1.00	1.00	1.00		0.98	0.98	0.98		1.00	0.98	0.99		0.98	0.98	0.99		1.00	1.00	1.00
6	Malex.gen!J		0.97	1.00	0.99		1.00	0.97	0.99		1.00	0.98	0.99		1.00	0.98	0.99		0.96	0.98	0.97
7	Obfuscator.AD		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
8	Rbot!gen		1.00	1.00	1.00		0.97	0.97	0.97		0.98	1.00	0.99		0.96	1.00	0.98		0.98	0.96	0.97
9	Swizzor.gen!I		1.00	1.00	1.00		0.97	1.00	0.99		1.00	1.00	1.00		0.97	0.94	0.95		0.97	0.97	0.97
10	VB.AT		1.00	0.99	1.00		1.00	0.99	1.00		1.00	1.00	1.00		1.00	0.84	0.91		0.99	0.98	0.98

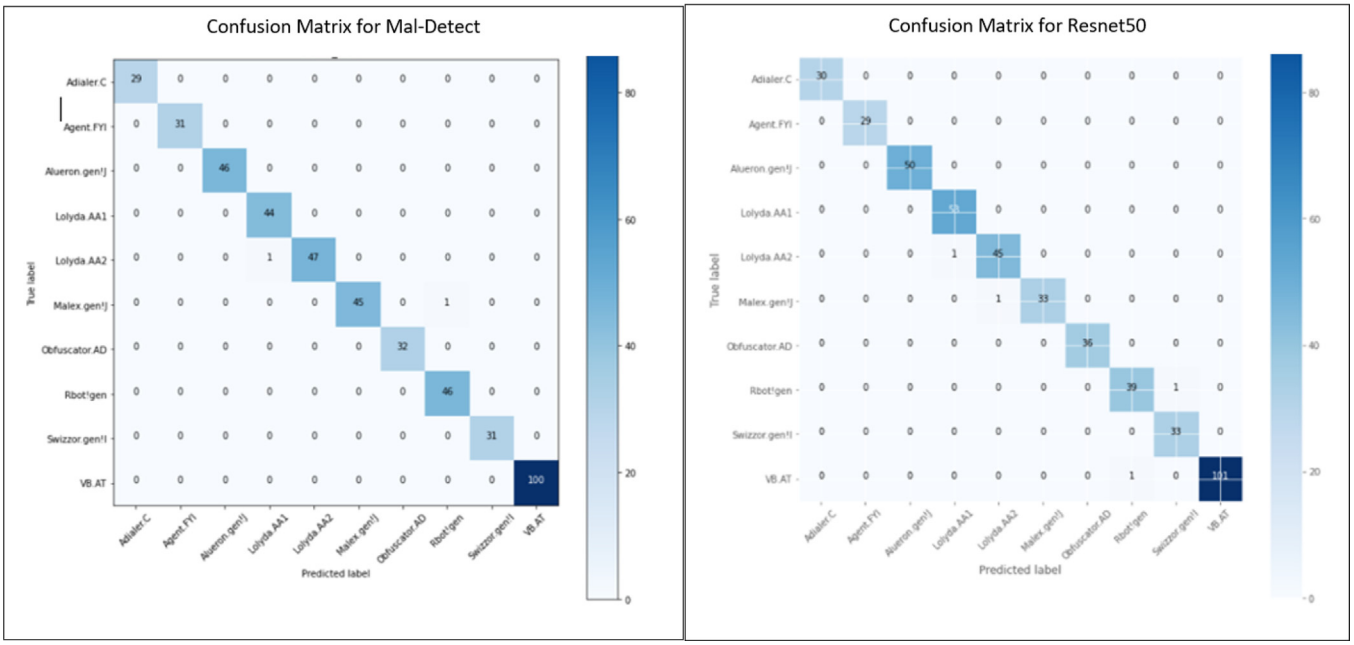

Fig. 9. Confusion Matrix Comparison Matrix on Mallmg Dataset using images of 112 by 112 pixels.

The results showed that Mal-Detect outperformed other detection methods with an accuracy of 99.8% as shown in Table 2. The same conclusion can also be drawn with the Confusion matrix in Fig. 9 and the ROC curve in Fig. 10. The confusion matrix in Fig. 9 shows that Mal-Detect is capable of classifying the malware correctly into different families.

ii. MaleVis Dataset

The same experiment was repeated on the MaleVis dataset which comprised of 25 families each of equal sample size. The Mal-Detect using an image size of 112 by 112 pixels achieved an accuracy of 96.7% while Resnet 50 achieved an accuracy of 91.6%, SVM 88.5%, KNN 78.7 and Bagging 92.6% as presented in Table 3.

