I. J. Computer Network and Information Security, **2022, 2, 14-26**
Published Online April 2022 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2022.02.02

Improved Deep Learning Model for Static PE Files Malware Detection and Classification

Sumit S. Lad.
Dept. of CSE, Rajarambapu Institute of Technology, Rajaramnagar, Sangli, Maharashtra, India.
E-mail: sumitx86@gmail.com

Amol C. Adamuthe
Dept. of CS & IT, Rajarambapu Institute of Technology, Rajaramnagar, Sangli, Maharashtra, India.
E-mail: amol.admuthe@gmail.com

Received: 05 June 2021; Revised: 08 September 2021; Accepted: 13 October 2021; Published: 08 April 2022

Abstract: Static analysis and detection of malware is a crucial phase for handling security threats. Most researchers stated that the problem with the static analysis is an imbalance in the dataset, causing invalid result metrics. It requires more time for extracting features from the raw binaries, and methods like neural networks require more time for the training. Considering these problems, we proposed a model capable of building a feature set from the dataset and classifying static PE files efficiently. The research work was conducted to emphasize the importance of feature extraction rather than focusing on model building. The well-extracted features help to provide better results when fed to neural networks with minimal numbers of layers. Using minimum layers will enhance the performance of the model and take fewer resources and time for the processing and evaluation. In this research work, EMBER datasets published by Endgame Inc. containing PE file information are used. Feature extraction, data standardization, and data cleaning techniques are performed to handle the imbalance and impurities from the dataset. Later the extracted features were scaled into a standard form to avoid the problems related to range variations. A total of 2381 features are extracted and pre-processed from both the 2017 and 2018 datasets, respectively.

The pre-processed data is then given to a deep learning model for training. The deep learning model created using dense and dropout layers to minimize the resource strain on the model and deliver more accurate results in less amount of time. The results obtained during experimentation for EMBER v2017 and v2018 datasets are 97.53% and 94.09%, respectively. The model is trained for ten epochs with a learning rate of 0.01, and it took 4 minutes/epoch, which is one minute lesser than the Decision Tree model. In terms of precision metrics, our model achieved 98.85%, which is 1.85% more as compared to the existing models.

Index Terms: Static malware analysis, Deep learning, Static PE files classification.

1. Introduction

Malware refers to pieces of code, scripts, and software designed to gain unauthorized access or harm the computer system. Malware attacks are hazardous to businesses and consumers. The rate of malware attacks is increasing day by day. According to AV-Test Institute, every day, they receive over 350,000 samples of Potentially Unwanted Applications (PUA) and malware [1]. Techniques like code obfuscation and inclusion or modification of existing malware behaviour are used to improve the damage and stealth. Anti-malware companies design Signature-based approaches, but that is not working precisely whenever new malware is detected [2]. Various anti-malware product organizations recognized these flaws in signature-based approaches [3]. Deep and machine learning nowadays has become famous for solving the problem related to the detection and classification of malware variants and the results obtained by various researchers proves there is large scope for the improvement this is the main reason behind working on this dataset. Still, the problem arises when the dataset or set of features is large, which takes more time for the execution of the model.

For creating a model for malware detection features, the set needs to be identified first. This can be done with the help of monitoring the execution process activities. The features of malicious files or programs are extracted and then given to the machine learning model building. There are two types of analysis static and dynamic [4]. Static analysis is performed by just observing the malicious scripts and code. Dynamic analysis requires an isolated environment in which the scripts are executed. Some malware is designed in such a way that they can alter them behavior when they recognize they are being analyzed [5,6,7] The reason behind working on Static PE files is static analysis techniques are applied at the initial stages, without executing the malicious codes or scripts. Considering this as an early phase for detecting and getting solutions for newly arriving malware.

In this research work, EMBER dataset versions 2017 and 2018 were used, which contains PE file information and extracted features from the dataset. Deep learning neural networks are considered as an evolutionary technique for learning from features present in images [8], signals [9], and text [10] problems. Studying this success of deep learning trained a model and tried to achieve the balance between accuracy and the number of resources required for execution. The proposed model classifies the PE files based on the features and behavior of two classes, malicious or benign.

The main objective of this research work is to make a model that can perform static analysis of malicious PE files and classify them accurately within minimum time, using fewer resources and capable of handling larger datasets. Instead of taking random PE files samples and spending time in extracting their feature, we used the EMBER datasets, which contain PE files information and creating deep learning, which will work on all the features, a smaller number of resources, and training time.

Our contributions in this research are as follows:

- We introduced a technique that performs the static analysis of malware by extracting the features from the PE files, which will address the issues related to building a feature set by extracting features for the analysis from PE files.
- For building a feature set, we applied data pre-processing techniques on extracted features for addressing the issues related to data or feature set imbalance.
- Proposed a deep learning model by using Dense and Dropout layers for classification of PE files, which addressed issues related to extensive computational resources and time required for analysis.
- Experimental results demonstrate that our proposed approach was effective in detection and efficient concerning time and resources.

This paper is divided into five sections. Section 1 is an introduction to malware analysis. Section 2 is about the literature review of static analysis of malware and deep learning. Section 3 presents the proposed methodology. Section 4 presents the experimental details, results, and discussion. Section 5 presents the conclusion.

2. Literature Review

The rate of malware produced is rising every year, and that is the main reason for considering research in the malware detection domain. Activities performed over the internet are not regarded as safe because of these malware threats. From 1995 several researchers worked in this area, but deep, and machine learning techniques for malware detection and classification are making evolutions. Malware analysis can be done using traditional Static and dynamic analysis [4]. In the static analysis, the samples of malware codes and scripts are analyzed for their work, behavior, and impact. In contrast, dynamic analysis involves the code analysis and the system’s execution into an isolated environment so that the analysis of its impact can be analyzed. Static analysis methods identify the malware before the execution, which is more essential, and they can determine whether the Program executable files are malicious or not.

Malware detection based on signatures and patterns is an essential method present in static analysis. The advantage of using static analysis is that it takes a small amount of time for execution and causes less overhead on kernels. In 2001, Schultz et al. [11] used several data mining techniques that automatically search for the patterns present in datasets for the classification of malicious and benign files. The researcher used only three different static features for performing the classification and gained higher accuracy but with a limited number of features. Researchers Kolter and Maloof [12] continued the same work and improved results with overlapping byte-sequences of static PE files. Titan et al. [13] used printable string information present in executable fines because obfuscated files will not contain words or sentences. These are some researches that used different techniques for classification. However, over a while, researchers started using features present in Static PE files. McDermott et al. [14] used the frequency of functions present in any PE file for classification, but the results achieved are not good enough. Later, researchers used machine learning and deep learning, which are particularly good for classification. Sisay Tumsa [61] applied ANN to for the detection malicious codes hidden in document files and proven that by applying ANN achieved good classification results.

