Received 15 October 2022 | Revised 24 November 2022 | Accepted 12 December 2022
Vol. 03, No. 02, 13 pages

RESEARCH ARTICLE TUJES Tobruk University Journal of Engineering Sciences

Malicious Software-based Image Classification Via Deep Convolutional Neural Networks

Ghada Alagel[1], Khaled Elgdamsi[2]

[1, 2] Electrical and Electronic Department, Faculty of Engineering, University of Tripoli, Libya
k.elgdamsi@uot.edu.ly

Abstract. Malicious software (Malware) classification is an important factor in the security of the computer systems. On the other hand, currently utilized signature-based methods cannot provide accurate detection of zero-day attacks and the ability to group malware variants into families with similar characteristics makes possible to create mitigation strategies that work for a whole class of programs.
This paper presents two malware family classification approaches, the first system based on transfer learning, using Convolutional Neural Networks pertained on ImageNet database by adapting the last layers to malware family classification, while the second system uses the DCNN as a bottleneck feature extractor and use these features to train a multiclass classifier using traditional machine learning algorithms, namely, support vector machines (SVM), k-Nearest Neighbor (KNN) and Naïve Bayes (NB), the main benefit of this method that it does not require any disassembly or execution of the actual malware code.
The experimental results showed that the first proposed approach could effectively be used to classify malware families with an accuracy of 92.0% using AlexNet and 88.8% using GoogleNet, while SVM classifier achieving an accuracy of 88.8% with AlexNet and 86.4% with GoogleNet gave best results using the second approach.
Achieved automatic malware classification can be very valuable to anti-malware industry and security researches.

Keywords: Cybersecurity, Malware Classification, Deep Learning, Convolutional Neural Networks.

1 Introduction

With the rapid development of the internet, malware became one of the major cyber threats nowadays. Any software performing malicious actions, including information stealing, espionage, etc. can be referred to as malware. Therefore, malware protection of computer systems is one of the most important cybersecurity tasks for single users and businesses, since even a single attack can result in compromised data and sufficient losses [1]. Research efforts that strive to enhance the understanding and identification of malware programs are of therefore very valuable to the anti-malware industry. An effective system for classifying different types of malware (that finds commonality and dissimilarities among malware samples) would provide valuable insights to the matter at hand.

CNNs have proven to be highly effective for image-based classification problems [2]. In this paper, we study the effectiveness of two CNN based architectures (AlexNet and GoogleNet) both as classification tools and bottleneck feature extractors.

Classification of malware has been an area attracting great interest. Several studies on malware classification have been performed using CNN architectures. Cui et al. [3] detected code variants that are malicious after converting to grayscale images and using a simple CNN model. Kalash et al. [4] classified malware images by converting malware files into grayscale images, using two different datasets, Malimg [5] and Microsoft Malware [6]. They obtained 98.52% and 99.97% accuracies, respectively. Kolosnjaji B et al. [7] used a neural network with convolutional layers and circular layers to classify malware using system call sequences as a feature. In a dataset consisting of 4,753 malware samples, this combined neural network architecture achieved an accuracy of 85.6% and a recall of 89.4%.

The rest of the paper organized as follows: Section 2 describes the insights from representing malicious software as gray scale images. Section 3 provides a brief description about convolutional neural networks models and machine learning classifiers used in the research. Section 4 describes the methods proposed in this work in details. Section 5 evaluates the performance of our method. Finally, Section 6 concludes with our remarks and future work suggestions.

2 Malware visualization

The malware visualization method was initially proposed by Nataraj et al [5] to represent malware binary files as byteplot grayscale images and using similarity calculations between images to classify malware families, The malware binaries were used are in Portable Executable (PE) form. PE files are usually recognized through their components, which are called .text, .rdata, .data and .rsrc. The first component, called .text, is the code section, containing the program's instructions. .rdata is the part that contains read only data, and .data is the part that contains data that can be modified, and .rsrc is the final component that stands for resources used by the malware [8].

The visualization process of a PE file into a grayscale image is shown in Fig 1, each 8-bit is converted into one unsigned integer, and the result of these variables is reorganized into a 2D array. The corresponding value in the array can be expressed as the gray value of the generated image in the range of [0, 255], where 0 and 255 represent black and white respectively and other values are intermediate shades of gray. The resulting grayscale images will have fixed width and the height is allowed to vary depending on the file size [9].

