ASTES

Advances in Science, Technology and Engineering Systems Journal Vol. 5, No. 4, 207-216 (2020)

www.astesj.com

ASTESJ

ISSN: 2415-6698

Malware Classification Based on System Call Sequences Using Deep Learning

Rizki Jaka Maulana, Gede Putra Kusuma*

Computer Science Department, BINUS Graduate Program, Bina Nusantara University, Jakarta, 11480, Indonesia

ARTICLE INFO

Article history:
Received: 27 March, 2020
Accepted: 06 June, 2020
Online: 22 July, 2020

Keywords:
Malware Classification
Malware Detection
System Call Sequence
Deep Learning
LSTM Model

ABSTRACT

Malware has always been a big problem for companies, government agencies, and individuals because people still use it as a primary tool to influence networks, applications, and computer operating systems to gain unilateral benefits. Until now, malware detection with heuristic and signature-based methods are still struggling to keep up with the evolution of malware. Machine learning is known to be able to automate the work needed to detect families of existing and newly discovered malware. Unfortunately, the machine learning method using Support Vector Machine (SVM) for detecting malware can only reach a low level of accuracy. In this work, we propose a dynamic analysis method and uses a system call sequence to monitor malware behavior. It uses the word2vec technique as word embedding and implements deep learning models, namely Long Short-Term Memory (LSTM) and Nested LSTM, as classifiers. To compare with existing machine learning approach, we also apply the Support Vector Machine (SVM) as a benchmark method. The Nested LSTM gets an accuracy of 93.11%, while the LSTM gets the best accuracy of 98.61%. The LSTM also achieved the best performance in terms of average precision at 97.57%, the average recall at 97.29%, and the average score of f1 at 97.43%. We have found that our model is lightweight but powerful for detecting malware with significant accuracy.

1. Introduction

The high use of the internet increases the level of connectivity of electronic devices, making questions about the integrity of the system. Conventionally, software and computer systems are developed for good purposes. However, some software was developed to produce crime (malware). Malware is a common word used for programs that have malicious code snippets that can cause significant threats to computer users or any digital device. Malware can contain malicious code viruses, worms, Trojan horses, can also make a back door to divulge personal information or control a person's system. Through malware, serious crimes can be done; This is why malware detection is needed [1]. To detect malware definitions must be made for analysis of which malware is essential. Malware analysis consists of analyzing various aspects of malware so that malware can be detected [2]. The definition of malware is also known as a signature/signature. This signature is used by virus scanners known as anti-viruses to detect malware. The research will experiment on seven types of malware, which are adware, backdoor, packed, riskware, trojan, virus, and worm.

Traditional malware detection is done on susceptible files that are not processed. This is mostly done with a signature, heuristic, and behavioral approach. The signature approach looks for static patterns of malware known in suspicious files [3]. Research has shown that the signature approach is very weak in dealing with polymorphic and metamorphic malware. The heuristic approach checks the characteristics of suspicious malware from suspicious files. Despite being able to detect unknown malware, they are very high at the false-positive level.

The Behavioral Approach monitors the implementation of programs to monitor suspicious behavior. Although this approach can detect different malware variants, this approach also has a high false-positive [4]. To help malware analysts retrieve useful information from large malware samples, the need for automatic classification in statistical variants is needed. Malware detection based on a signature cannot overcome this variant because it does not take polymorphic malware into account. Polymorphic is a form of malware that frequently always changes its identifiable features to evade detection. Furthermore, such a system can be easily avoided.

*Corresponding Author: Gede Putra Kusuma, inegara@binus.edu

The most important event that can be tracked to determine malware behavior is the system call. Before malware performs a malicious action, malware needs to use the operating system (OS) service of the target. For each activity that is carried out, such as opening a file, running a thread, writing a command to the administrator, or opening a network connection, interaction with the operating system is required. This interaction is carried out via the API call system of the target OS. Therefore, monitoring the behavior of malware is very important to monitor the order of system calls during malware execution. Different malware families certainly have different goals.

Detected malware is easily handled mainly by elimination. However, the current nature of malware is polymorphic and metamorphic, making them difficult to detect in traditional ways. They disguise their structure but not their operations. Because all malware must be executed to carry out its malicious actions successfully, some studies [5], analyze API calls to detect malware in high accuracy execution. However, this detection ends by marking malware or not malware [6]. It does not classify malware into its type (viruses, worms, Trojans, etc.). Classification is important because it helps simplify the course of action to neutralize it.

Research on malware classification has been done before. However, these studies do not use the Word2vec method. One example is a study of classification in system call sequences conducted in 2019, wherein that study the classification contained nine types of malware, namely kelihos_v3, vundo, rammit, lollipop, simda, tracur, obfuscator.ACY and gatak. The methods used are text and hex commands and LSTM [7]. Also, in 2016 there was research on evaluating machine learning methods such as the Hidden Markov Model [8] and SVM [9] in determining malware classification.