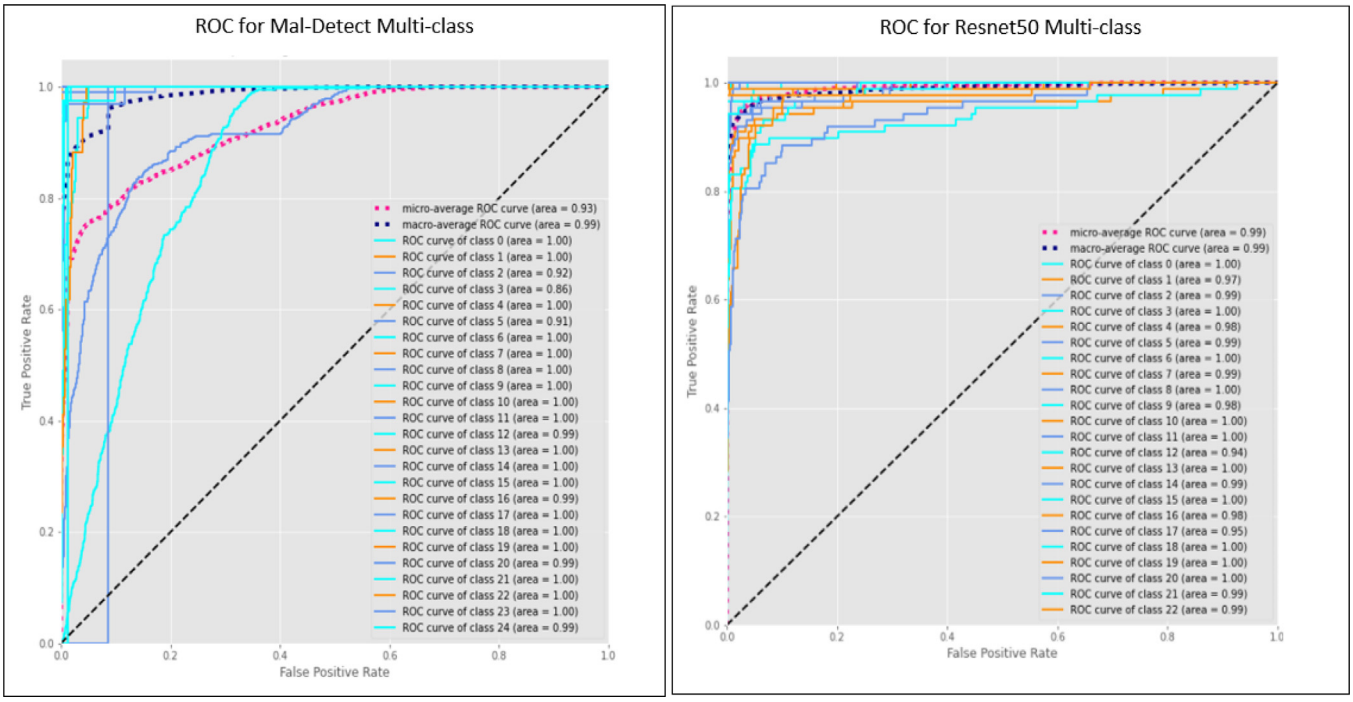

Fig. 10. ROC Comparison on Mallmg Dataset using images of 112 by 112 pixels.

Table 3
Accuracy comparison on MaleVis Dataset 112 by 112 pixel.