Rabia Tahir [63] described different techniques for malware detection that helped in our research to understand the working of new and existing systems. Researchers applied different techniques combined with neural network models. Huang et al. [15] used deep learning techniques to detect binary encoded samples of malware directly where the results were achieved by retraining the model. Nataraj et al. [16] used malware binaries and converted them into grayscale images of malware and transformed a malware classification problem into an image classification problem. This method is faster, but attackers can use obfuscation methods to penetrate the security measures. Researcher Philippe Owezarski [17] proposed a method for anomaly detection, classification, and characterization based on an unsupervised network. Research works [18,19,20] proposed an approach that identifies malware from the API calls present in the PE files. Xabier Ugarte-Pedrero et al. [21] used executables PE sections entropy for classification. In 2015 Joshua Saxe and Konstantin Berlin [22] proposed an approach that uses input features histogram of byte entropy and NN for classification. In 2017 researcher Edward Raff et al. [23] used RNN and FC-based models for the classification of malware. Nguyen Van Nhuong et al. [24] proposed a method for the detection of metamorphic malware based on semantic set methods in 2015. Vu Thanh Nguyen et al. [25] proposed a method by combining the Negative selection algorithm and AI network. Alile S.O, Egwali A.O [62] used Bayesian Belief Network Model for detecting attacks performed by using malicious IP addresses and applied this technique on data which is collected from cyber security repositories. These research works proven that deep learning techniques are helpful in the domain of malicious activities detection and classification. Referring to the research work performed by researchers Nath and Mhetre [26] following table will contain some of the well-known methods used by researchers to perform static analysis.

Table 1. Summary of well-known existing models

Author	Objective	Extraction method	Extraction source	Outcome
Kolter and Maloof [12, 27, 28, 29]	Learning and classifying malicious executables	Overlapping byte sequences	N-grams, Bytes	Using only 1651 samples achieved satisfactory results.
Dube et al. [30,31,32,33,34]	Classification of malicious executables using Machine learning and heuristic	Decision Tree and Static heuristic features	N-grams, Bytes	Works well with fewer samples, Decrease performance with more samples.
Zhang et al. [35]	Malware detection using ensemble learning	A probabilistic neural network, multi-classifier system	N-grams, Bytes based entropy	Individual classifiers used for collecting evidence and later combined by the combination rules Dempster-Shafer
Lynda et al. [36]	Finding encrypted and packed malware for analysis	Binary entropy analysis	Packed and Encrypted PE files	Analyzed 21576 PE files and useful for files with less than 500KB
Santos at el. [37,38,39]	Unknown Malware detection using Opcode sequencing	Opcodes sequences, data mining, and semi-supervised learning	Assembly language Opcodes	Work well on labeled data present in a small amount for each class.
Bilar [40]	Detection of malicious executables using statistical data	Frequency distribution	Opcodes	Identified 14 opcodes mostly present in malicious files.
Schultz at el. [11]	Detection of malicious executables using data mining	Data mining framework	PE files, DLL files, and Function calls	4,266 programs analyzed. Out of this, 38 malicious and 206 clean PE files were analyzed
Shafiq at el. [41,42,43]	PE files classification	Multi-Layer perceptron	PE files header	They failed as compared to the N-gram technique. Cannot be able to exploit header information.

The Table 1 presents research works are conducted with minimum number of samples, fixed size of the samples or by using less numbers of features than the actual dataset contains that defines the existing research not tested on entire datasets or features, deep learning models take much more resources and time. To tackle these identified research gaps from the above literatures and by taking inspiration from them, we proposed a deep learning-based model for the classification of malware PE files based on the EMBER dataset released by Endgame Inc. for the years 2017 and 2018. At the initial stage to reduce the strain on the neural network, we performed feature extraction, data cleaning, and scaling into a standard form. To make the model less resource-intensive, we used only dropout and dense layers for the model. By combining these techniques, we achieved excellent results with minimal training time for both the 2017 and 2018 datasets. The reason behind selecting and trying to achieve our objective with the help of this dataset is:

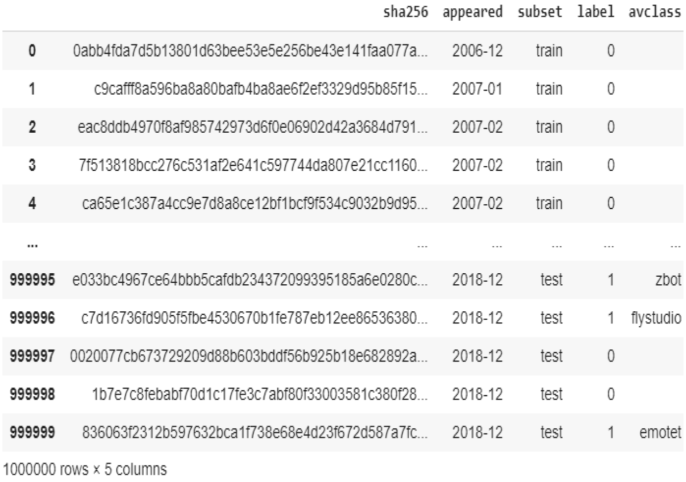
- The number of samples in both datasets is large, which will check our model's capability to handle a larger dataset.
- Focusing on the extraction of features, and data cleansing before feeding data to the deep learning model.
- The number of features is more than the existing as well as earlier research works, to identify that with a greater number of features how well the model works, and there is future scope to perform feature selection and check for the improvements in the results.
- Feeding well extracted and cleaned data to deep learning model to make the model faster, less resource-intensive, and perform results evaluation efficiently.

3. Proposed Methodology

3.1. Feature Extraction

It is the process of extracting a set of features from the given dataset. The feature extraction will reduce the feature space of the dataset. Reducing the feature space will avoid unnecessary processing of the whole dataset while training and reduces the computation overhead [44]. EMBER dataset consisting of PE files of the JSON format. The extraction of features from these JSON files is essential for training the deep learning model. Endgame Inc. suggested the LIEF (Library to Instrument Executable Formats) library, which is widely used for extracting features from PE files [45].