We use word embedding techniques in processing to convert malware system call sequences into vectors to achieve an increase in capturing the relationship between n-grams in the system call sequence and then proceeding to LSTM for the classification process. In essence, this approach expected to improve accuracy and precision for most families of malware, which brings a significant improvement from the methods used by previous researchers and thus can help to classify malware more accurately.

All sections of the paper have been organized as follows. Section 2 discusses related works on the detection and classification of malware. In section 3 provides details of the background Theories, word embedding, and deep learning methods. Section 4 discusses the details of the dataset, the methodology used, and the evaluation design. Section 5 discusses the details of all experimental results, including training and testing results. The conclusion and future work are places in section 6.

2. Related Works

Previous researchers have shown that using program behavior features such as API calls can detect malware, including metamorphic and polymorphic malware, with high accuracy. This is because, at a higher level, malware disguises itself by changing their behavior or continuously changing their signatures. However, to cause damage, they must execute and change execution behavior more difficult. This can make them harmless. Therefore, this approach targets malware at the execution level.

The first researcher who used a deep learning-based malware detection (DLMD) approach relied on static methods to predict behaviors that can be executed using system call sequences that provide sequences taken from running processes. Using SVM and CNN the results show that this method is quite effective in detecting polymorphic and metamorphic malware with an accuracy and detection rate of 89% to 96% [10]. In the proposed DLMD technique, SVM is used as a feature selector and CNN autoencoder is used as a feature extractor. After that, a Multilayer perceptron is used as a classifier.

Other researchers develop an 18-layer deep residual network to be issued bytecode to a 3-channel RGB image and then apply deep learning to classify malware. To convert malware to images, they first convert malware binaries to 8-bit vectors (bytecodes) [11]. After that, the bytecodes are converted into grayscale images with contribution values from 0 to 255, each vector that turns into pixels with added values from 0 to 255. In the next step, they then convert the grayscale images to 3-channel RGB images with duplicate the grayscale channel three times and then collect all three channels to create an RGB image. Their experimental results show that the network residual model achieves an average accuracy of 86.54% with 5-fold cross-validation.

In [12], the author proposed a new malware detection method based on Deep Graph Convolutional Neural Networks (DGCNNs) to learn directly from the sequence of API calls and related behavior graphs. The experimental results show that the model reaches a similar area under the ROC curve (AUC-ROC) and F1-Score of Long-Short Term Memory (LSTM) networks that produce up to 96%.

In [13], the author proposed a method for detecting malware variants that are packaged based on sensitive system calls and the Deep Belief Network. Different experimental groups and different data samples were used for analysis. The 10-fold cross-validation method is used for classification. Theoretical analysis and experimental results show that the proposed method can detect packed malware which reaches an accuracy of 92% and requires a detection time of fewer than 0.001 seconds.

In [14], the author proposed a conventional approach with deep learning-based using Recurrent Neural Networks (RNN) that are vulnerable to redundant API injection. They investigated the effectiveness of Convolutional Neural Networks (CNN) against injection of redundant APIs. Their malware detection system converts malware files into image representations and classifies image representations with CNN. CNN is implemented with spatial pyramid pooling layers (SPP) to handle various input sizes. They also evaluated the effectiveness of SPP and image color space (greyscale / RGB) by measuring system performance on unaltered data and adversarial data with the injected redundant API. The results show that Naive SPP implementation is not impractical due to memory constraints and effective greyscale imaging against redundant API injection.

The last researcher proposed an approach of how deep learning architecture using the stacked AutoEncoders (SAEs) model can be designed for intelligent malware detection. The SAEs model

functions as a greedy layerwise training operation for unsupervised feature learning, followed by supervised fine-tuning parameters (eg Weights and offset vectors). Based on the representation of different features, various types of classification methods, such as Artificial Neural Networks (ANNs), Support Vector Machines (SVM), Naïve Bayes (NB), and Decision Tree (DT) are used as a model construction to detect malware. Most of these methods are built on shallow learning architectures. Even though they have succeeded in isolating malware detection but shallow learning architectures are still unsatisfactory for malware detection problems [15]. The experimental results of the method showed that the proposed method achieves 96% accuracy. The bibliography comparison of previous works are summarised in Table 1.

