# An Intelligent Malware Classification Model Based on Image Transformation

Mohamed Abo Rizka[1], Mohamed Hamed[2], Hatem A. Khater[3]
College of Computing and Information Technology-Heliopolis Campus,
Arab Academy for Science, Technology & Maritime Transport, Cairo, Egypt[1, 2]
Electrical Department-Faculty of Engineering, Horus University Egypt, New Damietta 34518, Egypt[3]

***Abstract*—Due to financial incentives, the number of malware infections is steadily rising. Accuracy and effectiveness are essential because malware detection systems serve as the first line of defense against harmful attacks. A zero-day vulnerability is a hole in the target operating system, device driver, application, or other tools employing a computer environment that was previously unknown to anybody other than the hacker. Traditional malware detection systems usually use conventional machine learning algorithms, which call for time-consuming and error-prone feature gathering and extraction. Convolutional neural networks (CNNs) have been demonstrated to outperform conventional learning techniques in a number of applications, including the classification of images. This success prompts us to suggest a CNN-based malware categorization architecture. We evaluated our methodology using a bigger dataset made up of 25 families within a corpus of 9342 malware. Last but not least, comparisons are made between the model's measurement and performance with other cutting-edge deep learning techniques. The overall testing accuracy of 98.31% in the provided results attested to the excellent accuracy and robustness of the suggested procedure at a lower computational cost.**

***Keywords*—Malware Classification; zero-day; Convolutional Neural Networks (CNN); grayscale image transformation; Bytehist**

## I. INTRODUCTION

The quick enhancement of communication and information technologies has had a significant impact on cyber security. Systems and techniques for spotting intrusions and preventing them have significantly advanced. Even with more advanced security measures in place, hackers continue to develop methods to identify weaknesses and seize control of devices and systems. Static analysis approaches, like signature-driven method, pattern-match method, or data mining technology, examine the data inside the file to determine whether an executable portable file contains a programme that shouldn't be launched. The goal of the dynamic analysis method, which involves running the malware itself, is to observe how the portable executable file behaves while it is in use [1][2][3]. Methods for detection and classification were greatly hampered by the attackers' knowledge of infiltration tactics and strategies. Known as zero-day vulnerabilities and zero-day assaults, one of the most popular attack types in use today is malware [1]. The academic community and the security business have employed deep learning, machine learning, and intelligent systems to try and forecast potentially risky conduct. The first quarter of 2021 saw a 68.9% spike in new PowerShell malware and a 41.2% increase in business malware compared to the previous quarter [3]. The aforementioned statistics demonstrate that researchers in information technology disciplines have started to see IT as applying machine learning-based (neural language) detection and classification algorithms and NL processing to sort through the ever-increasing volume of malware and cunning escape strategies being deployed [3][4]. Due to the frequent requirement for traditional malware research approaches to design crucial traits, which costs money and time, machine learning has been found to be more effective [5]. Malware categorization has also been effectively accomplished using (CNN). The first stage of this scientific study will involve converting files from a regular image format to binary language, which will then be translated into grayscale images. Second, the files will be grouped into families of harmful programs. Twenty five (25) families of Trojan horses, malware, backdoors, etc. are negatively impacted. The accuracy in the identification and categorization of files into families of healthy files and families of hazardous files that will be used is then calculated by dividing a portion of these files into a portion for learning by 80% and a portion for testing by 20%.

This paper's primary divides are as follows: Section I presents the introduction; the Section II shows the related work, Section III introduces datasets, Section IV establishes the proposed AI algorithm and Section V introduces the proposed DL architecture; the experiment's results are laid out in Section VI, along with discussions; the conclusion is given in Section VII.

These are this paper's significant contributions:

*1)* Using a number of pre-learned CNN algorithms based on image modification, we suggest a supervised learning-deep technique to categorize malware.

*2)* Check files in the initial phase to verify whether hashing, signature, or encryption modifications have been made, and use the modified metric to create byte-usage-histograms for whole sorts of codes with an emphasis on binary executables in aportable executable (PE) presentation.

*3)* Utilize thedeveloped tool to transform any executable or binary file into a greyscale PNG image that can be seen in the range [0,255]. We offered a useful paradigm for handling data from an imbalanced dataset.

*4)* We conducted numerous tests to contrast our approach for classifying malware with a number of existing techniques;

the findings show that our approach works better than these other approaches.

*5)* In order to classify malware, we created a regularized strategy that performs better than competing models despite learning from a tiny dataset.

## II. Comparable Works

The most common malicious attempts on biometric technology are probably those on records with pattern data. The model contains a user's biometric details that might be abused in an assault. The confidentiality of the user is put at risk by the availability of patterns across multiple programmes [27]. Therefore, a strong technique is needed to protect the forms kept in the database. The following specifications [8] through [10][29] ought to be met by the most suitable pattern safety solution.

### A. Strategies for Analyzing Malware

Malware analysis comes in two sorts: static and dynamic. Malware analysis aims to comprehend the composition and operation of malware [3]. Malware samples must be examined in order to ascertain their nature and mode of operation [6]. Static and dynamic analysis are the dual basic techniques utilized to detect malware. This is so that malware can be identified during analysis, which enables the resolution of a number of issues, including the presentation of the harmful architecture, the detection of infections and propagation techniques, and the assessment of the specific harm to the victim's devices [6]. The dual chief methods of malware analysis are static and dynamic. While studying malware, basic static analysis is done first, followed by advanced dynamic analysis [7].

### B. Static Analysis

In order to do static analysis on Windows portable executable (PE) records, either the binary file or the malware program that has been disassembled must be used. The most popular programs for opening PE files are IDA Pro and Radar. This kind of reverse engineering can be applied to them.

Without running the malware code, static analysis can reveal the structure of a malware sample [8]. The two parts are fundamental static analysis and enhanced static analysis. Without going deeper, elementary static analysis inspects the programs, assessing file content, header information, and functions [6]. Among the tools that be able to utilize to abstract that information are PEiD, Bin Text, MD5deep, and PE view [7]. The first step in malware analysis is basic static analysis; advanced static analysis should be carried out to learn more about malware. The advanced static analysis does a complete study of the program directives.