Using these extracted features present in JSON files vectorizing is performed to obtain binary format features that can be later stored into CSV or any other data formats. During our research work, LIEF 0.9.0 is used for both the 2017 and 2018 datasets. The dataset is already divided into Training and Testing sets. Table 1 and Table 2 present the distribution of dataset samples. Both datasets have 2381 features, which we are going to feed to our deep neural network.



	sha256	appeared	subset	label	avclass
0	0abb4fda7d5b13801d63bee53e5e256be43e141faa077a...	2006-12	train	0	
1	c9cafff8a596ba8a80bafb4ba8ae6f2ef3329d95b85f15...	2007-01	train	0	
2	eac8ddb4970f8af985742973d6f0e06902d42a3684d791...	2007-02	train	0	
3	7f513818bcc276c531af2e641c597744da807e21cc1160...	2007-02	train	0	
4	ca65e1c387a4cc9e7d8a8ce12bf1bcf9f534c9032b9d95...	2007-02	train	0	
...	...	...	...	...	...
999995	e033bc4967ce64bbb5cafdb234372099395185a6e0280c...	2018-12	test	1	zbot
999996	c7d16736fd905f5fbe4530670b1fe787eb12ee86536380...	2018-12	test	1	flystudio
999997	0020077cb673729209d88b603bddf56b925b18e682892a...	2018-12	test	0	
999998	1b7e7c8febabf70d1c17fe3c7abf80f33003581c380f28...	2018-12	test	0	
999999	836063f2312b597632bca1f738e68e4d23f672d587a7fc...	2018-12	test	1	emotet

1000000 rows × 5 columns

Fig.1. Vectorizing Extracted features from the dataset.

3.2. Data Cleaning

Data cleaning is also known as data scrubbing or cleansing. This is used to recognize and remove the inconsistencies and errors present in the datasets, which improves the quality of the datasets. The EMBER dataset is published by Endgame Inc. to create this dataset. According to researchers [46], several resources might be used and showed how the data collected from single or multiple resources are required to perform the cleaning of data. The EMBER datasets consist of training and testing sets. In datasets, training sets the data is present in 3 distinct categories: Benign, Malicious, and Unlabeled. These categories are defined in the dataset as -1, 0, and 1, respectively. But testing sets present in the dataset do not contain Unlabeled data. To balance this, we ignored unlabeled samples from further processing and reconstructed the training set with only labeled data. This will equalize both training and testing sets and improve the performance of the deep learning model.

3.3. Data Standardization

The values of different measures on which the measurements are made need to be appropriately scaled. Otherwise, the dissimilarities in different measures will affect the model. The idea behind the standardization is to make the variables present in the dataset comparable with each other [47]. For better performance, we scaled numerical values into a standard range. If features of the dataset are not appropriately scaled, feature variance in which one feature dominates over other features. To avoid this problem, we performed data standardization. To perform this, we used sklearn. preprocessing. Standard Scaler [48] in which the standard score of x is calculated as:

$$z = \frac{(z-u)}{s} \tag{1}$$

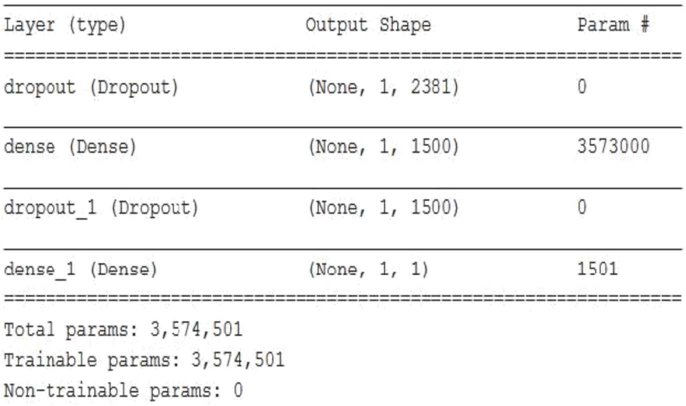
Were,
u= Mean of training samples,
s= Standard deviation of training samples.

This scaled data is then given to the neural network model for further training.

3.4. Proposed Model

For creating this model, we used the Keras library. Before creating the model, we referred to several researchers' works and books. Deep Learning with Python by François Chollet book explains the usage of dropout and dense layers of the proper usage and building efficient models with it. Taking inspiration from it, we created a simple model using these two layers. Dropout is mostly considered for neural networks better regularization. We already scaled the input to the standard form, so adding additional layers will only consume computing resources and affect training time. The dropout layer randomly removes different sets of neurons, which means we force the neural network to learn from less amount of data [49]. By doing this prevents conspiracies and reduces overfitting [50]. The reason behind using dense layers over convolutional layers is dense layers take all inputs from the previous layer. In contrast, convolution layers take only repetitive features [51]. These are the most common ways to create a simple and less prone to overfitting model we applied during our experimentation and achieved excellent results:

- A right amount of training data; in the EMBER dataset, around 600K training samples are present even after removing unlabeled samples.
- Dataset standardized before giving to the neural network, which reduces training time.
- Reduce the neural network’s capacity by proper usage of dropout and dense layers to keep the model simple, less resource-intensive, and less effective to overfitting.



```
Layer (type)                 Output Shape              Param #
=================================================================
dropout (Dropout)            (None, 1, 2381)           0
_________________________________________________________________
dense (Dense)                (None, 1, 1500)           3573000
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 1500)           0
_________________________________________________________________
dense_1 (Dense)              (None, 1, 1)              1501
=================================================================
Total params: 3,574,501
Trainable params: 3,574,501
Non-trainable params: 0
```

Fig.2. Proposed model details.

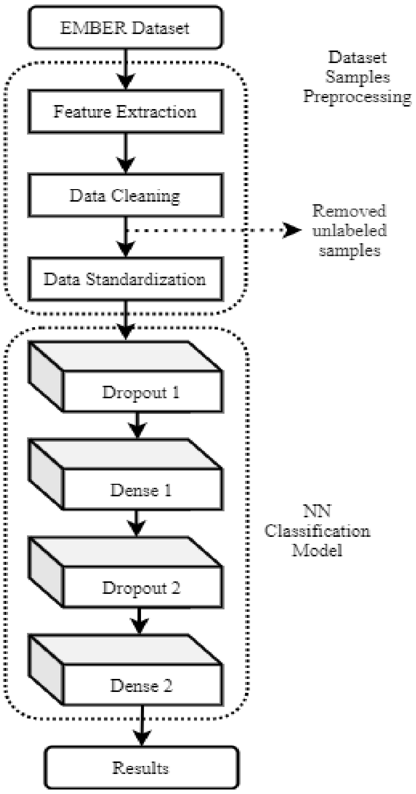


Fig.3. The workflow of the proposed architecture

Fig. 3 given below presents the overall system architecture of the proposed methodology. The system architecture is made up of combining several distinct techniques. As discussed earlier, EMBER datasets contain three diverse types of samples. Each sample present in the datasets has its information. Using this information and extracting it as a feature is performed before it is fed to the deep learning model. Table 5 presents that some researchers selected or reduced the number of features used for training to obtain better results, but we used all the 2381 features. Understanding the usage of each type of sample and using them accordingly for training is essential; that’s why data cleansing is performed so that unlabeled samples are eliminated from the model training. Fig. 3 shows that only malicious and benign samples are considered for training. Organizing these extracted and cleaned features into the standard form is necessary because while training the model, each sample will get processed equally as others. The StandardScalar library is used to perform this task. After performing these three separate techniques on the dataset, the samples are given to the training model, as shown in Fig. 2. This is how we utilized these different functionalities to work together and form a malware detection model.