Table 1: Bibliography comparison

Author	Dataset	Method	Accuracy	Class Malware
Rafique, Ali, Qureshi, Khan, & Mirza, 2019 [10]	10.868 Portable Execution (PE) files	SVM & CNN	89% - 96%	6 Class Malware
Yan Lu, Jonathan Graham, Jiang Li, 2019 [11]	2949 Portable Execution (PE) files	Deep Residual Network	86.54%	5 Class Malware
Oliveira, Julho, & Julho, 2019 [12]	42.797 Portable Executable (PE) files	LSTM & DGCNN	92.7 - 96.8%	2 Class Malware
Zhang, Chang, Han, & Zhang, 2020 [13]	7.195 Portable Executable (PE) files	SVM & Deep Belief Network	86.3% - 92.6%	4 Class Malware
Ke He, Dong-Seong Kim, 2018 [14]	2.413 Portable Executable (PE) files	CNN	95%	2 Class Malware
Hardy, Chen, Hou, Ye, & Li, 2016 [15]	22.500 Portable Executable (PE) files	SVM, NB, DT, ANN & DL4MD	92% - 95%	2 Class Malware

Based on the results of the literature review, there have been previous studies that have tried to classify malware based on system call sequences data. But, the methods used before did not achieve high classification accuracy. In other fields, many methods of deep learning have proven to be more accurate and therefore we use deep learning and the word2vec as a word embedding to improve accuracy. Since deep learning models are used, therefore we do not use feature extraction specifically like the study above. However, we use a word embedding, which convert the input text into numeric data as input to the LSTM model. As a result, it will increase classification accuracy.

3. Background Theories

3.1. Word2Vect

Word2vec is a two-layer neural network that can process text by converting words into vectors or can also be called "vectorization." Input from word2vec is a collection of text, and the output is a collection of vectors. Feature vector representing words in a corpus. Word2vec is not a deep neural network. Word2vec works by converting text into numerical forms which can then be translated by deep neural networks. Word2Vec is a word embedding technique that is quite popular and was developed by[16] at Google.

Word2vec can also be applied to codes, likes, playlists, social media graphics, sentiment sentences, and other verbal or symbolic series where patterns can be seen. The purpose of word vectorization is to group word vectors that are similar in vector space, which can later detect mathematical equations. Word2vec functions by making a distributed numerical vector representation of a word. For example like in the context of an individual word.

Word2vec works automatically. With enough data usage and context. Word2vec can make very accurate guesses about the meaning of the words based on previous appearance or interpretation. These guesses are used to build the association of words with other words (e.g. "Male" means "boy" and "woman" means "girl"), or classify a document and then group them according to their topic. Clusters can form the basis of sentiment analysis, e-commerce, search, malware analysis, and recommendations in areas such as scientific research and legal discovery. The output of word2vec is in the form of vocabulary where each item has a vector, which can be entered into further processes such as machine learning or deep learning. Also, it can be used just to detect the relationship between these words.

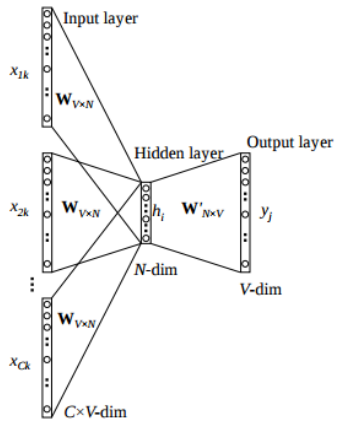

Figure 1: Continuous bag-of-words architecture

Figure 1 is a Word2vec Continuous-bag-of-words (CBOW) model. The way CBOW works is to take the context of each word and then make it as input and try to predict words that fit the context. As an example, When trying to predict the current target word (the center word) based on the source context words (surrounding words) [17]. If we make a simple sentence like "the black cat jump over the very big goat" this can be pairs of (context_window, target_word) where if we consider a context window of size 2, we have examples like ([the, cat], black), ([cat, over], jump), ([very, goat], big) and so on. This model tries to predict target_word based on context_window words.

3.2. Long Short-Term Memory (LSTM)

LSTM was first introduced by Sepp Hochreiter and Jurgen Schmidhuber in 1997 [18]. LSTM is a type of repetitive neuron that has been shown to increase the ability of RNN. LSTM can remove the effects of the problem by vanishing and bursting gradients, and is better to data-sensitivity relationships [19]. LSTM launched the forget gate inside the LSTM neuron, which allows accessing the information requested by the neuron allowing its access to focus on the critical parts and discard the information that is not useful.

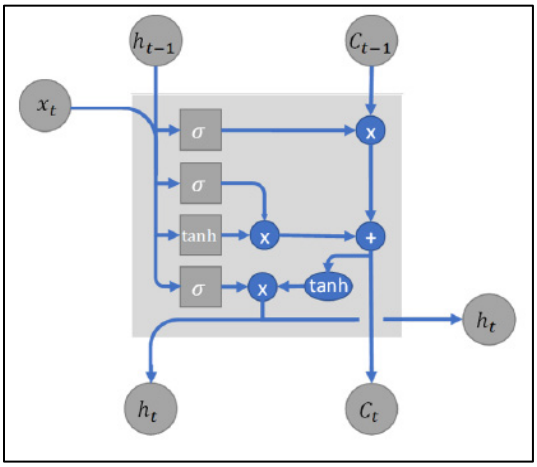

Figure 2: LSTM architecture

Figure 2 shows the structure of the LSTM. The key to LSTM architecture is its cell state. Cell state can be interpreted as a memory of a network and can delete or add information to a structure called a gate. For each "t" time-step in LSTM can be described by using this formula [20]:

$$f_t = \sigma\left(W_t * x_t + U_f * h_{t-1} + b_f\right) \quad (1)$$

$$i_t = \sigma\left(W_i * x_t + U_i * h_{t-1} + b_i\right) \quad (2)$$

$$\tilde{C}_t = tanh\left(W_C * x_t + U_C * h_{t-1} + b_C\right) \quad (3)$$

$$C_t = i_t * \tilde{C}_t + f_t * C_{t-1} \quad (4)$$

$$O_t = \sigma\left(W_o * x_t + U_o * h_{t-1} + b_o\right) \quad (5)$$

$$h_t = o_t * \tanh(C_t) \quad (6)$$

while f_t is forget gate, i_t is the input gate, O_t is output gate C_t is a memory cell, h_t is a hidden layer, x_t is input when time "t", σ is sigmoid activation function, tanh is hyperbolic tangent activation function, $W_t\ W_i W_C W_o U_f U_i\ U_C U_o$ are weight matrices for controlling the input and $b_f b_i b_C b_o$ are bias vector.

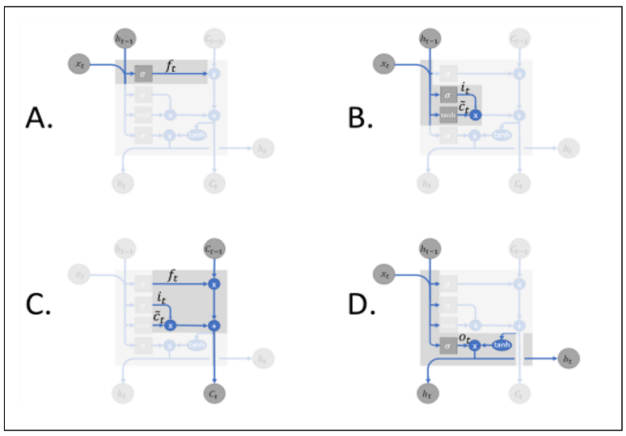

Figure. 3: LSTM steps

Figure 3 shows the steps contained in the LSTM model architecture. There are four steps in LSTM namely: Step A. First, the model needs to determine what needs to be changed from the state of the cell. Figure 3 (A) will have a value called forget gate f_t. The input of this step is the output of the previous step, which written by h_{t-1} and x_t input. The activation function will give a result of "0" or "1", where "0" means "not let anything pass" and "1" means "remember everything".

The next step is to determine what information will be added to the state of the cell. Shown by Figure 3 (B), Equations (2), and (3). At this stage, the input is h_{t-1} and x_t. The first layer is called the sigmoid layer, which serves to determine which part to be updated. And the tahn layer is to create a new candidate value C_t. In the next step, the two layers will be combined to update the status of C_t cell.

In step C, the old cell will be multiplied by f_t so that it can forget things that are no longer needed, so new information that will enter can be easily added to the cell's memory status. This section is shown in Figure 3 (C) and Equation (4). In the final step, the output of h_t is shown in figure 3 (D), Equation (5), and (6). Output results are based on the state of the cell but in the state that is being filtered. Initially, the sigmoid layer was applied to the previous output h_{t-1} and x_t input to determine the O_t gate output value. The resulting value is between "0" and "1", which indicates part of the cell state is output. Then the state of a cell C_t is changed by the tanh function to get the value between "-1" and "1". The value of the changed cell status is then multiplied by the output value at the O_t gate, which ends with h_t output and this output will be used for the next step in the model.

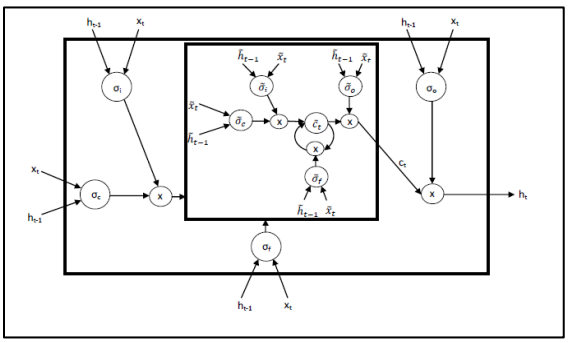

Figure 4: Nested LSTM architecture

3.3. Nested Long Short-Term Memory (NLSTM)

Figure 4 is an architectural drawing of a Nested LSTM [21]. Nested LSTM is a simple extension of the LSTM model that adding depth through nesting into the model. Inside Nested LSTM there are memory cells that make up internal memory and can only be accessed through external memory cells by applying a temporal hierarchy. The gate output in LSTM encodes the intuition that irrelevant memories at the current time step may still need to be remembered. Nested LSTM uses this intuition to create a temporal memory hierarchy. In Nested LSTM, access to internal memories is maintained in the same way, so that long-term information that is only situationally relevant can be selectively accessed. The equation in Nested LSTM can be described as follows:

$$\tilde{h}_{t-1} = f_t * C_{t-1} \quad (7)$$

$$\tilde{x}_t = i_t * tanh\,(x_t W_{xc} + h_{t-1} W_{hc} + b_C) \quad (8)$$

$$C_t = \tilde{h}_{t-1} + \tilde{x}_t \quad (9)$$

$$\tilde{\imath}_t = tanh\,(\tilde{x}_t \widetilde{W}_{xi} + \tilde{h}_{t-1} \widetilde{W}_{hi} + \tilde{b}_i) \quad (10)$$

$$\tilde{f}_t = tanh\,(\tilde{x}_t \widetilde{W}_{xf} + \tilde{h}_{t-1} \widetilde{W}_{hf} + \tilde{b}_f) \quad (11)$$

$$\tilde{C}_t = \tilde{f}_t * \tilde{c}_{t-1} + \tilde{\imath}_t * \tanh(\tilde{x}_t \widetilde{W}_{xc} + \tilde{h}_{t-1} \widetilde{W}_{hc} + \tilde{b}_c) \quad (12)$$

$$\tilde{o}_t = \tanh(\tilde{x}_t \widetilde{W}_{xo} + \tilde{h}_{t-1} \widetilde{W}_{ho} + \tilde{b}_o) \quad (13)$$

$$\tilde{h}_t = \tilde{o}_t * \tanh(\tilde{C}_t) \quad (14)$$

$$C_t = \tilde{h}_t \quad (15)$$

Where, f_t is forget gate, $\tilde{f}_t$ is inner forget gate, i_t is the input gate, $\tilde{\imath}_t$ is inner input gate, O_t is the output gate, $\tilde{o}_t$ is inner output gate, C_t is a memory cell, $\tilde{C}_t$ is an inner memory cell, h_t is a hidden layer, $\tilde{h}_t$ is an inner hidden layer, x_t is input when time "t," $\tilde{x}_t$ is inner input when time "t," σ is sigmoid activation function, tanh is hyperbolic tangent activation function, $W_{xc} W_{hc}$ are weight matrices, $\widetilde{W}_{xi} \widetilde{W}_{hi} \widetilde{W}_{xf}\ \widetilde{W}_{hf} \widetilde{W}_{xc} \widetilde{W}_{hc} \widetilde{W}_{xo} \widetilde{W}_{ho}$ are inner weight matrices b_C is bias vector and $\tilde{b}_i\ \tilde{b}_f\ \tilde{b}_c\ \tilde{b}_o$ are the inner bias vector.

3.4. Support Vector Machine (SVM)

SVM is machine learning that is usually used for classification or regression. SVM is also a type of supervised learning. The main purpose of SVM is to determine data with decision boundaries and extend to non-linear boundaries using kernel tricks [22]. SVM is used in many applications such as word sentiment, categorization of text and documents, pattern recognition, face recognition, handwriting analysis, and binary classification. the idea behind SVM is to share data with the best method. The binary classification used to compile we need to classify 2 data sets. In multi-classification, the most frequent method is to create a one-versus-rest classifier (OVA) where each category is divided, and all other categories are combined and to choose the class that classifies collecting data with the largest margins. Divide the class into binary problems. The classifier learning step is carried out by all training data, taking certain class patterns as positive and all other examples as negative. Support Vector Machine has three main parameters, namely, C, gamma, and kernel. The kernel is always used as the Radial Base Function (RBF) because of its best performance [23]. While C and gamma are hyperparameters that have different values and produce different accuracy and results.

4. Research Methodology

4.1. Dataset Generation

We collect malware samples and track the behavior of malware using Cuckoo malware analysis [24]. The malware collection consists of samples collected from two primary sources: Virus Share [25] and GitHub / TheZoo [26]. We chose this source because it provided a large and varied sample Portable Executable (PE) file for evaluation. Because malware authors can use obfuscation and packers code for sub-vertical static analysis, we use dynamic malware analysis to collect data about malware behavior. Then, several tools allow tracking malware execution and gathering logs from the order of execution [27]. We use Cuckoo Sandbox, which is open-source and provides a controlled environment for executing malware. In the dataset experiment, that will be used as many as 13356 data, where the data is divided into three groups, namely training, validation, and testing.