To accomplish this, assembly codes are generated from machine codes using a disassembler [6] [9]. For thorough static analysis, researchers typically utilize the IDA Pro packet splitter and the supplemental Hex-Rays de-compiler. The investigation is thoroughly scrutinized to look for signs of malice in the procedures for assembling. With the use of sophisticated static analysis and inverse compilation, specific malware functionality may be retrieved. The advanced static analysis offers an in-depth understanding of the functionality and intent of malware. However, a thorough understanding of operating system principles and assembly code instructions is required for this subject [6][11].

### C. Dynamic Analysis

Allows us to monitor its behavior and gather all of the virus's traces as we perform the dynamic analysis. This study is often utilized as a secondary analysis to have additional parameters or if we were unable to gather significant information by Employing static analysis, the malware infection developer's considerable obfuscation. This scan should be carried out in a totally isolated setting to prevent damaging our system. There are several habitats to pick from, with Cuckoo Sandbox being the most well-known. They provide an overview of both the methods used for each type of study and the data that was extracted [1].

Due to the dynamic analysis' use of program execution, malware behavior analysis was done. To prevent infection of the devices, the analysis is done in enclosed environments like sandboxes or virtual PCs. Examining the execution of functions, arguments, data transfers, modifications to the file database, and network usage are all part of the process. When describing the actual operation of malware, static analysis is less accurate than dynamic analysis. There are two different kinds of dynamic analysis: basic and advanced. Basic dynamic analysis is used to examine the behavior of malware [10]. Utilizing Sandboxes, Regshot, ApateDNS, Procedure Explorer, API observe, and Procedure Monitor. The extensive dynamic analysis employs tools for debugging like WinDbg and OllyDbg. Experts who study malware can use debuggers to examine and modify the outcomes of individual commands.

### D. Analysis of Statistics and Dynamics

The static analysis makes it simple and quick to evaluate earlier detected malware and gain a quick summary of the software [35]. Unfortunately, it is incredibly difficult to analyze malware that employs obfuscation, packing, polymorphism, and other techniques. Because dynamic analysis involves computer programs, malicious software may be employed. Obfuscation techniques used by malicious software can be recognized. Certain malware variants, however, might be aware that they are being tested in sandboxes and virtual environments, which would conceal their genuine behavior. Dynamic analysis is more efficient when dealing with unknown malware, despite the fact that static analysis is quicker and more precise when dealing with already identified malware [12].

### E. Machine Learning Techniques

The two methods utilized in ML are unsupervised learning, which involves identifying hidden patterns or internal frameworks in incoming data, and supervised learning. To be able to forecast future outcomes, supervised learning requires training an algorithm utilizing available data for both input and output.

### F. Supervised Learning

An algorithm that generates forecasts using data in the presence of unpredictability is created through supervised ML. A technique for supervised learning employs a set of

predetermined input data to predict results. A supervised learning approach teaches the model to produce accurate forecasts in response to new data using a well-known collection of input data and identified reactions to data (output data). If the outcomes you are attempting to forecast have known data, use supervised learning. To build ML frameworks, supervised learning uses regression and classification techniques. Classification techniques predict specific outcomes, such as if an email is real or spam, or if a tumor is malignant or not. The given data are categorized by Common uses including speech recognition, credit scoring, and medical visualization. When you can tag, classify, or divide your data into distinct groups or classes, use classification. For instance, categorization is used by a handwriting recognition program to identify letters and digits [14][15]. Unsupervised pattern recognition algorithms are used in image processing for object recognition and image segmentation. Some commonly popular classification techniques include (SVMs), KNN, nave bays, differential analysis, logistic regression, and NN [17]. Techniques for regression forecast continuous responses like variations in electricity consumption and temperature. Forecasting electricity load and algorithm trading are examples of common applications. Nonlinear models, linear models, progressive organization, regression, reinforced and packed decision trees, adaptive neuro-fuzzy learning, and neural networks are examples of common regression techniques [18].

### G. Unsupervised Learning

Data is scanned for underlying structures or hidden patterns using unsupervised learning. From datasets without any marked responses, it is utilized to draw conclusions. Clustering is the technique used most often in unsupervised learning. To find undiscovered patterns or groupings in the data, it is employed in exploratory data analysis. Object recognition and DNA arrangement analysis are a few instances for use for cluster analysis. For instance, a smartphone provider can use machine learning to determine how many different groups of people rely on its towers in order to optimize where it places its cell towers [19]. Because mobile phones can only communicate to one station at a time, the crew used a clustering approach to identify the best locations for cell sites to improve the reception of signals for their customer sets or clusters. Typical clustering methods include clustering based on hierarchy, GM systems, self-organizing maps, HMM, subtractive clustering, fuzzy c-means clustering, and k-medoids and k-means [20].

### H. Deep Learning Approaches

DL models are from time to time referred to as DNN since the majority of deep learning methods employ NN architectures. DNNs are simply neural networks that have a lot of hidden layers. While DNN can have up to 150 hidden layers, conventional NN is limited to two or three. Large volumes of categorized data and NN topologies that acquire parameters from the data before learning them are employed for building algorithms for DL [21].

CNN or ConvNet are among the most popular DNN kinds. Specifically, a CNN is well suited for analyzing 2D data, such as photographs, because it mixes learned features with incoming data and makes use of 2D convolutional layers. You won't have to figure out what characteristics are used to classify photos because CNNs do manner with the need for non-automatic parameter extraction. CNN uses direct feature extraction from images to run its business. The necessary features are not pre-trained; rather, they emerge when the network trains on a batch of images. For computer vision applications like object categorization, deep learning models are especially accurate [22].

CNNs are taught to recognize various features of a picture using considerable hidden layers. The complexity of the learned visual elements increases with each buried layer [16]. For instance, the initial hidden layer might train to recognize edges, while the final layer might learn to recognize more intricate forms that are particular to the form of the object, we're able to recognize. In conclusion, because they typically identify and extract a set of parameters in advance and are not built to handle vast volumes of data, conventional ML algorithms have a great complex cost. The technique of DL on the contrary, performs the extraction of features and selection, cutting down on considerable computational costs. Yet, studies have shown that DL is superior to ML in terms of effectiveness and accuracy.

## III. Dataset

We assessed our method on a big dataset containing 25 families in malware groups of 9,342. Nataraj et al. contributed MalImg collection [28]. The assessment outcomes display that our technique presents high precision with less computational cost. Moore's details are demonstrated in Fig. 1.