		Deep Learning Model								Machine Learning								Ensemble			
		Mal-Detect (DCCN-DGAN)				Resnet50				SVM				KNN				Bagging			
S/N	Malware Family	ACC	P	R	F1	ACC	P	R	F1	ACC	P	R	F1	ACC	P	R	F1	ACC	P	R	F1
1	Adposhel	**0.967**	0.98	1.00	0.99	**0.916**	1.00	1.00	1.00	**0.885**	0.98	1.00	0.99	**0.787**	0.92	0.99	0.95	**0.926**	0.97	1.00	0.98
2	Agent		0.98	0.85	0.88		0.72	0.91	0.88		0.70	0.88	0.78		0.68	0.63	0.65		0.94	0.85	0.89
3	Allaple		0.91	0.93	0.92		0.91	0.93	0.92		0.88	0.80	0.84		0.97	0.66	0.78		0.99	0.89	0.93
4	Amonetize		0.97	0.98	0.97		0.91	0.97	0.94		0.91	0.99	0.95		0.95	0.94	0.94		1.00	0.94	0.97
5	Androm		0.78	0.82	0.81		0.80	0.69	0.74		0.69	0.71	0.70		0.51	0.73	0.60		0.81	0.86	0.83
6	Autorun		0.89	0.89	0.89		0.77	0.83	0.80		0.82	0.76	0.79		0.81	0.60	0.69		0.87	0.70	0.78
7	BrowseFox		0.98	0.98	0.98		0.97	0.94	0.95		0.91	0.96	0.94		0.84	0.91	0.87		0.94	0.98	0.96
8	Dinwod		0.98	0.98	0.98		0.99	0.99	0.99		1.00	0.94	0.97		0.71	0.88	0.79		1.00	0.99	0.99
9	Elex		0.98	0.93	0.95		0.97	0.98	0.97		0.93	1.00	0.96		0.95	0.97	0.96		0.95	0.89	0.92
10	Expiro		0.98	0.93	0.95		0.71	0.80	0.75		0.61	0.75	0.67		0.18	0.05	0.08		0.59	0.89	0.71
11	Fasong		0.99	1.00	0.99		1.00	1.00	1.00		1.00	1.00	1.00		0.98	1.00	0.99		0.99	1.00	0.99
12	HackKMS		0.89	1.00	0.92		0.99	1.00	0.99		1.00	1.00	1.00		1.00	0.99	0.99		1.00	1.00	1.00
13	Hlux		1.00	1.00	1.00		1.00	1.00	1.00		0.99	0.99	0.99		0.98	0.99	0.99		1.00	0.99	0.99
14	Injector		0.96	0.82	0.88		0.88	0.75	0.85		0.65	0.76	0.70		0.58	0.75	0.65		0.91	0.89	0.90
15	InstallCore		0.99	1.00	0.99		1.00	0.98	0.99		0.99	0.99	0.99		1.00	0.76	0.87		1.00	0.99	0.99
16	Multiplug		0.95	0.93	0.94		0.88	0.93	0.91		0.88	0.89	0.88		0.88	0.84	0.86		1.00	0.90	0.95
17	Noreklami		0.98	0.99	0.98		0.97	0.98	0.97		0.94	0.95	0.95		0.98	0.95	0.96		1.00	0.95	0.98
18	Neshta		0.81	0.80	0.80		0.76	0.60	0.67		0.65	0.65	0.65		0.17	0.40	0.24		0.71	0.89	0.79
19	Regrun		0.87	0.99	0.92		0.99	1.00	0.99		1.00	1.00	1.00		1.00	1.00	1.00		1.00	0.99	0.99
20	Sality		0.76	0.63	0.69		0.71	0.60	0.65		0.67	0.38	0.48		0.78	0.07	0.12		0.70	0.66	0.68
21	Snarasite		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00		0.97	1.00	0.98		1.00	1.00	1.00
22	Stantinko		0.99	0.99	0.99		0.99	0.98	0.98		1.00	0.96	0.98		1.00	0.95	0.98		1.00	1.00	1.00
23	VBA		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
24	VBKrupt		0.94	0.93	0.94		0.87	0.95	0.91		1.00	0.91	0.95		0.59	0.83	0.69		0.98	0.96	0.97
25	Vilsel		1.00	0.98	0.99		1.00	0.98	0.99		1.00	0.98	0.99		1.00	0.98	0.99		1.00	0.99	0.99

Fig. 11 shows the confusion matrix for Mal-Detect and Resnet50. The ROC for multiclass classification using Mal-Detect and Resnet50 were also presented in Fig. 12.

d) Mal-Detect was also compared with existing malware classification methods that were based on visualization using APIs, Visualization using grayscale Images and data visualization with adversarial training. Table 4 presents the summarised result of cutting-edged techniques in literature that used visualization features to classify malware.

To benchmark the performance of the proposed method, Mal-Detect was applied on multiclass classification and compared with other state-of-the-art techniques as shown in Fig. 13. The result of Mal-Detect shows high accuracy and robustness in classifying malware due to the combined effects of adversarial malware generation and fine-tuned deep convolutional neural network.