Table 2: Description of malware dataset

Malware	Training Data	Validation Data	Testing data	Total
Adware	2159	719	719	3597
Backdoor	504	167	167	838
Packed	664	220	220	1104
Riskware	733	243	243	1219
Trojan	2484	827	827	4138
Virus	592	197	197	986
Worm	886	294	294	1474
Total	8012	2672	2672	13356

Table 2 shows the distribution of the amount of training and testing data used in this research. At the training stage, the model will be trained using 8012 data, while at the data testing stage will be tested using 2672 data. Experiments will be conducted on both models. Prediction of testing data will be an experimental result where the results will be described through a confusion matrix so that the accuracy of each model is obtained.

4.2. Word Embedding

We extracted the PE file by preprocessing the PE Headers and opcodes from the code section. To use this data in the classification process, we need to make numerical vectors with word embedding. The PE file is run in the Cuckoo sandbox which is a malware analysis tool. Can extract API calls from PE files during execution. The sandbox tool is configured on Ubuntu 18.04.2 LTS along with the Windows 7 virtual environment using the Oracle virtual box where the PE files are executed. Virtual environments help in such a way that malicious files are executed and behave in the same way as in a conventional system [6]. This is very helpful in understanding malware behavior when trying to infect a system.

During PE file execution, the Cuckoo sandbox generates log files. The log file contains snapshots taken during execution (behavior profile) [28]. This is done for every sample that is executed. Each sequence of API calls is recorded according to the

class label specified by Kaspersky [29] and VirusTotal [30]. We determined seven classes of malware (Adware, Backdoor, Packed, Riskware, Trojan, Virus, and Worm). The API call log that has been collected is always long and continuous. We will apply text mining with word2vec techniques. To select API calls that are relevant for classification. Word2vec helps identify a set of API calls that are more common in the malware class. This works in a way that if there is a word API call, it often appears in a class. But when it appears in many other classes, it is not a unique identifier and must be given a lower score. Only the words API calls with high scores or frequently appearing words are considered as PE file profile behavior.

Word2vec has two techniques, namely Skip-gram and Continuous Bag of Words (CBOW). This CBOW method takes the context of each word from the whole sentence or paragraph as input and tries to predict the word for word that fits the context. In contrast, the skip-gram model predicts the meaning of words after searching for their target words, and the author uses CBOW for this research. First, we did a mapping for seven labels and turned it into one-hot encoding. Then, the writer converts the whole sentence to the lower case and removes the punctuations. The next step is to create a word2vec embedding model generator to convert words to vectors with the specified model size.

```
counts=Counter(word_bag).most_common(1200)   # selecting only top 1200 most common words
print(counts)

words_list=[]   # making words list out of tuples present in counts
for c in counts:
    words_list.append(c[0])

[('loadlibraryexw', 136606), ('localalloc', 123640), ('getprocaddress', 98303), ('getmodulehandlew', 79815),
('closehandle', 69078), ('createfilew', 68124), ('localfree', 59950), ('getsystemmetrics', 59885),
('mapviewofileex', 56920), ('getmodulefilenamew', 55725), ('lstrlenw', 55347), ('getcurrentthreadid', 53075),
('getmodulehandlea', 51668), ('getthreadlocale', 49396), ('getversionexw', 48844), ('lstrlena', 47563),
('loadlibraryw', 44079), ('freelibrary', 43570), ('loadresource', 43324), ('createfilemappingw', 41745),
('waitforsingleobjectex', 40288), ('waitforsingleobject', 39071), ('getversionexa', 39035), ('seterrormode',
37751), ('releasedc', 37417), ('comparestringw', 36537), ('gettickcount', 35830), ('disablethreadlibrarycalls',
35516), ('isdbcsleadbyte', 34347), ('searchpathw', 33961), ('lstrcpyw', 33138), ('loadcursorw', 32191),
('lstrcpyna', 32009), ('loadlibraryexa', 30218), ('findresourceexw', 29998), ('lockresource', 29646),
```

Figure 5: Code snippet for Word Bag

The next step is to create a Word Bag with the same number of words counted in various types of malware that is calculated to help determine how relevant a word is to a specific class or how often the word appears in the word bag. The code snippet for Word Bag is shown in Figure 5.

```
word='getversion'
print("Vecotr: ",model1.wv[word])

Vecotr:  [ 2.59232450e+00  6.84048295e-01  6.26165152e-01 -1.40236092e+00
  1.06598151e+00  6.68558717e-01 -1.79842436e+00  3.80435050e-01
  1.46896768e+00  8.00294206e-02 -1.01184219e-01  1.30136698e-01
  9.99615192e-02  1.23285269e-02 -5.94269335e-01 -1.94395602e+00
  1.35559547e+00  1.02511263e+00  1.47519684e+00  1.61271095e-01
 -3.92647773e-01 -6.11079276e-01 -1.14168262e+00 -1.34283960e+00
 -4.08858657e-01 -1.46569645e+00 -1.60189641e+00 -1.31877005e+00
  7.46764421e-01  3.13887417e-01 -2.35597193e-01 -4.39115703e-01
 -1.52561033e+00  4.51700360e-01 -4.99426723e-01 -3.69457185e-01
 -1.37319434e+00  1.59534901e-01 -5.51769376e-01  3.89835179e-01
  6.48715258e-01  1.29709542e+00 -1.66487455e-01 -2.22325325e+00
  5.04632413e-01  6.96626082e-02 -4.84076947e-01 -6.93618238e-01
  2.18451515e-01 -3.01782936e-02  1.77319932e+00  7.37498820e-01
 -1.01685035e+00  5.78746140e-01 -7.53057301e-01  4.32285607e-01
```