The proposed model works on static PE files malware detection and classification of them into Malicious or Benign. With the proposed model, we tried to minimize the gap between finding the right static PE files and building the features set. The proposed model is a combination of several individual modules. By using these modules, we proposed this architecture, which not only performs the classification but can extract features, clean the extracted features by removing impurities and redundancies as well as maintain the extracted features into a standard form, which can be given to the neural network for further classification. This is how combining individual modules and utilizing them became the strength of our research work.

In this proposed research work, each method is contributed their part to achieve the objectives and results in such way:

Feature Extraction: Instead of using any random method, we studied the dataset and found the LIEF library is recommended by Endgame Inc. Which helped us to extract the 2381 features from the dataset, which the earlier researchers failed to do. Even with a greater number of features, our model performed well in terms of performance.

Data Cleaning: Before working on all samples present in the dataset, observed the training and testing samples identified unlabeled samples from the dataset. Our classification model will only work on classifying malware or benign files, there is no use in taking these unlabeled samples to increase processing overhead for the feature extraction as well as for the deep learning model. Which will have helped us to handle the imbalance in the dataset by removing unnecessary samples.

Data Standardization: All the samples present in the dataset do not contain uniform values and ranges of values vary. To avoid this problem, this technique is used which helps to standardize the range of dataset samples, which is important for the evaluation of the deep learning model.

Deep learning model: In our research, the data provided to the model is finely organized and extracted, so there is no need to add multiple layers and make excessive processing. Our model will only pass the samples through the model and act as a classifier which only classifies the samples without taking much processing effort. This will be helped to make the model less resource as well as time-intensive.

In this way the proposed methodologies helped us to achieve our objectives during the entire research work, the results we achieved during this research work are significantly better than the models which calculated their results with a smaller number of features as well as by applying very well-known resource and time-intensive methods.

4. Experimental Details and Results

For the experiment, two different versions of the EMBER dataset 2017 and 2018 are used. Both datasets are passed through proposed methodologies. The experiments conducted are explained with the help of series of in general steps as follows:

- The Extraction of features is done on both datasets, 2381 features are extracted from the datasets
- The dataset is cleaned by removing unlabeled samples which are unhelpful for this classification problem.
- Data standardization is applied to the features to range the values to process all the values similarly.
- The well-extracted and standardized data samples are now given to the deep learning model for the further task of classifying samples into two classes malicious or benign.

4.1. Dataset

Endgame Malware BEnchmark for Research (EMBER) published by Endgame Inc. works in the cybersecurity domain, such as detection, mitigation, and exploitation of security threats. The dataset [52] contains extracted features of malicious Windows programs executable binary files. These features were extracted from 1.1 million binaries. Table 2 presents the samples present in EMBER v2017 dataset.

Table 2. EMBER 2017 Dataset samples distribution [52]

Training Samples			**Testing Samples**	
Malicious	**Benign**	**Unlabeled**	**Malicious**	**Benign**
300 K	300 K	300 K	100 K	100 K
Total training samples 900 K			Total testing samples 200 K	

Version 2018 of the EMBER dataset was created by extracting 1 million binary files of windows executables. Table 3 presents the samples present in EMBER v2018 dataset.

Table 3. EMBER 2018 Dataset samples distribution [52]

Training Samples			**Testing Samples**	
Malicious	**Benign**	**Unlabeled**	**Malicious**	**Benign**
300 K	300 K	200 K	100 K	100 K
Total training samples 800 K			Total testing samples 200 K	

These datasets are available in the form of JSON (JavaScript Object Notation) files. Each line on these JSON files contains a single JSON object. These objects may contain:

- SHA256- The hash value of the original PE file as a unique identifier.
- Year of PE file when it was created.
- A label which 1,0, -1 for malicious, benign, and unlabeled, respectively.
- The subset, which indicates whether the file is from the Training set or Testing set.
- AVClass is present in the EMBER v2018 dataset, which shows a malicious class of PE files.

As per Endgame Inc. [52], the dataset consists of raw features of PE files that are readable for humans. They provided the code to produce numeric features from the raw features, which will help build models. The same code is used during the experimentation for the feature extraction process. Python library lief version 0.9.0 is used for feature computation. The experimental analysis is carried out by using the Python programming language for the implementation. Keras library is used to implement the deep learning model. For executing the programs, Nvidia GeForce 940M 2GB GPU for training and Intel Core i5-4500 processor with 8 GB RAM is used. For resource-intensive tests, the Google Colab platform is used.

4.2. Result and Discussion

During this research for performance evaluation of the network following evaluation, metrics are considered. Researchers most commonly use these measures for assessing performance [53,54,55].

Accuracy: The ratio of true positive samples of all classes to the sum of all samples is present in the confusion matrix [53].

$$\text{Accuracy}=\text{TP}+\text{TNTP}+\text{TN}+\text{FP}+\text{FN} \tag{2}$$

Precision: The ratio of true positive samples of the class to the sum of true positive and false positive samples of the predictions [53].

$$\text{Precision}=\text{TPTP}+\text{FP} \tag{3}$$

F1_Score It is the average weighted values of Precision and recall [53].

$$\text{F1 Score}=2*\text{Precision}*\text{RecallPrecision}+\text{Recall} \tag{4}$$

Recall It is the ratio of labeled data samples to the correctly predicted samples [53].

$$\text{Recall}=\text{TPTP}+\text{FN} \tag{5}$$

$$\text{TPR}=\text{TPTP}+\text{FN} \tag{6}$$

$$\text{FPR}=\text{FPTP}+\text{FP} \tag{7}$$

Here is the list of parameters used during experimentation, which delivered excellent results. The model is trained for 10 epochs with a batch size of 256. Dataset split into training and testing as 600K samples for training and remaining 200K samples for the validation set. The same parameters and models are used for both datasets v2017 and v2018.

Table 4 presents the parameters used for training the model, which helped us during experimentation for obtaining satisfactory results.