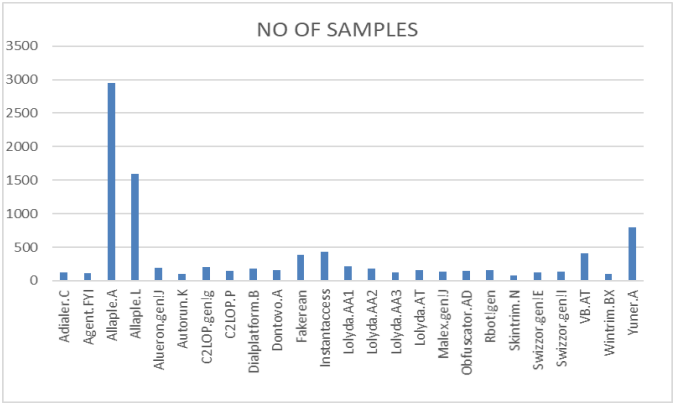


Fig. 1. Malware families found in the MalImg dataset.

## IV. Algorithms

We discuss the dataset processing and implementation specifics for the proposed mathematical frameworks in this part of the article. We use the Bytehist software to generate byte-usage histograms for a variety of records, including binary executables in (PE) presentation. Using the Bytehist tool [23], we check files in the initial phase to see if hashing, signature, or encryption changes have been applied. We have divided malware from the MalImg Dataset into a variety of classifications. We use CNN to train and test the DL system for identifying and classifying malware in 25 families of images, and families of grayscale images.

### A. Modified Binary File Detection

Static analysis frequently encounters problems while analyzing compressed or encrypted executables. AI algorithms regularly identify harmful executables as safe, despite the fact that many of them are updated to fulfill business or intellectual property objectives. This is understandable given that these modifications would significantly change the executable's entropy and byte spread. When creating a performance-improving technique for detection models, take the likelihood of binary file modifications into account. The tool ByteHist may produce byte-usage histograms for a variety of file formats, with a concentration on binary executables in the PE format. For instance, ByteHist [23] offers information into the nature of data before an examination. We can look at how bytes are distributed within a program that runs using ByteHist.

The distribution gets extra even with each executable compression. Examples of both negative and positive analogs are shown in Fig. 2, together with unpacked and UPX-transformed byte distributions. As shown, UPX alters the byte spread of the binary file, especially when malware is present. It is also a widely used packer and binary unpacking is straightforward, in contrast to neutral le, which has more altered cations [24]. UPX generates less. But a lot of malware comes with more sophisticated software, which complicates the investigation. Statistics could be a useful tool for locating encrypted or compressed data.

Bytes in the data are dispersed quite uniformly as a result of this type of alteration. Typical data typically consists of specific bytes that are constantly in use due to any type of structure. The byte distributions of database files, executable binaries, and plain text that haven't been encrypted or compressed differ significantly from those of those that have. This "phenomenon" is displayed using histograms, which make it easy to distinguish between the two.

### B. Employing Images to represent Malware

The objective of this research is to visualize malware using a technique created by Nataraj et al. (2011) that enables a malware binary to be read as a stream of 8-bit integers without signs before being structured into a 2-dimensional matrix. Our tool transforms any executable or binary file into a greyscale PNG image that can be seen in the domain [0,255] (0: black, 255: white) [14]. Malware presentation as a grayscale picture process is shown in Fig. 3 Due to the method's reliance on binary code, a new infection might be created by a malware producer by updating the code of an existing virus, which would result in a very similar image being used to display the new infection. Then, we may use our classification model (CNN), which will be illustrated later, to put it all into one family.

### C. Using Transfer Learning to Classify Malware

DL is a branch of ML that includes algorithms designed to mimic the operation of neural networks or the human brain. These structures go by the term neural networks. It trains the computer to perform actions that come naturally to people. Some of the models used in deep learning include autoencoders, recurrent neural networks (RNN), (ANN), and reinforcement learning. Convolutional Neural Networks (CNN) or ConvNet, in particular, have significantly advanced the areas of computer vision and image analysis [13]. CNNs, a subcategory of DNN, are often utilized for image analysis as they able recognize and categorize certain structures in frames. They have a variety of uses. Only a few of their applications include picture and video recognition, image classification and NLP. Fig. 4 concludes the convolutional neural networks' historical development.

### D. CNN'S Principal Architecture

According to Fig. 5, there are two parts to CNN architecture [25].

- Feature Extraction: A convolution tool isolates and classifies the distinctive features of an image for examination throughout the feature extraction process.
- Fully connected: A completely connected layer that predicts the frame's group applying the data collected in earlier steps and the convolution procedure's output.

### E. Convolution Layers

The CNN is consisted of three distinct kinds of layers: completely connected (FC), convolutional, and pooling layers. The CNN architecture will be built by stacking these layers. Additionally, to these three layers, there are two more crucial necessities: the dropout layer and the activation function.


(a) File unpacked. (b) File packed.

Fig. 2. Using ByteHist, compare the byte spread of normal and malicious programs.


Fig. 3. Malware as a procedure for grayscale representations.

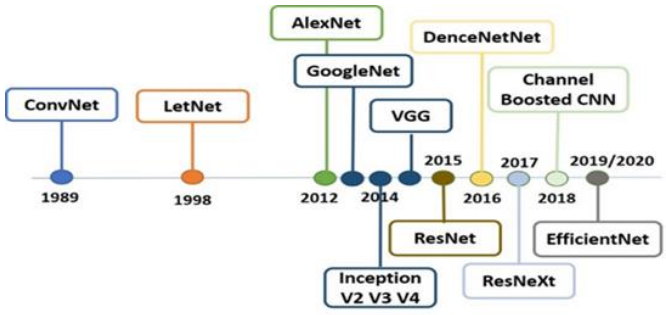


Fig. 4. A brief history of convolutional neural network.