Figure 6: Code snippet from changing word to vector

The word was changed to vector using the word2vec embedding model that was created, as shown in Figure 6. After getting the vector for the word, the average value of the vector (mean) is taken and multiplied (multiplied) by the frequency of words in the class and label. The following entire preprocess process is summarized below:

- Enter a sentence and repeat each word
- For each word, it will be changed to represent a numeric / vector.
- Take the mean vector and multiply with the number of classes and add them as features.
- Pad the sentence to fixed-length 128 then move to the next sentence.

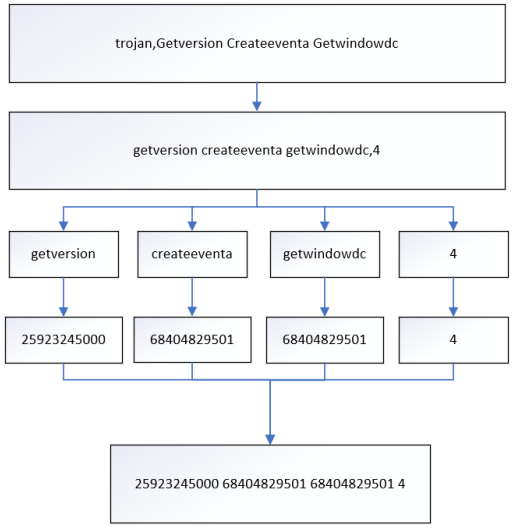

Figure 7: Illustration of preprocessing stages

After going through this process, as illustrated in figure 7, a fixed length of 128 vectors is obtained as a feature for each sentence. If a sentence has more than 128 words, the word will be truncated, and if it has less than 128 words, then padding "0" will be added so that each sentence has the same length.

After the feature making process, data mining classification is applied using a classification approach. We use Long Short-Term Memory (LSTM). Based on the type of API call chosen to describe a particular class of malware, the classification approach helps in concluding whether the file is malicious by determining the class in which the malware is. Because the process ends with the accuracy of determining the class in which the file is located after behavioral detection. All PEs have a direct relationship with the Operating System (OS) via the system calls API. This shows that API calls can easily notify malware behavior when attempting to execute.

4.3. Deep Learning Model

Deep Learning is one area of artificial neural networks to deal with problems on more large datasets. Deep Learning provides a very compelling architecture for supervised learning. By adding more layers to the deep learning model, it can do better at represent labeled malware data. To implement Deep Learning techniques for malware classification, a computer-based program is needed that can do computing. Therefore it is necessary to design an algorithm that can support the development of programs for this research. The algorithm used in this study is divided into three main parts, namely the training algorithm, the testing

algorithm, and the classification algorithm. These three algorithms follow the concept of writing code with API and the basic theory of Machine Learning for feature learning. In the training algorithm, five main stages will be carried out, namely the stages of Data Augmentation, Load Training Data, Modeling Long Short-Term Memory (LSTM), Training Model, and Final Weight Storage.

The LSTM model has several layers, including the embedding layer, LSTM layer, and Output layer. The input of the model is the preprocessing text that has been transformed into numeric where the input length is 128, where each number or vector represents a word, at the embedding layer, the input will be transformed into a vector that has a length of 128 vectors. Furthermore, LSTM consists of 3 gates, which will process each input vector to produce 128 vectors and where each output is connected to the output layer. At the output layer, there are seven neurons. each of these neurons has softmax activation to make a value in each classification. The classification prediction results are the highest output value.

The Nested LSTM model made consists of several layers, including the Embedding layer, Nested LSTM layer, and Output layer. Similarly, the LSTM input model of the Nested LSTM model is the preprocessing text that has been changed to numeric, where the length of the input is different where each number represents a word. At the embedding layer, the input will be transformed into a vector that has a length of 128. Furthermore, the Nested LSTM cell consisting of 3 gates (depth = 2) will process each input vector to produce 128 output vectors where each output is connected to the output layer. At the output layer, the same number neurons like LSTM and each of these neurons have softmax activation, which results in a value for each classification.