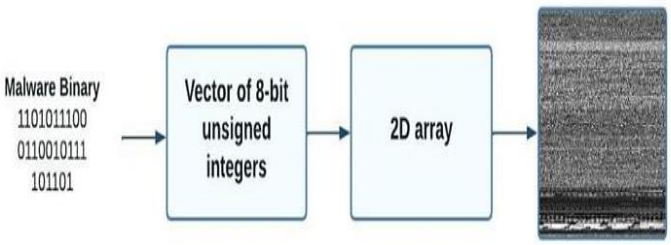

Fig. 1. Visualizing Malware as an Image

The main advantage of the representation of malware as images is certainly that significant visual similarities in image texture and overall layout for malware belonging to the same family, this due to the fact that malware authors make reuse of old malware binaries to generate new variants.

Fig. 2 shows the several sections of a common Trojan downloader, Dontovo A, which downloads and executes arbitrary files. Different sections of the malware have distinct feature patterns when seen as an image. Based on these patterns, we can classify malware [5].

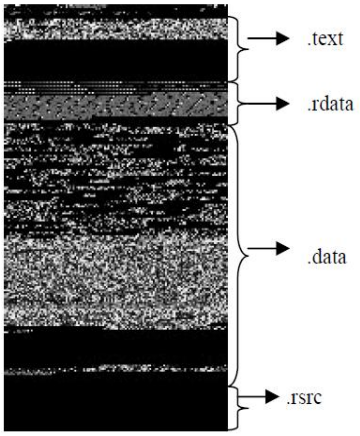

Fig. 2. Various sections of Trojan: Dontovo.A

3 Convolutional neural networks

CNN is a type of neural network that has proven to be very effective in different areas such as image recognition and classification and has been successfully applied in different fields, providing state-of-the-art performance for vision-based applications.

A typical CNN architecture generally comprises alternate layers of convolution and pooling followed by one or more fully connected layers at the end. The arrangement of CNN components plays a fundamental role in designing new architectures and thus achieving enhanced performance. This section briefly discusses the role of these components in a CNN architecture [10].

Convolution & pooling Layers

A convolution layer is a fundamental component of the CNN architecture that performs feature extraction, it takes its input from the input layer or sampling layer. A pooling layer takes each feature map output from the convolutional layer and down-samples it, i.e., pooling layer summarizes a region of neurons in the convolution layer.

The process of training a CNN model with regard to the convolution layer is to identify the kernels that work best for a given task based on a given training dataset. Kernels are the only parameters automatically learned during the training process in the convolution layer; on the other hand, the size of the kernels, number of kernels, padding, and stride are hyper parameters that need to be set before the training process starts as tabulated in Table 1.

Table 1. A list of parameters and hyper parameters in a convolutional neural network [10]

	Parameter	hyper parameters
Convolution layer	Kernels	Kernel size, number of kernels, stride, padding, activation function
Pooling layer	None	Pooling method, filter size, stride, padding
Fully connected layer	Weights	Number of weights, activation function
Others		Model architecture, optimizer, learning rate, loss function, mini-batch size, epochs, regularization, weight initialization, dataset splitting

Fully connected layer

Once the features extracted by the convolution layers and down sampled by the pooling layers are created, they are mapped by a subset of fully connected layers to the final outputs of the network, such as the probabilities for each class in classification tasks. The final fully connected layer typically has the same number of output nodes as the number of classes.

3.1 CNN Models Used

This section presents the convolutional neural network (CNN) models used in the study to classify malware images to detect malware.

AlexNet Architecture

AlexNet is considered as the first deep CNN architecture, which showed groundbreaking results for image classification and recognition tasks. AlexNet was proposed by [2] which enhanced the learning capacity of the CNN by making it deeper and by applying several parameter optimizations strategies. The basic architectural design of AlexNet shown in

Fig. 3. In AlexNet, depth was extended to 8 layers to make CNN applicable for diverse categories of images. Overlapping subsampling and local response normalization were also applied to improve the generalization by reducing overfitting. Other adjustments made were the use of large size filters (11×11 and 5×5) at the initial layers, compared to previously proposed networks [10].