4. Conclusion and future work

As malware writers continue to adopt different techniques for evading detection, this study developed a system named Mal-Detect for analysing and categorizing malware. First, both malware

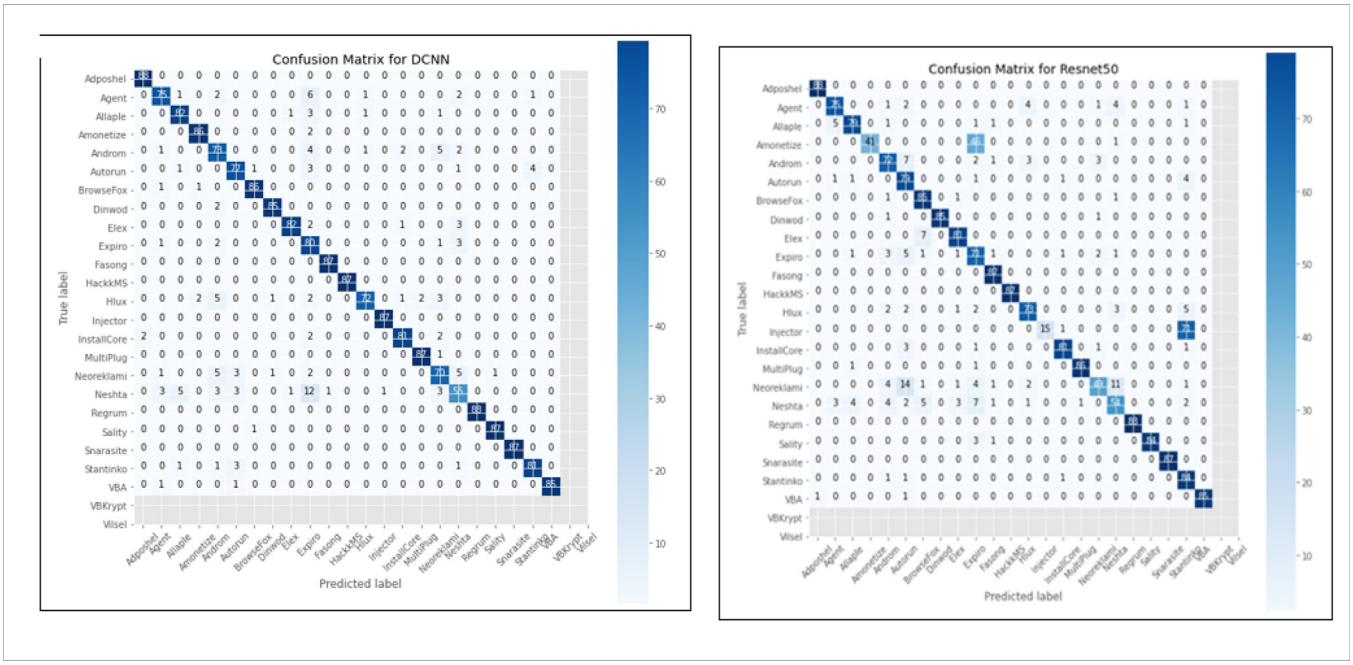

Fig. 11. Confusion Matrix for MaleVis Dataset using images of 112 by 112 pixels on Mal-Detect and Resnet50.