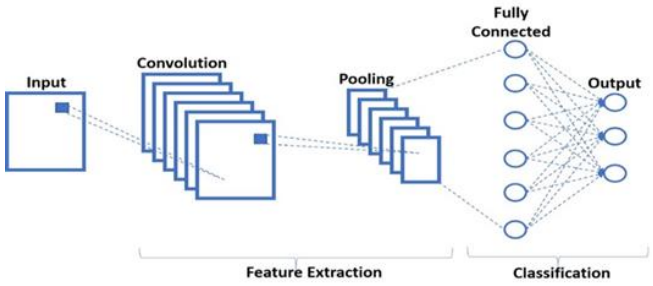


Fig. 5. CNN's elementary architecture.

### F. Convolutional Layer

This is the first layer separating the many structures from the entrance frame. In this layer, the beginning picture is mathematically convolutional using a filter of a certain size MxM. The number of dots that exist across the filter and various areas of the given picture can be calculated according to the filter's size (MxM) by moving the filter through the frame. The final outcome, also referred to as the feature map, contains information about the picture, such as its contours and borders. Then, further layers receive this feature map, which they use to pick up additional features from the input image.

### G. Pooling Layer

A Pooling Layer is frequently used after a Convolutional Layer. This layer's primary goal is to scale down the convolved feature map in order to save processing expenses. This is accomplished by minimising the connections within layers and working independently on every element map. There are multiple sorts of pooling techniques based on the technology employed.

The Max Pooling feature map is used to determine the largest element. With middling pooling, the elements within an image segment of a specific size are averaged. The cumulative total of the elements in the pre-known segment is estimated using totality pooling. Connecting the Convolutional Layer and the FC Layer is frequently done via the Pooling Layer.

### H. Complete Layer Connectivity

Weights and biases are included in the Fully Connected (FC) layer, which connects the neurons amongst layers. The resulting layer is frequently positioned before the last few layers in a CNN architecture. The input pictures from the layers above are now smoothed and sent to the FC layer. The standard theoretical useful procedures are then performed on the flattened vector via a few additional FC levels. The classifying procedure officially starts at this point.

### I. Dropout

When all of the characteristics are linked to the FC layer, the learning dataset is susceptible to excessive fitting. Overfitting is the process of an algorithm doing such well on data used for training that it has a detrimental impact on how well it works on fresh data. In order to tackle this issue, a dropout layer is implemented, which results in a smaller model by eliminating a limited neurons from the NN throughout learning. After achieving a dropout of 0.3, 30% of the nodes in the NN discontinue arbitrarily.

### J. Activation Functions

To summarize, the activation function of the CNN framework is one of its most crucial elements. Any kind of persistent and complicated network variable-to-variable linkage is learned and approximated using them. It decides which design data the network terminal ought to convey as well as which ought not to, to put it simply. The network gains linearity as a result. The ReLU, Softmax, tanH, and sigmoid process are some of the most frequently utilized activation functions. Each of these functions has a unique use. While softmax is frequently employed for a variety of classes sigmoid and softmax functions are chosen for a CNN algorithm for binary classification [26].

### K. Proposed Malware Classification Algorithm

The analytical pipeline of the suggested architecture is introduced in Fig. 6 and includes various processing phases. The first step involves preparation of along with information enhancement, which involves changing files from a common picture format to binary language and then back again to grayscale images. Following this, the established CNN framework is described along with its details, including learning through transfer, learning models, variable adjustment, and ultimately categorization. The specifics of those phases are then extensively explained.

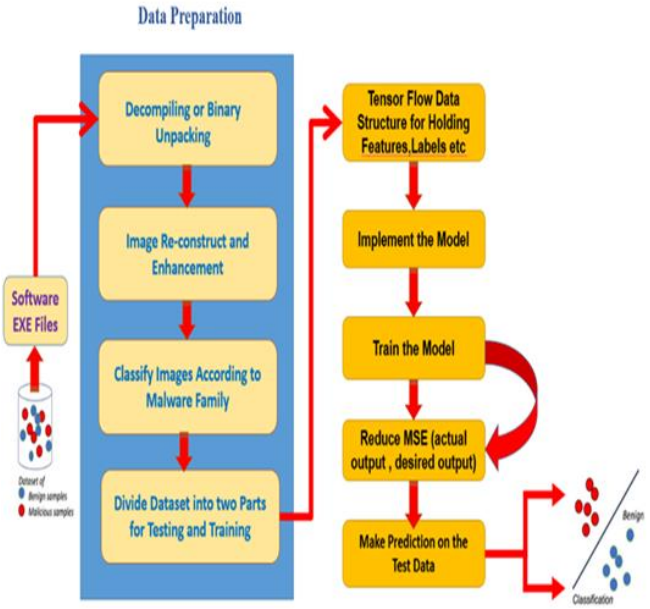


Fig. 6. Illustration depicting the suggested model for the analysis.

## V. Speculated Deep Learning Framework

The proposed DL architecture comprises many steps in which, following preprocessing, the pictures are provided to the suggested CNN for testing and learning in a 64 × 64 array dimension. The proposed CNN structure consists of a source of input, an amount of intended layers, and an output. In this research, five 2D layers of convolution were specifically used, each of which had a 2D max-pooling layer [33]. Convolution is a linear procedure between the input and a kernel (or filter) that acts as an operational monitor. The filters are designed to

extract certain information from photos and have a constrained response region. The convolution layer is identified as follows:

$$X_n^r = \alpha\left(\sum_{m=1}^{k} X_m^{r-1} * w^r{}_{mn} + \dot{b}_m\right) \quad (1)$$

The current layer's (r^th) activation map is characterized by X_n^r, the previous layer's (r-1)^th activation map is represented by X_m^(r-1), and how many enter activation maps are there, is indicated by k. The weight and bias vectors are (w^r)_mn and b^r_m, respectively. Convolution is performed using the * operator, and (α) stands for the function of activation.

After each generated activation pattern has been activated by the function of activation, it is subsequently transported to the layer of pooling. The layer of pooling produces a transformation constant by reducing the overall quality of the activation maps and the layer of pooling activations is produced by the convolution layer's dxd (for example, d=2) structure of activation maps. The pooling technique that is most frequently employed is max pooling. The fully connected layer uses the data from all of the activation maps from the layer before it to create a categorization map. The optimizer is essential in learning the DCNN algorithm since it continually modifies the network's layer settings.