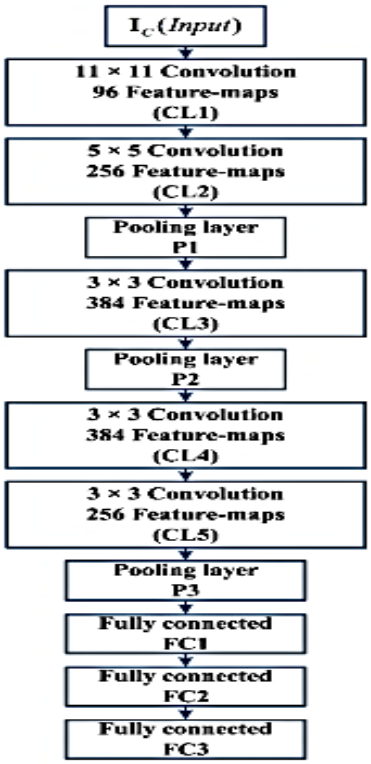

Fig. 3. Basic layout of AlexNet architecture showing its five convolution and three fully connected layers

GoogleNet Architecture

GoogleNet was the winner of the 2014-ILSVRC competition and is also known as Inception-V1. The main objective of the GoogleNet architecture was to achieve high accuracy with a reduced computational cost [11]. It introduced the new concept of inception block in CNN, whereby it incorporates multi-scale convolutional transformations using split, transform and merge idea. The architecture of the inception block is shown in Fig. 4. In GoogleNet, conventional convolutional layers are replaced in small blocks similar to the idea of substituting each layer with micro NN as proposed in Network in Network (NIN) architecture [12]. This block encapsulates filters of different sizes (1×1, 3×3, and 5×5) to capture spatial information at different scales, including both fine and coarse grain level. The exploitation of the idea of split, transform, and merge by GoogleNet, helped in addressing a problem related to the learning of diverse types of variations present in the same category of images having different resolutions. GoogleNet regulates the computations by adding a bottleneck layer of 1×1 convolutional filter, before employing large size kernels. In addition to it, it used sparse connections (not all the output feature-maps are connected to all the input feature-maps), to overcome the problem of redundant information and reduced cost by omitting feature-maps that were not relevant. Furthermore, connection's density was reduced by using global average pooling at the last layer, instead of using a fully connected layer [10].

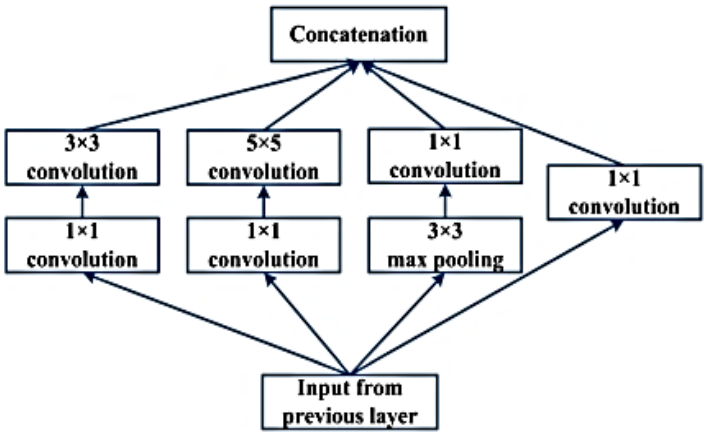

Fig. 4. Basic architecture of the inception block showing the split, transform, and merge concept

3.2 Classification methods

In the proposed method, the last layer of both Alexnet and Googlenet were replaced by another classifier and three different classifiers were evaluated for this task.

Support Vector Machines

Support Vector Machines (SVM) is one of machine learning algorithms that is generally used for classification problems. The main idea relies on finding such a hyperplane, that would separate the classes in the best way. The term 'support vectors' refers to the points lying closest to the hyperplane, that would change the hyperplane position if removed. The distance between the support vector and the hyperplane is referred to as margin. Intuitively, we understand that the further from the hyperplane our classes lie, the more accurate predictions we can make. That is why, although multiple hyperplanes can be found per problem, the goal of the SVM algorithm is to find such a hyperplane that would result in the maximum margins [13].

K-Nearest Neighbors

K-Nearest Neighbors (KNN) is one of the simplest, though, accurate machine learning algorithms. KNN is a non-parametric algorithm, meaning that it does not make any assumptions about the data structure. In real world problems, data rarely obeys the general theoretical assumptions, making non-parametric algorithms a good solution for such problems. KNN can be used for both classification and regression problems. In both problems, the prediction is based on the *k* training instances that are closest to the input instance. In the KNN classification problem, the output would be a class, to which the input instance belongs, predicted by the majority vote of the k closest neighbors [14].

Naive Bayes

Naive Bayes is the classification machine-learning algorithm that relies on the Bayes Theorem. Naive Bayes method evaluates the probability of each feature independently, regardless of any correlations, and makes the prediction based on the Bayes Theorem. That is why this method is called ”naive” – in real-world problems features often have some level of correlation between each other [15].