Table 4. Hyperparameters used for training.

Feature's size	2381
Dropout 1	0.2 (Activation: ReLU)
Dense 1	1500
Dropout 2	0.5 (Activation: Sigmoid)
Dense 2	1
Learning Rate	0.001
Optimizer	Adam
Loss	Binary_crossentropy
Batch Size	256
Epochs	10
Split Ration (%)	80-20

Table 5 presents the results of both EMBER 2017 and 2018 datasets after training for 10 epochs. The model takes less than 4 minutes for each epoch and 40 minutes for the whole 10 epochs. During this experimentation, we achieved the best results with the same model on the EMBER 2018 dataset. However, due to less research performed on this dataset, we are considering 2017 datasets only.

Table 5. Results of the proposed model on EMBER 2017 and 2018 datasets

	EMBER 2017	**EMBER 2018**
Accuracy (%)	97.53	94.09
Precision (%)	98.85	90.14
F1 Score (%)	95.23	88.66
Recall (%)	95.40	88.85
AUC (%)	99.13	91
TPR (%)	97.52	91.11
FPR (%)	0.05304	0.1571

Fig. 4 (a) presents the accuracy graph for the EMBER 2017 dataset in which the training and validation graph is shown. For EMBER 2017 dataset, we achieved an accuracy of 97.53%.

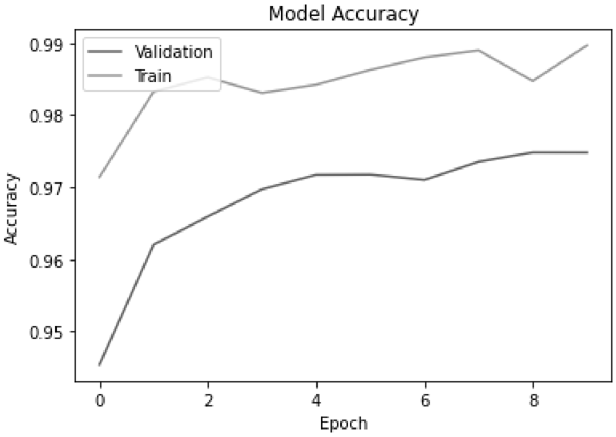


Fig.4 (a). Accuracy graph for EMBER 2017

Fig. 4 (b) presents the loss graph for the EMBER 2017 dataset in which the training and validation graph is shown. In the figure below, both losses are getting reduced while the model is trained for a greater number of epochs.

Fig. 5 presents the confusion matrix for both classes, Malicious and Benign. Here clearly how the number of samples classified after training is visible.

Fig. 6 (a) presents the accuracy graph for the EMBER 2018 dataset in which the training and validation graph is shown. For EMBER 2018 dataset we achieved an accuracy of 94.09%.

Fig. 6 (b) presents the loss graph for the EMBER 2018 dataset in which the training and validation graph is shown. In this figure, the loss is getting reduced as the number of epochs increases.

Fig. 7 presents the confusion matrix for EMBER 2018 dataset, which shows the sample distribution results after classification.