To attempt to reduce the effect of the loss function ($i \cdot e \cdot \nabla\_\theta L(\theta)$), the settings are modified in the contrary orientation of the variation of the loss function (i.e., L(θ) compared to the variables). Following every repetition, the intended and forecast outputs are contrasted, and the mistake is back propagated. One of the greatest commonly employed evaluation of performance measures is cross-entropy. The basic objective of any optimisation technique is to have a cross-entropy score that is almost zero while the desired and predicted results are the same.

These models will locally identify patterns as CNNs operate internally using convolutions in several sliding windows, enabling a robust differentiation between how every category is represented. The layer of dropouts has been modified to 0.25 for the first and succeeding layers of convolution and to 0.3 for the following layer of convolution. Reformed Linear Units, or ReLUs, serve as the activation function for every layer of convolution. The framework may identify patterns in the provided data and transfer those patterns onto subsequent levels. Following adjusting, the outcome of the preceding convolution is sent to the final dual layers, a full connectivity (FC) layer with 0.2 dropouts and a softmax layer with four neurons. The layer of networks responsible for categorizing determines the likelihood that a data source will fit into a certain classification. For analysis of time-series data employing pooling and expanding filter dimensions ranging this type of multi-layer architecture has shown to be effective. [25][26].

The outcome patterns are y= y1, y2... ym, while the given input patterns for the model are x= x1, x2... xn. The result of the last layer of the network was improved by means of the cost array (xi). If y is the result of every specific method, (L) is the value of the loss function, ($\partial i$) is the result after the following adjustments and (₵) is the desired class, then (y) is the result.

$$\partial^i = L(\xi ¢, y⍰), : \partial⍰¢ \geqslant \partial_j⍰ \forall j \neq ₵ \quad (2)$$

The loss function has been changed to:

$$L = \sum_n t_n \log(\partial_n) \quad (3)$$

Where $\partial_n$ contains the cost that is class-dependent (ξ) and is associated with the result on ($y_n$).

$$\partial_n = \frac{\xi ¢_n exp(y_n)}{\Sigma_k \xi ¢_k \exp(y_k)} \quad (4)$$

The quantity of samples in a class determines how much weight it has. If class Ŋ has t times extra trials than class p, making one trial from class p as significant as t samples from class Ŋ is the goal. Therefore, the class weight of p is t times more than the class weight of Ŋ. We employ 2D convolutional layers in our model, which is depicted in Fig. 7, using 3x3 kernels for each of the subsequent blocks and 5x5 kernels for the initial block. Moreover, we employ 2x2 for the final two blocks. Every block's second layer of convolution used the ReLU activation function while down-sampled with a stride of two. The first block contained 64 filters, and every block after which included double number of filters. A layer of dropouts (p = 0.3) was added after the last convolutional layer had been applied and connected to one FC-dense layer with ReLU activation scores of 1024.

There was also a layer of dropouts (p = 0.3) sandwiched in among those thick layers. Finally, the algorithm result was provided by a softmax-activated multi-dense neuron. The Adam optimizer was used to learn the algorithm for up to 100 epochs at a rate of learning of 0.001, utilizing a batch dimension of 40. Additionally utilized was the class cross-entropy process of loss that is often employed for several classes' problems with categorization. The class cross-entropy is described as follows, using p standing for the actual distribution and q for the calculated distribution:

$$H(p,q) = -\sum_x p(x) \log\big(q(x)\big) \quad (5)$$

The suggested deep learning pipeline's parameters are recorded in Table I as a whole.

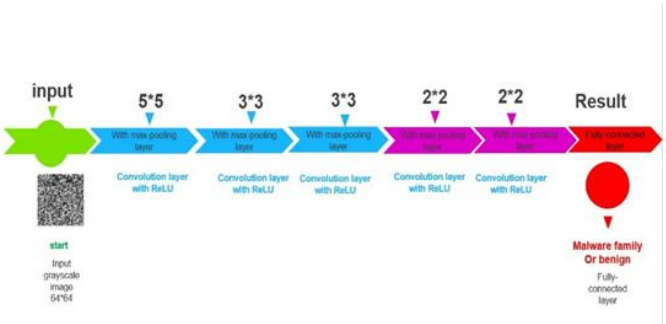


Fig. 7. CNN algorithm structure utilized for malware classification.

TABLE I. SETTINGS FOR THE SUGGESTED SYSTEM PARAMETER

| Layer | First Layer | Second Layer | Third Layer | Fourth Layer | Fifth Layer |
|---|---|---|---|---|---|
| Convolution | filter =64 Kernel_size=(5.5), padding='Same', activation ='relu' | filter =128 Kernel_size=(3.3), padding='Same', activation ='relu' | filter =128 Kernel_size=(3.3), padding='Same', activation ='relu' | filter =128 Kernel_size=(2.2), padding='Same', activation ='relu' | filter =128 Kernel_size=(2.2), padding='Same', activation ='relu' |
| Max pooling | pool_size=(2,2) | pool_size=(2,2), strides=(2.2) | pool_size=(2,2), strides=(2.2) | pool_size=(2,2), strides=(2.2) | pool_size=(2,2), strides=(2.2) |
| Dropout | (0.25) | (0.25) | (0.3) | (0.3) | (0.3) |
| Bach Size | 256 | 256 | 256 | 256 | 256 |
| Learning Rate | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| Optimizer | Adam | Adam | Adam | Adam | Adam |
| No.of Epochs | 100 | 100 | 100 | 100 | 100 |
| Total Parameters | 4619524 | 4619524 | 4619524 | 4619524 | 4619524 |
| Trainable Parameters | 4619524 | 4619524 | 4619524 | 4619524 | 4619524 |
| Non.Trainable Parameters | -- | -- | -- | -- | -- |

## VI. EXPERIMENTAL RESULTS AND DISCUSSIONS

This part goes into considerable depth about both the investigational design and the outcomes. The trial setup includes the framework and code structure training information used in the present study. We conducted separate experiments and compared the outcomes. The experiment's findings are presented and discussed in this part of the paper. We adjusted the hyper-factors for the suggested algorithm's batch size, epochs, and folds in order to get the most effective findings.

Forty (40) batches of data each epoch from a total of 100 epochs are used to learn the network. For every experiment, data is separated into 20%–80% segments for network testing and learning. The set for validation uses 16% of the training set's data. The setup makes use of the Keras platform. The parallel implementation is essential for deep learning training. As a result, we employed Kaggle and the open-source software Python 3.11.0 to perform out the classifier's learning and validation (GPU: NVIDIA TESLA P100 GPUs, 16 GB RAM). The recommended strategy was constructed using the Keras library from Tensor flow applications, and the execution duration was 560.7 seconds. Five series of trials show the changes in how well the suggested solution performs [34].