4 Methodology

An overview of the main steps that followed in this paper is given in Fig. 5. Two approaches were presented, the first using transfer learning by adapting the last layers of the pre-trained network to our running, while the second approach, is to use the DCNN as bottleneck features extractor. The bottleneck features are then used to train a multiclass classifier for the MFC task.

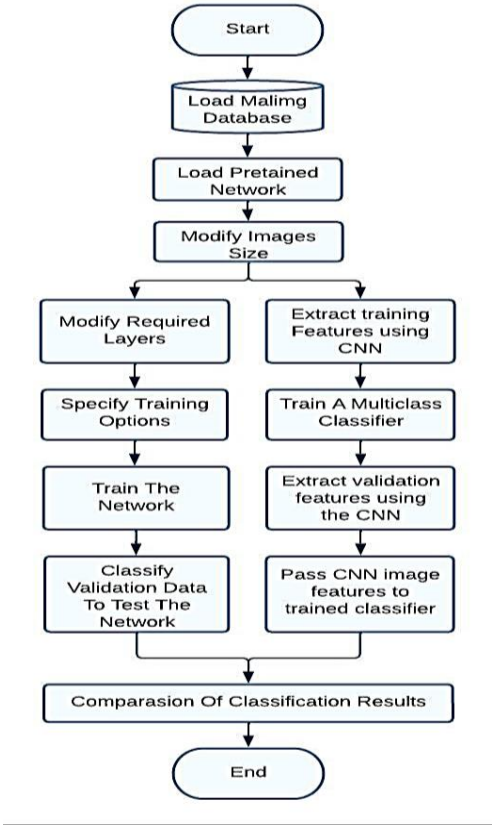

Fig. 5. Simulation Flowchart

4.1 Experimental setup

The simulation was done by using MATLAB R2018b as the development tool running on a desktop with the following hardware specifications:

- CPU: Intel i5-2520M.
- Memory: 4 GB

4.2 Dataset

There are a few malware datasets available for academic research. In this study, the publicly available (Malimg) dataset was used to train/verify the proposed learning models. The Malimg dataset was provided by [5] consists of 9435 byte plot grayscale images of malwares collected from 25 families. As reported by the authors, all byte plot images were generated from malware executables submitted to the Anubis analysis system and labels provided by Microsoft Security Essentials were used to obtain the ground truth for the dataset [16]. The images have been created from various malware families such as Dialer, Backdoor, Worm, Trojan, Trojan-Downloader, Rouge and PWS. The Malimg dataset family breakdown is given in Table 2.

Table 2. Sample distribution of malware for the Malimg dataset [5].

Family Name	Class	No. of variants
Adialer.C	Dialer	125
Agent.FYI	Backdoor	116
Allaple.A	Worm	2949
Allaple.L	Worm	1591
Alueron.gen!J	Trojan	198
Autorun.K	Worm: AutoIT	106
C2LOP.gen!g	Trojan	200
C2LOP.P	Trojan	146
Dialplatform.B	Dialer	177
Dontovo.A	Trojan Downloader	162
Fakerean	Rogue	381
Instantaccess	Dialer	431
Lolyda.AA1	PWS	213
Lolyda.AA2	PWS	184
Lolyda.AA3	PWS	123
Lolyda.AT	PWS	159
Malex.gen!J	Trojan	136
Obfuscator.AD	Trojan Downloader	142
Rbot!gen	Backdoor	158
Skintrim.N	Trojan	80
Swizzor.gen!E	Trojan Downloader	128
Swizzor.gen!I	Trojan Downloader	132
VB.AT	Worm	408
Wintrim.BX	Trojan Downloader	97
Yuner.A	Worm	800

After analyzing the number of samples of this dataset, we can see that the Malimg dataset is very unbalanced as can be seen from counts in Table 2: 31.6% of the images have a place with class: Allaple.A and 17% to class : Allaple.L! ,while the remaining 23 families only accounted for 51.4%; [9].

In this study, part of Malimg database was used as training and validation images, the database includes 1,000 malware images for 25 different families (30 image per family used as a training images and 10 image per family used as a validation images).

Note that, the byteplot grayscale image consists of a variable resolution image with only one channel, while our CNN models require constant input dimensionality with 3 channels (RGB). Therefore, the grayscale images were converted to RGB and rescaled as needed for each network. Using an augmented image data store to automatically resize the training images, and then used as input features to our CNN models.

4.3 CNN based models

In this section, CNNs pre-trained for object detection task were used as the base model for classification of malware program, and the steps that followed to accomplish the model will be illustrated.