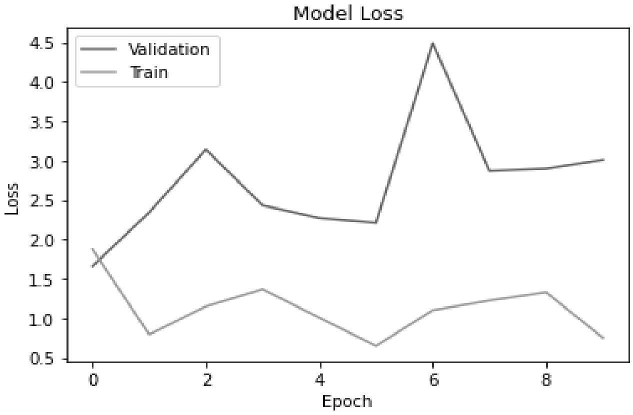


Fig.4. (b). Loss graph for EMBER 2017

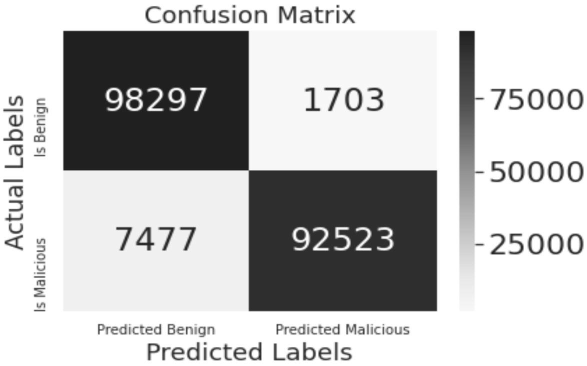


Fig.5. Confusion matrix for the EMBER 2017 dataset

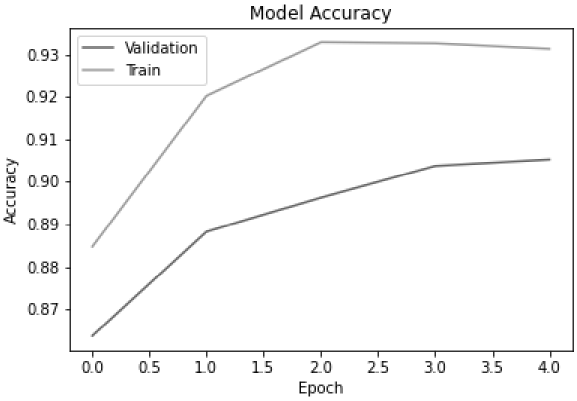


Fig.6 (a). Accuracy graphs for EMBER 2018

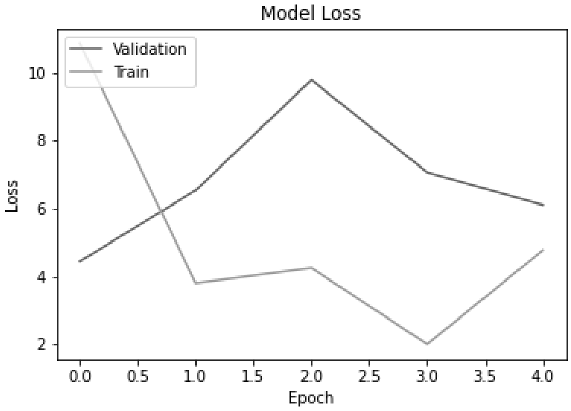


Fig.6 (b). Loss graphs for EMBER 2018

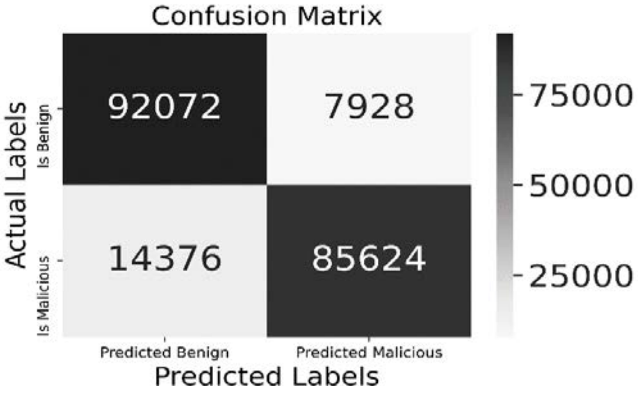


Fig.7. Confusion matrix for the EMBER 2018 dataset

Table 6 presents the proposed model comparison with the present research work. Existing research work is compared based on the Detection accuracy rate (TPR). Here you can see the proposed model outperforms existing models in terms of TPR as well as time. Some researchers used feature selection, but our model achieved comparable results by considering more features, 2381. Even using a greater number of features than the existing models, our model consumed less amount of time than the existing one.

Table 6. Comparison with existing models

Model	Input	Specifications	TPR (%)	Training time
MalConv [56]	Raw Binaries	2 NVIDIA TITAN X (Pascal) GPUs	92.20	10 days (25 h/epoch)
EMBER [57]	2351 Value Vectors	8 vCPUs (2015 MacBook Pro i7)	92.99	20 hrs
H.D. Pham (D.T. model) [58]	1711 Value Vectors	24 vCPUs (Google Compute Engine)	97.57	5 mins
Cangshuai Wu [59]	-	-	93.5	-
Proposed model	2381 Value vectors	Intel(R) Xeon(R) CPU (Google Colab)	97.52	40 mins (10 Epochs)

Researchers [58] not mentioned the number of epochs trained during research. Hence, we consider if a single epoch gets trained in 5 minutes, then our model takes 4 minutes for training a single epoch even with a greater number of features.

Here our model achieved the results as per the objectives:

- Outperformed the existing research works with respect to the computation time required to execute and get the results.
- Even by using a greater number of features which shows even with the lesser number of features author [58] model failed to produce satisfactory results. In future by doing feature selection our model will outperform the results.
- As per Table 7, authors of [60] made evaluation based on precision matrix, our model is ahead of the existing results by 1.85%.

Table 7 presents the comparison of the proposed model with the existing one based on the Precision metric. In this table precision score of the current models is less as compared to the proposed model.

Table 7. Comparison with the existing model based on Precision.

Author	Model	Precision (%)
Subhojeet Pramanik [60]	CNN	95
	FFN	97
Proposed model	NN	98.85

Table 8. K-fold Cross Validation

	K=3 (%)	K=5 (%)	K=7 (%)
EMBER 2017	97.35	97.46	97.53
EMBER 2018	93.68	93.87	94.09

Table 8 shows the cross-validation results which are used for obtained results analysis and defines the reliability of results across several cross folds. We applied three different values of K as 3,5 and 7 for the evaluation.

5. Conclusions

Working on Static PE files is an attempt to detect malware at earlier stages; so that later, it will not affect any working environment. Several researchers found it is hard to identify correct static PE files for the problem. Some research faced problems while extracting features from the PE files and tried various extraction methods. Few of them achieved satisfactory results, but the time and resources took for the computation were more. In this research work, we tried to solve these problems by creating a malware detection and classification model based on Static PE files of malware, which is capable of extracting features as well as cleaning the redundancies or missing information and organize the extracted features into the standard range. To address the issues related to resource and time consumption, we designed a deep learning model with only dropout and dense layers. For the experimentation, we used the EMBER dataset, which is published by Endgame Inc. We performed the training on both EMBER 2017 and 2018 datasets for ten epochs using 2381 features and a learning rate of 0.01. We achieved 97.53% and 94.09% results, respectively. The results obtained are compared with the existing model based on two statistical measures Precision and time required for execution. Our model based on EMBER 2017 dataset achieved 98.85% precision, which is 1.85% more than other models. It took 4 minutes to train one epoch, which is shorter than existing models. These obtained results will help in identifying malware present in static PE files.

This research classifies the malicious codes PE files at the earlier static analysis phase so that the appropriate prevention measures need to be taken in later phases. This research work defines the usage and importance of dealing with datasets and various data pre-processing techniques available in this domain and provides a systematic flow for the solving classification problems with stages instead of creating and feeding dataset to the deep learning models.

In the future, we are planning to work on removing impurities from the dataset, adding, or removing layers of the model, tuning the parameters for the resources shading, and applying other data scaling techniques.

References

[1] Av-test.org. 2020. Malware Statistics & Trends Report | AV-TEST. [online] Available at: https://www.av-test.org/en/statistics/malware/ [Accessed 14 August 2020].

[2] Li, Bo, Kevin Roundy, Chris Gates, and Yevgeniy Vorobeychik. “Large-scale identification of malicious singleton files.” In Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, pp. 227-238. 2017.

[3] Firdausi, Ivan, Alva Erwin, and Anto Satriyo Nugroho. “Analysis of machine learning techniques used in behavior-based malware detection.” In 2010 second international conference on advances in computing, control, and telecommunication technologies, pp. 201-203. IEEE, 2010.

[4] Damodaran, Anusha, Fabio Di Troia, Corrado Aaron Visaggio, Thomas H. Austin, and Mark Stamp. “A comparison of static, dynamic, and hybrid analysis for malware detection.” Journal of Computer Virology and Hacking Techniques 13, no. 1 (2017): 1-12.

[5] Raffetseder, Thomas, Christopher Kruegel, and Engin Kirda. “Detecting system emulators.” In International Conference on Information Security, pp. 1-18. Springer, Berlin, Heidelberg, 2007.

[6] Garfinkel, Tal, Keith Adams, Andrew Warfield, and Jason Franklin. “Compatibility Is Not Transparency: VMM Detection Myths and Realities.” In HotOS. 2007.

[7] Carpenter, Matthew, Tom Liston, and Ed Skoudis. “Hiding virtualization from attackers and malware.” IEEE Security & Privacy 5, no. 3 (2007): 62-65.

[8] Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. “Going deeper with convolutions.” In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1-9. 2015.

[9] Graves, Alex, Abdel-rahman Mohamed, and Geoffrey Hinton. “Speech recognition with deep recurrent neural networks.” In 2013 IEEE international conference on acoustics, speech and signal processing, pp. 6645-6649. IEEE, 2013.