The framework's assessment establishes how effectively a certain data structure generalizes to new data in order to distinguish among multiple approaches. To do this, we need to assess the effectiveness of multiple algorithms using a method of estimation besides an evaluating approach, such as a learn-test break or cross-validation [27].

A crucial indicator is the accuracy of classification (ACC), which assesses in what way effectively the algorithm foretells a class of instances in the validation set. Further measurements include those defined by terms like sensitivity (SEN), precision, and specificity (SPE) [29][30][31]:

$$\text{Accuracy} = \frac{tn+tp}{tn+tp+fn+fp} \quad (6)$$

$$\text{Sensitivity (Recall)} = \frac{tp}{tp+fn} \quad (7)$$

$$\text{Specificity} = \frac{tn}{tn+fp} \quad (8)$$

$$\text{Precision} = \frac{tp}{tp+fn} \quad (9)$$

$$\text{F1 Score} = \frac{2*Percision*Recall}{Percision+Recall} \quad (10)$$

The symbols tp, tn, fp, and fn stand for true positive, true negative, false positive, and false negative, respectively. In order to analyze measurements that are quantitative, the confusion matrix is utilized. The confusion matrix is a table that categorizes forecasts into those that were right and those that were wrong [31][32][35]. A confusion matrix is used in Fig. 8 to show the link between the expected class and the true class. Fig. 8 displays the CNN algorithm's evaluation outcomes for the multinomial categorization of malware groups.

A figure illustrating how intelligent the model is used to identify the family of each malware is shown, and we discover that there was some overlap in identifying some malware families as a result of the limited set of grayscale images on which the model was trained. This is what happened with the family (Autorun. K), which contained a number of images used for only a few grayscale images, and this had an impact on the effectiveness of correctly recognizing the family.

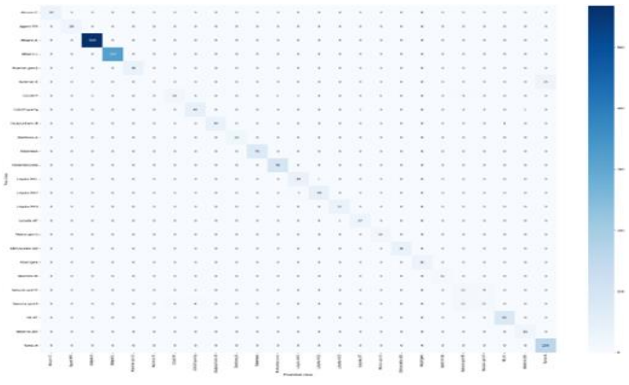

Fig. 8. Results of CNN testing for the Confusion Matrix, showing the accuracy with which it predicted each malware family shown in Fig. 1's list.

Fig. 9. Displays the confusion matrices for the suggested systems.

The produced confusion matrix may be used to create other indices, such as accuracy, precision, F1-score, specificity, and sensitivity (recall). The weighted average of recall and accuracy is the F1 score. The confusion matrix and associated metrics are typically used in conjunction to examine and evaluate categorization methods. Fig. 9 displays the confusion matrices for the suggested systems. Examining which classes, if any, are being misclassified more is quite helpful in determining this. Confusion matrices are helpful for model administration and monitoring in addition to model evaluation. Create confusion matrices for each family class to identify true negatives, false positives, and true positives.

We have employed the criteria already described before to contrast the effectiveness of our methodology. The CNN framework utilizing the basic structure, learned from the beginning via various time-running epochs with the values 20, 40, 50, and 100, achieves an overall classification accuracy of 98.31%. Fig. 10 to 13 show the comparison of accuracy, precision, recall and specificity values for the suggested systems at different epochs respectively. The stated algorithm had a precision of 97.59% as shown in Fig. 10 while Fig. 11 presents a Precision of 97.59 %. Fig. 12 demonstrates a Recall of 90.06% where Fig. 13 introduces Specificity of 99.87% and a F1 score of 99%. Table II introduces the contrasts of the results of accuracy performance by different techniques with the proposed system. According to the findings, our suggested method can provide a reliable algorithm to have an optimum performance to reduce the error and offer an overall accuracy of about 98.31 %. By enhancing the CNN model's architectural design with additional hidden layers, improved nonlinearities, and/or an optimized dropout, it may be possible to get a greater understanding of how to apply it to the categorization of malware. These insights may provide information on the architecture that will work best for creating an intelligent anti-malware system.

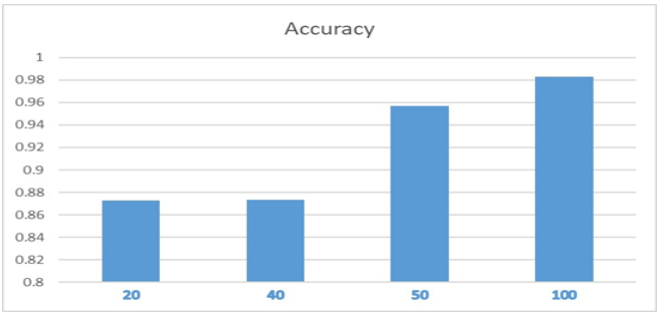


Fig. 10. Compares the accuracy values for the proposed systems at various epochs.

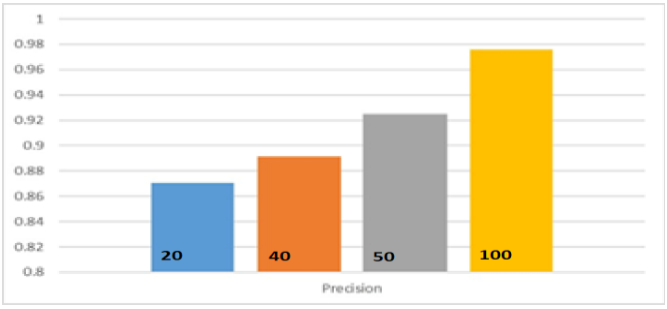


Fig. 11. Displays a contrast of the recommended systems' precision values at various epochs.