The main steps summarized as follows:

1. Replace Required Layers

The last three layers of the pre-trained deep neural networks are configured for 1000 classes. These three layers must be fine-tuned for the new classification problem in this case 25 classes, the fine-tuning done by extracting all layers except the last three from the pre-trained network. Then transferring the layers to the new classification task by replacing the last three layers with a Fully Connected layer, a SoftMax layer and a classification output layer.

2. Specify Training Options

The training options were specified using MATLAB training Options function to control the training algorithm details.

3. Gradient Descent algorithm

To minimize the loss, these algorithms update the network parameters (weights and biases) by taking small steps in the direction of the negative gradient of the loss function. The stochastic gradient descent used in this network is 'adam' (derived from *adaptive moment estimation*).

4. Mini Batches & Epochs

The stochastic gradient descent algorithms update the parameters using a subset of the data each step. This subset is called a mini-batch, the mini batch size is 128. Each parameter update is called an iteration. A full pass through the entire data set is called an epoch, the number of epochs is 6.

5. Learning Rate

The global learning rate is set to be 1e-4, by default this value used throughout the entire training process. Increasing learning rate leads to shorten training time but may introduce more losses or poor learned network.

6. Network Training

The train/validation loss and accuracy of the CNN models for each epoch presented in section 5.

7. Classify Validation Images

The fine-tuned networks were used to classify the validation images, and the classification accuracy for each model was calculated.

4.4 Machine Learning classifiers

This approach is done by transferring convolutional layers of CNNs to our CNN model. The transferred convolutional layer's parameters are used to extract the bottleneck features. In the last step, the bottleneck features are used to train a multiclass classifier for malware family classification. This approach is equivalent to replace the fully-connected layers of the CNN by a classifier freezing the parameters of the convolutional layers during the training process, with the advantage of a much smaller training time. Finished the training process, the multiclass classifier is stacked on the top of the convolutional layers and the whole model is used to classify the validation samples.

Feature extraction using CNN

Features were extracted using the last fully connected layer present in each CNN architecture thereby 25 features from each model were extracted, as there are 25 categories of malware programs.

Classification

After feature extraction, the function fitcecoc is used to fit multiclass models for the three different classifiers were used for the classification task these are Support Vector Machine, K-Nearest Neighbor and Naïve Bayes as they have proven to be highly effective for classification of malware programs.

5 Experimental results

In this section, we present the experimental results obtained using pretrained CNNs as classification architectures for distinguishing malware families. At first, Transfer learning was employed to develop a learning model of Malimg dataset by fine-tuning the softmax layer to classify the 25-malware families, **Error! Reference source not found.** and **Error! Reference source not found.** show the training/validation progresses for AlexNet and GoogleNet respectively.