[10] Zhang, Xiang, and Yann LeCun. “Text understanding from scratch.” arXiv preprint arXiv:1502.01710 (2015).

[11] Schultz, Matthew G., Eleazar Eskin, F. Zadok, and Salvatore J. Stolfo. “Data mining methods for detection of new malicious executables.” In Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001, pp. 38-49. IEEE, 2000.

[12] Kolter, J. Zico, and Marcus A. Maloof. “Learning to detect and classify malicious executables in the wild.” Journal of Machine Learning Research 7, no. Dec (2006): 2721-2744.

[13] Tian, Ronghua, Lynn Batten, Rafiqul Islam, and Steve Versteeg. “An automated classification system based on the strings of trojan and virus families.” In 2009 4th International Conference on Malicious and Unwanted Software (MALWARE), pp. 23-30. IEEE, 2009.

[14] McDermott, J., James Kirby, B. Montrose, Travis Johnson, and M. Kang. “Re-engineering Xen internals for higher-assurance security.” Information Security Technical Report 13, no. 1 (2008): 17-24.

[15] Al-Dujaili, Abdullah, Alex Huang, Erik Hemberg, and Una-May O’Reilly. “Adversarial deep learning for robust detection of binary encoded malware.” In 2018 IEEE Security and Privacy Workshops (SPW), pp. 76-82. IEEE, 2018.

[16] Nataraj, Lakshmanan, Sreejith Karthikeyan, Gregoire Jacob, and Bengaluru (Bengaluru) S. Manjunath. “Malware images: visualization and automatic classification.” In Proceedings of the 8th international symposium on visualization for cyber security, pp. 1-7. 2011.

[17] Owezarski, Philippe. “Unsupervised classification and characterization of honeypot attacks.” In 10th International Conference on Network and Service Management (CNSM) and Workshop, pp. 10-18. IEEE, 2014.

[18] Wang, Cheng, Jianmin Pang, Rongcai Zhao, and Xiaoxian Liu. “Using API sequence and Bayes algorithm to detect suspicious behavior.” In 2009 International Conference on Communication Software and Networks, pp. 544-548. IEEE, 2009.

[19] Xu, J-Y., Andrew H. Sung, Patrick Chavez, and Srinivas Mukkamala. “Polymorphic malicious executable scanner by API sequence analysis.” In Fourth International Conference on Hybrid Intelligent Systems (HIS’04), pp. 378-383. IEEE, 2004.

[20] Ye, Yanfang, Tao Li, Qingshan Jiang, and Youyu Wang. “CIMDS: adapting postprocessing techniques of associative classification for malware detection.” IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 40, no. 3 (2010): 298-307.

[21] Ugarte-Pedrero, Xabier, Igor Santos, Borja Sanz, Carlos Laorden, and Pablo Garcia Bringas. “Countering entropy measure attacks on packed software detection.” In 2012 IEEE Consumer Communications and Networking Conference (CCNC), pp. 164-168. IEEE, 2012.

[22] Saxe, Joshua, and Konstantin Berlin. “Deep neural network-based malware detection using two-dimensional binary program features.” In 2015 10th International Conference on Malicious and Unwanted Software (MALWARE), pp. 11-20. IEEE, 2015.

[23] Raff, Edward, Jared Sylvester, and Charles Nicholas. “Learning the pe header, malware detection with minimal domain knowledge.” In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, pp. 121-132. 2017.

[24] Van Nhuong, Nguyen, Vo Thi Yen Nhi, Nguyen Tan Cam, Mai Xuan Phu, and Cao Dang Tan. “Semantic set analysis for malware detection.” In IFIP International Conference on Computer Information Systems and Industrial Management, pp. 688-700. Springer, Berlin, Heidelberg, 2015.

[25] Nguyen, Vu Thanh, Toan Tan Nguyen, Khang Trong Mai, and Tuan Dinh Le. “A combination of negative selection algorithm and artificial immune network for virus detection.” In International Conference on Future Data and Security Engineering, pp. 97-106. Springer, Cham, 2014.

[26] Nath, Hiran V., and Babu M. Mehtre. “Static malware analysis using machine learning methods.” In International Conference on Security in Computer Networks and Distributed Systems, pp. 440-450. Springer, Berlin, Heidelberg, 2014.

[27] Kolter, J. Zico, and Marcus A. Maloof. “Dynamic weighted majority: An ensemble method for drifting concepts.” Journal of Machine Learning Research 8, no. Dec (2007): 2755-2790.

[28] Kolter, Jeremy Z., and Marcus A. Maloof. “Learning to detect malicious executables in the wild.” In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 470-478. 2004.

[29] Kolter, Jeremy Z., and Marcus A. Maloof. “Using additive expert ensembles to cope with concept drift.” In Proceedings of the 22nd international conference on Machine learning, pp. 449-456. 2005.

[30] Dube, Thomas, Richard Raines, Gilbert Peterson, Kenneth Bauer, Michael Grimaila, and Steven Rogers. “Malware type recognition and cyber situational awareness.” In 2010 IEEE Second International Conference on Social Computing, pp. 938-943. IEEE, 2010.

[31] Dube, Thomas, Richard Raines, Gilbert Peterson, Kenneth Bauer, Michael Grimaila, and Steven Rogers. “Malware target recognition via static heuristics.” Computers & Security 31, no. 1 (2012): 137-147.

[32] Dube, Thomas E. A novel malware target recognition architecture for enhanced cyberspace situation awareness. No. AFIT/DCE/ENG/11-07. AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING AND MANAGEMENT, 2011.

[33] Dube, Thomas E., Richard A. Raines, Michael R. Grimaila, Kenneth W. Bauer, and Steven K. Rogers. “Malware target recognition of unknown threats.” IEEE Systems Journal 7, no. 3 (2012): 467-477.

[34] Dube, Thomas E., Richard A. Raines, and Steven K. Rogers. “Malware target recognition.” U.S. Patent 8,756,693 issued June 17, 2014.

[35] Zhang, Boyun, Jianping Yin, Jingbo Hao, Dingxing Zhang, and Shulin Wang. “Malicious codes detection based on ensemble learning.” In International conference on autonomic and trusted computing, pp. 468-477. Springer, Berlin, Heidelberg, 2007.

[36] Lyda, Robert, and James Hamrock. “Using entropy analysis to find encrypted and packed malware.” IEEE Security & Privacy 5, no. 2 (2007): 40-45.

[37] Santos, Igor, Felix Brezo, Borja Sanz, Carlos Laorden, and Pablo Garcia Bringas. “Using opcode sequences in single-class learning to detect unknown malware.” IET information security 5, no. 4 (2011): 220-227.