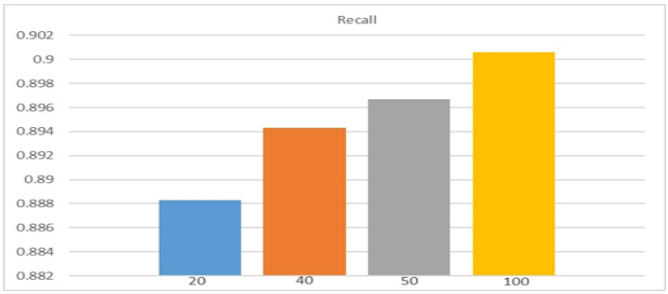


Fig. 12. Compares the recall values for the proposed systems at various epochs.

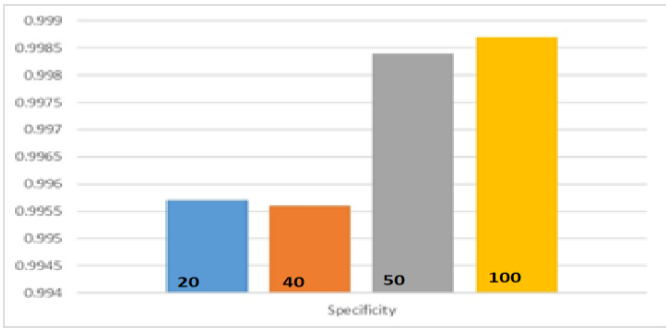


Fig. 13. Shows the comparison of specificity values for the suggested systems at different epochs.

TABLE II. CONTRASTS THE RESULTS OF ACCURACY PERFORMANCE BY DIFFERENT TECHNIQUES WITH THE PROPOSED SYSTEM

| Author | Algorithm | Accuracy | Precision | Recall | F1 Score | Specificity |
|---|---|---|---|---|---|---|
| (PRIMA 2020)[32] | **CNN** | **97%** | **91%** | **91%** | **91%** | -- |
| (PRIMA 2020)[32] | **VGG16** | **98%** | **95%** | **95%** | **95%** | -- |
| (Nataraj et al. 2011)[28] | **GIST + KNN** | **96.97%** | -- | -- | -- | -- |
| (Gibert et al. 2019)[31] | **CNN** | **97.5%** | -- | -- | **95%** | -- |
| (Yue2017) [35] | **Fine-tuning VGG19** | **97.3%** | -- | -- | -- | -- |
| (Abien 2019)[30] | **GRU-SVM** | **≈84.92%.** | **85%** | **85%** | **85%** | -- |
| (Abien 2019)[30] | **MLP-SVM** | **≈80.47%** | **83%** | **80%** | **81%** | -- |
| (Abien 2019)[30] | **CNN-SVM** | **≈77.23%** | **84%** | **77%** | **97%** | -- |
| Our Proposed System | **CNN** | **98.31%** | **97.59 %** | **90.09%** | **99%** | **99.87%** |

## VII. CONCLUSION

In the current research, we develop an advanced (DL) image classification algorithm that was previously trained on the MalImg dataset to classify malware based on images. (CNN)-based (DL) methods were contrasted with an extra simple technique created from beginning. We used the same dataset and equal image sizes for our experiments. Since there were no malware zero days in the sample, the model cannot learn and cannot accomplish its objective of identifying zero days and will be considered in the future. Based on transfer learning findings, the model has been demonstrated to be the most effective after accuracy testing. As a result, we can conclude that the transfer learning approach is suitable for classifying malware to categorize. By enhancing the CNN model architecture design with more hidden layers, improved nonlinearities, and/or an optimal dropout, it may be possible to gain more understanding of how well these models apply to the classification of malware. These findings could help in the development of an intelligent anti-malware platform by informing the type of structure to employ. The total testing accuracy of 98.31% in the reported findings attested to the excellent accuracy and robustness of the recommended technique.

## REFERENCES

[1] P. Bouchaib, and B. Mohamed, “Using Transfer Learning for Malware Classification”, in The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 2020, pp. 1-7, doi: 10.5194/isprs-archives-XLIV-4-W3-2020-343-2020.

[2] Kuo, WC., Chen, YT., Huang, YC., Wang, CC. (2023). Malware Detection Based on Image Conversion. In: Tsihrintzis, G.A., Wang, SJ., Lin, IC. (eds) 2021 International Conference on Security and Information Technologies with AI, Internet Computing and Big-data Applications. Smart Innovation, Systems and Technologies, vol 314. Springer, Cham. https://doi.org/10.1007/978-3-031-05491-4_19.

[3] Ö. ASLAN, and A. YILMAZ, “A New Malware Classification Framework Based on Deep Learning Algorithms”, in IEEE Access, vol. 9, pp. 1–16, Jun. 2021, doi: 10.1109/ACCESS.2021.

[4] Yan, H., Zhou, H., Zhang, H.: Automatic malware classification via PRICoLBP. Chin. J. Electron. 27, 852–859 (2018).

[5] Wadkar, M., Di Troia, F., Stamp, M.: Detecting malware evolution using support vector machines. Expert Syst. Appl. 143, 113022 (2020).

[6] M. Sikorski and A. Honig, “Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software” San Francisco, CA, USA: No starch press, 2012.

[7] Ö. Aslan, ``Performance comparison of static malware analysis tools versus antivirus scanners to detect malware," in Proc. Int. Multidisciplinary Stud. Congr. (IMSC), 2017, pp. 1-6.

[8] K. Pandey and B. M. Mehtre, ``Performance of malware detection tools: A comparison," in Proc. IEEE Int. Conf. Adv. Commun., Control Comput. Technol., May 2014, pp. 1811_1817.

[9] S. A. Roseline, S. Geetha, S. Kadry, and Y. Nam, ``Intelligent vision-based malware detection and classi_cation using deep random forest paradigm," IEEE Access, vol. 8, pp. 206303_206324, 2020.

[10] Ö. Aslan and R. Samet, ``Investigation of possibilities to detect malware using existing tools," in Proc. IEEE/ACS 14th Int. Conf. Comput. Syst. Appl. (AICCSA), Oct. 2017, pp. 1277_1284.