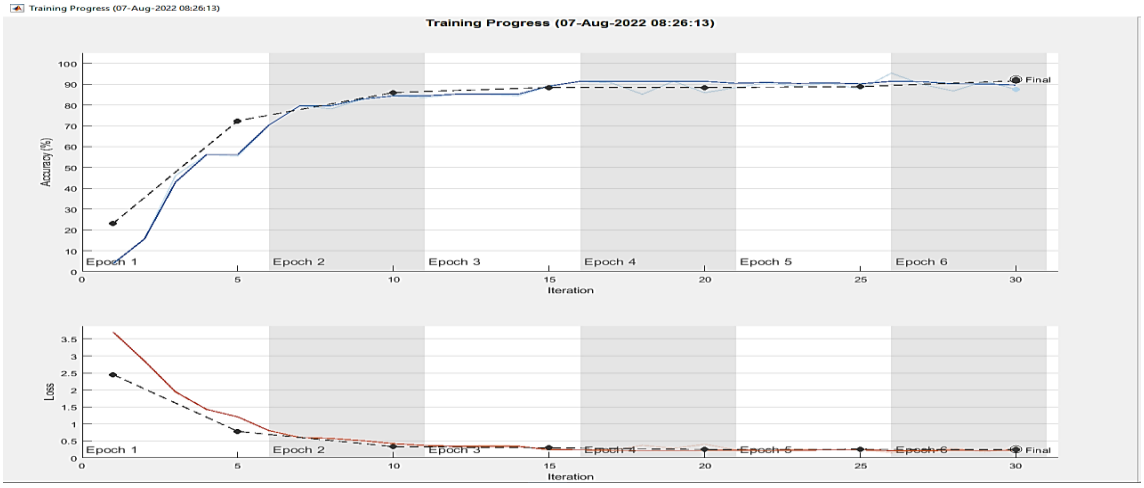

Fig. 6. Training progress of GoogleNet for MFC tas

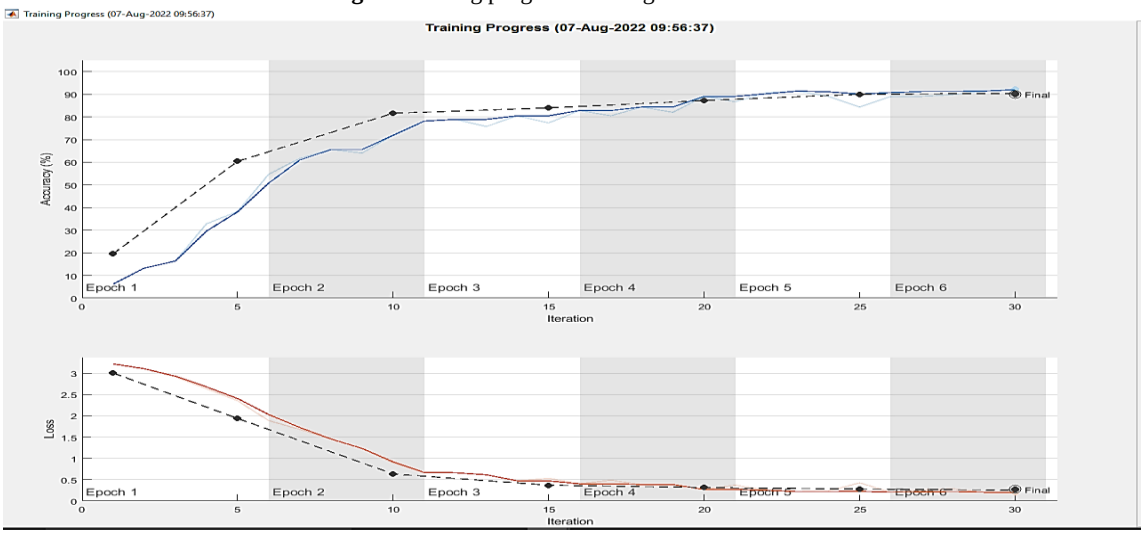

Fig. 7. Training progress of AlexNet for MFC task

In the top of **Error! Reference source not found.** which presents the accuracy of the model, it is noticed that the algorithm starts settling within the vicinity of its final value halfway at round 15 iterations, also from the training progress in **Error! Reference source not found.**, its noticeable that GoogleNet was computationally and timely expensive compared with AlexNet model, this due to the fact that GoogleNet has more layers and suffers from overfitting problem.

Fig. 8 and Fig. 9 show the confusion matrix that summarizes the performance of AlexNet and GoogleNet models resulted from simulation running process respectively.