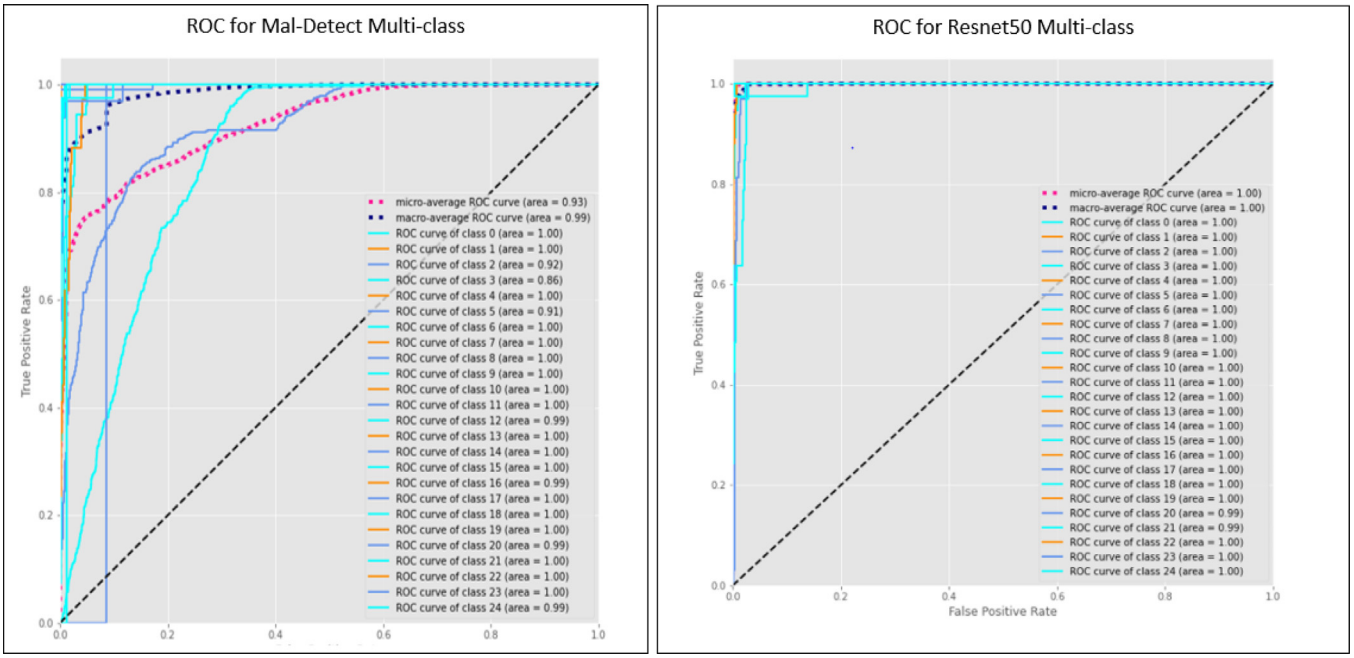

Fig. 12. ROC comparison for MaleVis Dataset using images of 112 by 112 pixels.

Table 4
A comparative summary with existing work.

Approach	Dataset	Feature type	Classification Algorithm	Training Time	Predicting time	Accuracy
Visualization using APIs						
Gibert et al. (2020)	Microsoft Malware Dataset challenge	APIs, Bytes sequence, Opcode sequence	Multimodal Deep Neural Network	–	–	**0.9975**
D'Angelo et al. (2020)	Malgenome, Contagio Minidump, VirusShare	API-images)	Autoencoders	–	–	0.94
Visualization using Gray Scale Images						
Vasan et al. (2020a), Vasan et al. (2020b)	Mallmg	Grayscale images	Ensemble of VGG16, Resnet50 and SVM		1.18 s per sample out of 25 samples	0.9950
Nataraj et al. (2011)	Mallmg	Grayscale images	GIST			0.9718
Data visualization with adversarial training						
Liu et al. (2020)	Microsoft Malware Dataset challenge	Grayscale images				0.9625
Mal-Detect	Custom Dataset (Malware + Benign	RGB Images	DGAN, DCNN	1224.296 s for 50 epochs	0.91 s	0.967
	Malimg	Grayscale to RGB Images	DGAN, DCNN	400.97 s for 50 epochs	0.5232	0.998
	MaleVis	RGB Images	DGAN, DCNN	1799.465 s for 50 epochs	4.561 s	0.967

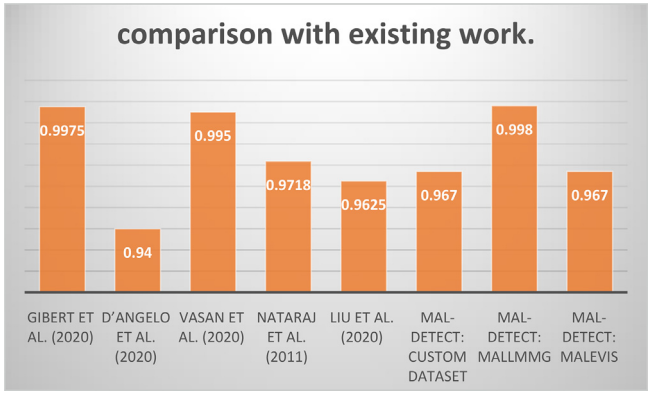

Fig. 13. Comparison with existing work.

files and benign files were visualized as RGB images. Next, DGAN was used to generate novel malware from the train malware datasets in order to prevent the adversarial attack. The generated malware samples and the original malware dataset from train samples were augmented for training the model. DCNN is applied for training and categorising the malware and benign features into different malware families. Finally, the accuracy of Mal-Detect was compared with other existing state-of-the-art techniques in malware visualization. The results showed that using Mal-Detect improves the accuracy and efficiency of all the experiments. The findings of this study have implications for visualizing and detecting novel malware. In the future work, we intend to test Mal-Detect against larger datasets, as well as integrate a design of the proposed framework in an IoT based system for assessment and precision.