[11] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, and S. Venkatraman, ``Robust intelligent malware detection using deep learning," IEEE Access, vol. 7, pp. 46717_46738, 2019.

[12] P. Prajapati, F. Troia, and M. Stamp, “Transfer Learning for Image-Based Malware Classification”, in 3rd International Workshop on Formal Methods for Security Engineering (ForSE 2019), in conjunction with the 5th International Conference on Information Systems Security and Privacy (ICISSP 2019), 2019, pp. 1-9, doi: 10.5220/0007701407190726.

[13] C.Zhang, E. Nateghinia, L. Miranda-Moreno and L. Sun “Pavement distress detection using convolutional neural network (CNN): A case study in Montreal, Canada”, in International Journal of Transportation Science and Technology, 2021, pp. 7, doi: 10.1016/j.ijtst.2021.04.008.

[14] B. Marais, T. Quertier, and C. Chesneau “Malware Analysis with Artificial Intelligence and a Particular Attention on Results Interpretability”, in International Symposium on Distributed Computing and Artificial Intelligence. Springer, Cham, 2021, pp 1-11, doi: 10.1007/978-3-030-86261-9_5.

[15] McAfee Labs, Threats Report, June 2021 [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-threats-jun-2021.pdf.

[16] K. Simonyan, A. Zisserman, “Very deep convolutional networks for large-scale image recognition”, in 3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc, 2015, pp. 1–14.

[17] Du, D., Sun, Y., Ma, Y., Xiao, F.: A novel approach to detect malware variants based on classified behaviors. IEEE Access 7, 81770–81782 (2019).

[18] M. Huang, "Theory and Implementation of linear regression," 2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL), Chongqing, China, 2020, pp. 210-217, doi: 10.1109/CVIDL51233.2020.00-99.

[19] M. Usama et al., "Unsupervised Machine Learning for Networking: Techniques, Applications and Research Challenges," in IEEE Access, vol. 7, pp. 65579-65615, 2019, doi: 10.1109/ACCESS.2019.2916648.

[20] N. Amruthnath and T. Gupta, "A research study on unsupervised machine learning algorithms for early fault detection in predictive maintenance," 2018 5th International Conference on Industrial Engineering and Applications (ICIEA), Singapore, 2018, pp. 355-361, doi: 10.1109/IEA.2018.8387124.

[21] Kaluarachchi, T.; Reis, A.; Nanayakkara, S. A Review of Recent Deep Learning Approaches in Human-Centered Machine Learning. Sensors 2021, 21, 2514. https://doi.org/10.3390/s21072514.

[22] Antonio Hernández-Blanco, Boris Herrera-Flores, David Tomás, Borja Navarro-Colorado, "A Systematic Review of Deep Learning Approaches to Educational Data Mining", Complexity, vol. 2019, Article ID 1306039, 22 pages, 2019. https://doi.org/10.1155/2019/1306039.

[23] Marais, B., Quertier, T., Chesneau, C. (2022). Malware Analysis with Artificial Intelligence and a Particular Attention on Results Interpretability. In: Matsui, K., Omatu, S., Yigitcanlar, T., González, S.R. (eds) Distributed Computing and Artificial Intelligence, Volume 1: 18th International Conference. DCAI 2021. Lecture Notes in Networks and Systems, vol 327. Springer, Cham. https://doi.org/10.1007/978-3-030-86261-9_5.

[24] Christian Wojner. Bytehist. https://www.cert.at/en/downloads/software/software-bytehist.

[25] C.A. Ronao, S.-B. Cho, Human activity recognition with smartphone sensors using deep learning neural networks, Exp. Syst. Appl. 59 (2016) 235–244.

[26] Z. Wang, W. Yan, T. Oates, Time series classification from scratch with deep neural networks: A strong baseline, 2017 International joint conference on neural networks (IJCNN), IEEE, 2017, pp. 1578–1585.

[27] [19] S.H. Khan, M. Hayat, M. Bennamoun, F.A. Sohel, R. Togneri, Cost-sensitive learning of deep feature representations from imbalanced data, IEEE Trans. Neural Networks Learn. Syst. 29 (8) (2017) 3573–3587.

[28] Lakshmanan Nataraj, S Karthikeyan, Gregoire Jacob, and BS Manjunath. 2011. Malware images: visualization and automatic classification. In Proceedings of the 8th international symposium on visualization for cyber security. ACM, 4.

[29] Marwa EL-Geneedy, Hossam El-Din Moustafa, Fahmi Khalifa, Hatem Khater, Eman AbdElhalim, An MRI-based deep learning approach for accurate detection of Alzheimer's disease, Alexandria Engineering Journal, Volume 63, 2023, Pages 211-221, ISSN 1110-0168, https://doi.org/10.1016/j.aej.2022.07.062.

[30] Agarap, Abien Fred. "Towards building an intelligent anti-malware system: a deep learning approach using support vector machine (SVM) for malware classification." arXiv preprint arXiv:1801.00318 (2017).

[31] Gibert, Daniel, Carles Mateu, Jordi Planes, and Ramon Vicens 2019. Using Convolutional Neural Networks for Classification of Malware Represented as Images. Journal of Computer Virology and Hacking Techniques 15(1): 15–28.

[32] Prima, B. & Bouhorma, Mohammed. (2020). USING TRANSFER LEARNING FOR MALWARE CLASSIFICATION. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. XLIV-4/W3-2020. 343-349. 10.5194/isprs-archives-XLIV-4-W3-2020-343-2020.

[33] Abdullah Farid, A. Applying Artificial Intelligence Techniques for Prediction of Neurodegenerative Disorders: A Comparative Case-Study on Clinical Tests and Neuroimaging Tests with Alzheimer's Disease. Proceedings of the 2nd International Conference on Advanced Research in Applied Science and Engineering, 2020. https://doi.org/10.33422/2nd.rase.2020.03.101.

[34] W. Abdelmoez, H. Khater and N. El-shoafy, "Comparing maintainability evolution of object-oriented and aspect-oriented software product lines," 2012 8th International Conference on Informatics and Systems (INFOS), Giza, Egypt, 2012, pp. SE-53-SE-60.

[35] Yue, Songqing, 2017. Imbalanced Malware Images Classification: A CNN Based Approach. ArXiv:1708.08042 [Cs, Stat]. http://arxiv.org/abs/1708.08042.