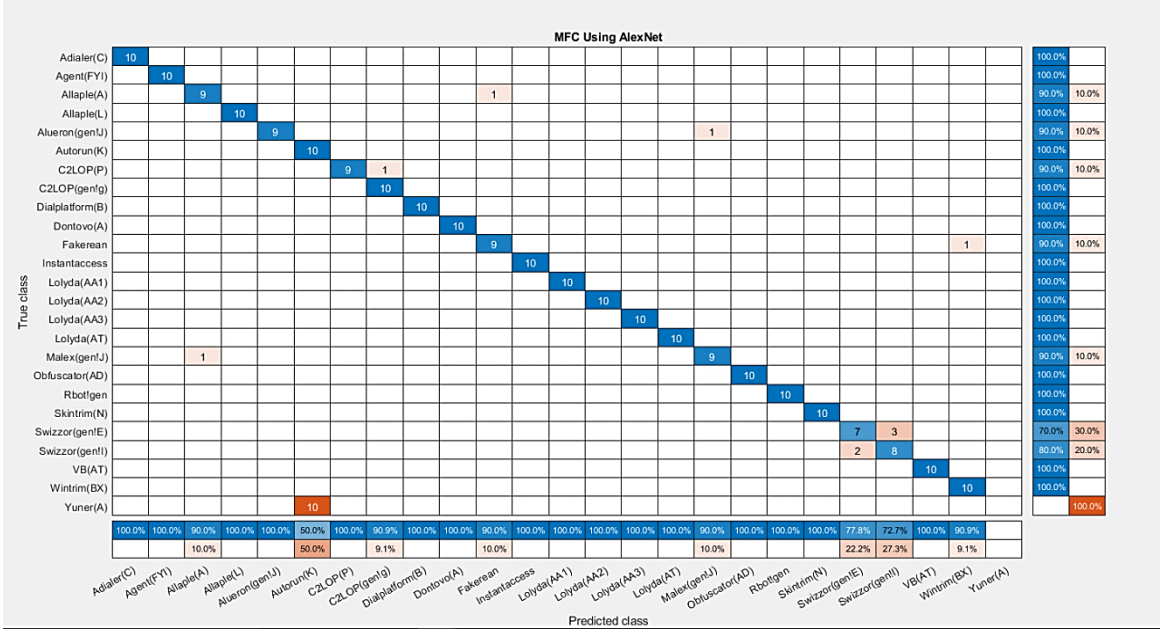

Fig. 8. Confusion matric obtained using AlexNet

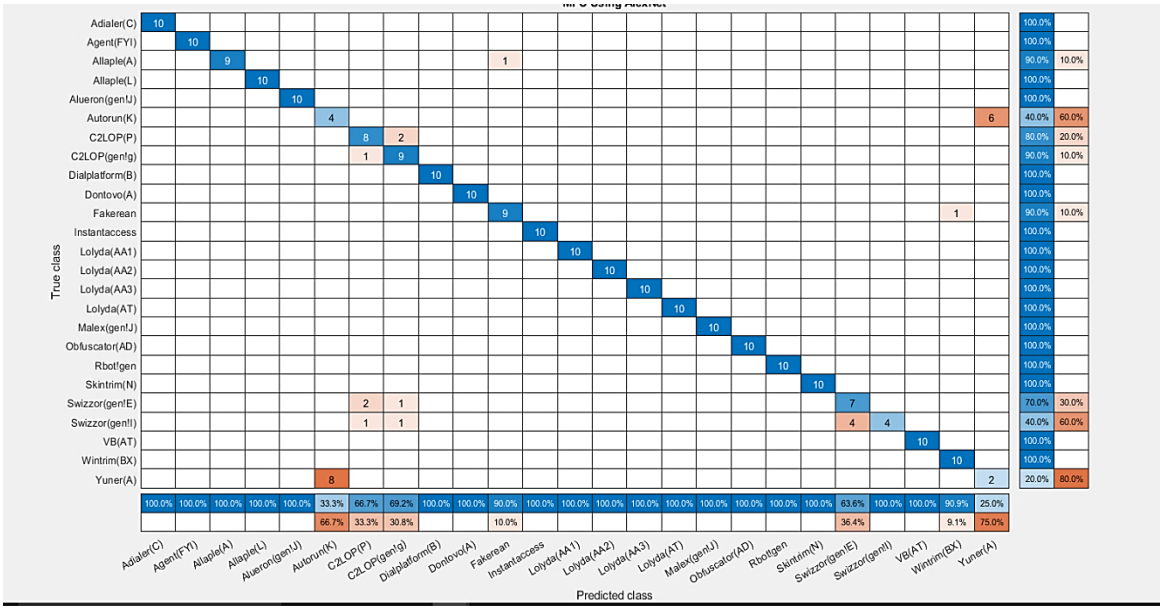

Fig. 9. Confusion matric obtained using GoogleNet

The performance measure parameters in terms of accuracy, recall, precision, and F1-score for the AlexNet and GoogleNet architectures along with their computation time for training are tabulated in Table 3.

Table 3. Performance of CNN based Architectures for malware classification

CNN	Training Time (min)	Accuracy (%)	TPR (%)	PPV (%)	F1-Score (%)
AlexNet	11	92.0	90.1	92.0	91.0
GoogleNet	704	88.8	89.5	88.8	89.1

Now, the results obtained by using the pertained CNNs architectures as bottleneck feature extractors and implementing different machine learning algorithms as a multiclass classifiers for malware classification are presented. Table 4 below compares the performance measure parameters in terms of accuracy, recall, precision, and F1-score for the models. It is obvious that the best results using pre-trained CNNs as bottleneck feature extractors and machine learning algorithms for classification were given by SVM classifier having an accuracy of 88.8% with AlexNet and 86.4% with GoogleNet, Naïve Bayes classifier shows the lowest performance having an accuracy of 52.8% using AlexNet and 58.8% using GoogleNet.

Table 4. Results obtained by various pretrained models with different classifiers

Feature Extractor	Classifier	Accuracy (%)	TPR (%)	PPV (%)	F1-Score (%)
AlexNet	SVM	88.8	88.8	88.8	88.8
	KNN	81.6	81.6	81.6	81.6
	Naïve Bayes	52.8	58.5	52.8	55.5
GoogleNet	SVM	86.4	86.2	86.4	86.3
	KNN	80.0	79.9	80.0	80.0
	Naïve Bayes	58.8	62.4	58.8	60.5

6 Conclusion

These days many antivirus programs rely on deep learning techniques to protect devices from malware. Deep learning architectures have achieved good performance in detecting and classifying malware when used with Windows PE binaries. In this paper, a malware classification mechanism using byteplot malware images and deep learning techniques were presented. a wide variety of experiments were carried out, each representing a different learning technique making use of a pretrained networks which have been trained on a large dataset and tune it to distinguish different malware families. The results were particularly impressive, with high accuracy attained over a large number of malware families which confirm that visual malware similarities can be used for accurate malware classification. Based on the practical results described before, it is recommended to implement the classification based on the AlexNet model, as it resulted in the best accuracy and high performance.