[38] Santos, Igor, Felix Brezo, Xabier Ugarte-Pedrero, and Pablo G. Bringas. “Opcode sequences as representation of executables for data-mining-based unknown malware detection.” Information Sciences 231 (2013): 64-82.

[39] Santos, Igor, Javier Nieves, and Pablo G. Bringas. “Semi-supervised learning for unknown malware detection.” In International Symposium on Distributed Computing and Artificial Intelligence, pp. 415-422. Springer, Berlin, Heidelberg, 2011.

[40] Bilar, Daniel. “Opcodes as predictor for malware.” International journal of electronic security and digital forensics 1, no. 2 (2007): 156-168.

[41] Shafiq, M. Zubair, S. Tabish, and Muddassar Farooq. “PE-probe: leveraging packer detection and structural information to detect malicious portable executables.” In Proceedings of the Virus Bulletin Conference (VB), vol. 8. 2009.

[42] Shafiq, M. Zubair, S. Momina Tabish, Fauzan Mirza, and Muddassar Farooq. A framework for efficient mining of structural information to detect zero-day malicious portable executables. Technical Report, TR-nexGINRC-2009-21, January 2009, available at http://www. nexginrc. org/papers/tr21-zubair. pdf, 2009.

[43] Shafiq, M. Zubair, S. Momina Tabish, Fauzan Mirza, and Muddassar Farooq. “Pe-miner: Mining structural information to detect malicious executables in realtime.” In International Workshop on Recent Advances in Intrusion Detection, pp. 121-141. Springer, Berlin, Heidelberg, 2009.

[44] Liu, Huan, and Hiroshi Motoda, eds. Feature extraction, construction, and selection: A data mining perspective. Vol. 453. Springer Science & Business Media, 1998.

[45] “LIEF - Library to Instrument Executable Formats.” 2020. Lief.Quarkslab.Com. https://lief.quarkslab.com/#download. [Accessed 14 August 2020].

[46] Rahm, Erhard, and Hong Hai Do. “Data cleaning: Problems and current approaches.” IEEE Data Eng. Bull. 23, no. 4 (2000): 3-13.

[47] Kassambara, Alboukadel. Practical guide to cluster analysis in R: Unsupervised machine learning. Vol. 1. Sthda, 2017.

[48] "Sklearn.Preprocessing. Standardscaler — Scikit-Learn 0.23.2 Documentation". 2020. Scikit-Learn.Org. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html. [Accessed 14 August 2020].

[49] Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting.” The journal of machine learning research 15, no. 1 (2014): 1929-1958.

[50] Chollet, Francois. Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek. MITP-Verlags GmbH & Co. KG, 2018.

[51] “Classification With Tensorflow and Dense Neural Networks.” 2020. Medium. https://heartbeat.fritz.ai/classification-with-tensorflow-and-dense-neural-networks-8299327a818a#:~:text=What%20is%20a%20dense%20neural%20network%3F&text=Each%20neuron%20in%20a%20layer,those%20in%20the%20next%20layer. [Accessed 14 August 2020].

[52] “Endgameinc/Ember.” 2020. Github. https://github.com/endgameinc/ember. [Accessed 14 August 2020].

[53] Powers, David Martin. “Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation.” (2011).

[54] Oyama, Yoshihiro, Takumi Miyashita, and Hirotaka Kokubo. “Identifying Useful Features for Malware Detection in the Ember Dataset.” In 2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW), pp. 360-366. IEEE, 2019.

[55] Pham, Huu-Danh, Tuan Dinh Le, and Thanh Nguyen Vu. “Static PE malware detection using gradient boosting decision trees algorithm.” In International Conference on Future Data and Security Engineering, pp. 228-236. Springer, Cham, 2018.

[56] Raff, Edward, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles Nicholas. “Malware detection by eating a whole exe.” arXiv preprint arXiv:1710.09435 (2017).

[57] Anderson, Hyrum S., and Phil Roth. “Ember: an open dataset for training static pe malware machine learning models.” arXiv preprint arXiv:1804.04637 (2018).

[58] Pham, Huu-Danh, Tuan Dinh Le, and Thanh Nguyen Vu. “Static PE malware detection using gradient boosting decision trees algorithm.” In International Conference on Future Data and Security Engineering, pp. 228-236. Springer, Cham, 2018.

[59] Wu, Cangshuai, Jiangyong Shi, Yuexiang Yang, and Wenhua Li. “Enhancing machine learning based malware detection model by reinforcement learning.” In Proceedings of the 8th International Conference on Communication and Network Security, pp. 74-78. 2018.

[60] Pramanik, Subhojeet, and Hemanth Teja. “EMBER-Analysis of Malware Dataset Using Convolutional Neural Networks.” In 2019 Third International Conference on Inventive Systems and Control (ICISC), pp. 286-291. IEEE, 2019.

[61] Tumsa, Sisay. "Application of Artificial Neural Networks for Detecting Malicious Embedded Codes in Word Processing Documents." Pezzottaite Journals 8, no. 2 (2019).

[62] Alile S.O, Egwali A.O, " A Bayesian Belief Network Model For Detecting Multi-stage Attacks With Malicious IP Addresses ", International Journal of Wireless and Microwave Technologies, Vol.10, No.2, pp. 30-41, 2020.

[63] Rabia Tahir,"A Study on Malware and Malware Detection Techniques", International Journal of Education and Management Engineering, Vol.8, No.2, pp.20-30, 2018.

Authors’ Profiles

Sumit S. Lad, pursuing the Master of Technology in Computer Science and Engineering, from Rajarambapu Institute of Technology, Rajaramnagar, Sangli, MS, India. Completed B.E. from Shivaji University, Kolhapur, MS, India. His areas of interest are cloud computing, machine learning and classification problems.

Amol C. Adamuthe received Ph.D. from Mumbai University and Master of Technology in Computer Engineering from Dr. B. A. Technological University, Lonere, MS, India. He is currently an Assistant Professor at Rajarambapu Institute of Technology, Sakharale, Sangli, MS, India. His areas of interest are technology forecasting, optimization algorithms, soft computing, and cloud computing. He has published more than 50 papers at the international and national level. He is the recipient of the Institution of Engineers (India) Young Engineers Award for year 2018-19 in Computer Engineering discipline.

How to cite this paper: Sumit S. Lad., Amol C. Adamuthe, "Improved Deep Learning Model for Static PE Files Malware Detection and Classification", International Journal of Computer Network and Information Security(IJCNIS), Vol.14, No.2, pp.14-26, 2022. DOI: 10.5815/ijcnis.2022.02.02