CRediT authorship contribution statement

Olorunjube James Falana: . **Adesina Simon Sodiya:** Supervision, Methodology, Conceptualization. **Saidat Adebukola Onashoga:** . **Biodun Surajudeen Badmus:** .

Declaration of Competing Interest

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

References

Alam, S., Horspool, R.N., Traore, I., Sogukpinar, I., 2015. A framework for metamorphic malware analysis and real-time detection. Comput. Secur. 48, 212–233.
Almarri, S., Sant, P., 2014. Optimised malware detection in digital forensics. Internat. J. Network Secur. Appl. 6 (1), 01–15.
D'Angelo, G., Ficco, M., Palmieri, F., 2020. Malware detection in mobile environments based on Autoencoders and API-images. J. Parallel Distrib. Comput. 137, 26–33.
Ding, Y., Xia, X., Chen, S., Li, Y., 2018. A malware detection method based on family behavior graph. Comput. Secur. 73, 73–86.
Firch, J., 2021. Cyber Security Statistics, Data, & Trends. PurpleSec LLC, Vienna, Virginia.
Gibert, D., Mateu, C., Planes, J., 2020. HYDRA: A multimodal deep learning framework for malware classification. Comput. Secur. 95, 101873.
Grosse, K., Papernot, N., Manoharan, P., Backes, M., McDaniel, P., 2017. Adversarial examples for malware detection. In: Paper Presented at the European Symposium On Research In Computer Security, pp. 62–79.
Hemalatha, J., Roseline, S.A., Geetha, S., Kadry, S., Damaševičius, R., 2021. An efficient DenseNet-based deep learning model for malware detection. Entropy 23 (3), 344.
Kang, B., Yerima, S. Y., Sezer, S., & McLaughlin, K. (2016). N-gram opcode analysis for android malware detection. *arXiv preprint arXiv:1612.01445*.
Kumar, S., 2020. An emerging threat Fileless malware: a survey and research challenges. Cybersecurity 3 (1), 1–12.
Liu, X., Lin, Y., Li, H., Zhang, J., 2020. A novel method for malware detection on ML-based visualization technique. Comput. Secur. 89, 101682.
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, 102154.
Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B., 2011. Malware images: visualization and automatic classification. In: Paper presented at the Proceedings of the 8th International Symposium On Visualization For Cyber Security, p. 4.
Ni, S., Qian, Q., Zhang, R., 2018. Malware identification using visualization images and deep learning. Comput. Secur. 77, 871–885.
Otsu, N., 1979. A thresholding selection method from gray-scale histogram. IEEE Trans. Syst. Man Cybern. 9, 62–66.
Potter, B., Day, G., 2009. The effectiveness of anti-malware tools. Computer Fraud & Security 2009 (3), 12–13.
Santos, I., Brezo, F., Ugarte-Pedrero, X., Bringas, P.G., 2013. Opcode sequences as representation of executables for data-mining-based unknown malware detection. Inf. Sci. 231, 64–82.
Sodiya, A., Falana, O., Onashoga, S., Badmus, B., 2014. Adaptive neuro-fuzzy system for malware detection. J. Comput. Sci. Appl. 21 (2), 20–31.

Symantec. (2021). Cyber Security Predictions-Looking toward the future.
Vasan, D., Alazab, M., Wassan, S., Naeem, H., Safaei, B., Zheng, Q., 2020a. IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture. Comput. Networks 171.
Vasan, D., Alazab, M., Wassan, S., Safaei, B., Zheng, Q., 2020b. Image-Based malware classification using ensemble of CNN architectures (IMCEC). Comput. Secur. 92.
Wu, J., 2017. Introduction to convolutional neural networks. In: National Key Lab for Novel Software Technology. Nanjing University, China, p. 495.
Xin, Q., Hu, S., Liu, S., Lv, H., Cong, S., Wang, Q., 2020. Fruit image recognition based on census transform and deep belief networkPaper. In: presented at the International Conference on Multimedia Technology and Enhanced Learning, pp. 438–446.
Yuan, X., He, P., Zhu, Q., Li, X., 2019. Adversarial examples: Attacks and defenses for deep learning. IEEE Trans. Neural Networks Learn. Syst. 30 (9), 2805–2824.