Future work could be focused on conducting results using additional models from leaderboards of image classification competitions, extending the database to coverage more malware families, distinguishing malware from benign samples and use a more powerful hardware equipment that will accelerate the processing time of our models. In addition, the transformation of malicious code into color images would be a good topic for future research.

Conflict of Interest

This is to certify that all authors have seen and approved the manuscript being submitted and they declare no competing interest.

References

1. K. Chumachenko, “Machine Learning Methods for Malware Detection and Classification,” *Proc. 21st Pan-Hellenic Conf. Informatics - PCI 2017*, p. 93, 2017.
2. A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *25th International Conference on Neural Information Processing Systems*, vol. 60, no. 6, pp. 1097–1105, 2012.
3. Z. Cui, F. Xue, X. Cai, Y. Cao, G. G. Wang, and J. Chen, “Detection of Malicious Code Variants

Based on Deep Learning," *IEEE Trans. Ind. Informatics*, vol. 14, no. 7, pp. 3187–3196, 2018.
4. M. Kalash, M. Rochan, N. Mohammed, N. D. B. Bruce, Y. Wang, and F. Iqbal, "Malware Classification with Deep Convolutional Neural Networks," *2018 9th IFIP Int. Conf. New Technol. Mobil. Secur. NTMS 2018 - Proc.*, vol. 2018-Janua, pp. 1–5, 2018.
5. L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," *ACM Int. Conf. Proceeding Ser.*, no. July, 2011.
6. R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft Malware Classification Challenge," *CODASPY 2016 - Proc. 6th ACM Conf. Data Appl. Secur. Priv.*, pp. 183–194, 2016.
7. Bojan Kolosnjaji, Apostolis Zarras, George Webster and C. E., "Deep Learning for Classification of Malware System Call Sequences," *AI 2016 Adv. Artif. Intell. AI 2016. Lect. Notes Comput. Sci.*, vol. 9992, pp. 403–415, 2016.
8. A. Bensaoud, N. Abudawaood, and J. Kalita, "Classifying Malware Images with Convolutional Neural Network Models," 2020.
9. D. Gibert, C. Mateu, J. Planes, and R. Vicens, "Using convolutional neural networks for classification of malware represented as images," *J. Comput. Virol. Hacking Tech.*, vol. 15, no. 1, pp. 15–28, 2019.
10. A. Patil and M. Rane, "Convolutional Neural Networks: An Overview and Its Applications in Pattern Recognition," *Smart Innov. Syst. Technol.*, vol. 195, pp. 21–30, 2021.
11. C. Szegedy *et al.*, "Going Deeper with Convolutions," in *IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 1–9, 2015.
12. Takuya Yoshioka , Nobutaka Ito , Marc Delcroix , Atsunori Ogawa , Keisuke Kinoshita , Masakiyo Fujimoto, "THE NTT CHIME-3 SYSTEM : ADVANCES IN SPEECH ENHANCEMENT AND RECOGNITION FOR MOBILE MULTI-MICROPHONE DEVICES" NTT Communication Science Laboratories , NT," pp. 436–443, 2015.
13. R. Jing and Y. Zhang, "A view of support vector machines algorithm on classification problems," *Proc. - 2010 Int. Conf. Multimed. Commun. Mediacom 2010*, pp. 13–16, 2010.
14. J. Laaksonen and E. Oja, "Classification with learning k-nearest neighbors," *IEEE Int. Conf. Neural Networks - Conf. Proc.*, vol. 3, pp. 1480–1483, 1996.
15. Bharadwaj, K. B. Prakash, and G. R. Kanagachidambaresan, *Pattern Recognition and Machine Learning*. 2021.
16. E. Rezende, G. Ruppert, T. Carvalho, F. Ramos, and P. De Geus, "Malicious software classification using transfer learning of ResNet-50 deep neural network," *Proc. - 16th IEEE Int. Conf. Mach. Learn. Appl. ICMLA 2017*, vol. 2017-Decem, pp. 1011–1014, 2017.