Support Vector Machine Model has three main parameters, namely C, Gamma, and Kernel. The kernel is always used as the Radial Base Function (RBF) because of its best performance. C and gamma are hyperparameters that have different values between the two and produce different accuracy and results. We need to find the best C and gamma values. That is why we use GridSearch. In GridSearch, we make all possible C and gamma combinations and then choose the one that has the best. Sklearn has a GridSearch Cross-Validation (CV) function that takes the SVM model, the Cs and Gammas grid parameters, and the number of folds. The number of folds means that the data will be divided into that many folds. In this case, it is three and then is trained on two and tested on one.

4.4. Evaluation Design

In this research, the dataset will be used as many as 13356 data, where the data is divided into three groups, namely training, validation, and testing. Data need to be converted in numerical value before going into the deep learning model. So the first step is to convert labels to one-hot encoding. After that, sentences are being converted into lowercase and remove punctuations to create a clean word2vec model using CBOW. The deep learning model will be trained using 8012 data, while the data testing stage will be tested using 2672 data. The LSTM model uses Adam optimizer using 64 batch sizes and 30 epochs because after several test we found that this combination works best on accuracy and added with 512 dense layers with 20% dropping units rate to prevent overfitting and also using softmax for classification. Whereas the Nested LSTM model uses Adam optimizer using 64 batch sizes and 50 epochs added with 1024, 2048, and 7 dense layers with recurrent dropping to prevent overfitting and also using softmax for classification. The Support vector machine uses the RBF kernel and Grid Search Cross-Validation for hyperparameter tuning to find the best value for the C parameter and gammas for the training model. In this research, performance will be measured based on the level of accuracy, recall, precision, and f1-score achieved to measure the performance, the results of the evaluation will be set forth in the form of a confusion matrix. The confusion matrix contains information from actual classifications and predicted classifications [31]. All methods were implemented on Python 3.6, Jupyter Notebook 6.0.3, Tensorflow 1.15.0 version, Intel Core i5-6400 (with 16 GB RAM), and Nvidia GeForce GTX 1050 Ti GPU.

5. Experimental Results

5.1. Training Result

After the two models are made, the model is trained using 13356 training data. After several trials, we decided to use 30 epochs with 64 batch sizes for the LSTM model and 50 epochs with 64 batch sizes for the nested LSTM model because they produce high accuracy in training.

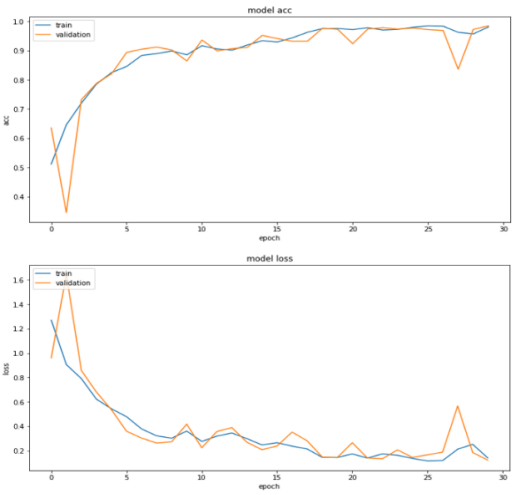

Figure 8: Accuracy and loss of the LSTM model

Figure 8 shows the training process of the LSTM model. The accuracy and loss of the model during training are indicated by the line above, the blue line shows the data in training, and the yellow line shows the validation data. The accuracy and loss of the LSTM model always increase from the start. At the end of the epoch, the accuracy reached 98%, and the loss was 0.14%.

Figure 9 shows the results of the training and validation process of the Nested LSTM model. The results of the accuracy and loss accuracy of this model can be seen in the line above, the blue line shows the data in training and the yellow line shows the validation data. the level of accuracy and loss in this model is quite good with an increase from the beginning to the end of the

test. At the end of the epoch, the accuracy reached 93.1%, and the loss was 0.18%.

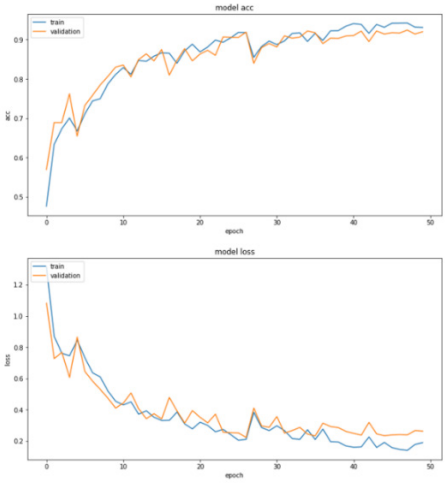

Figure 9: Accuracy and loss of the Nested LSTM model

5.2. Testing Result of LSTM Model

After the LSTM and Nested LSTM models are created and trained, the SVM is built and tested only as a benchmark to compare the two models above. All three models were tested using 2672 test data.

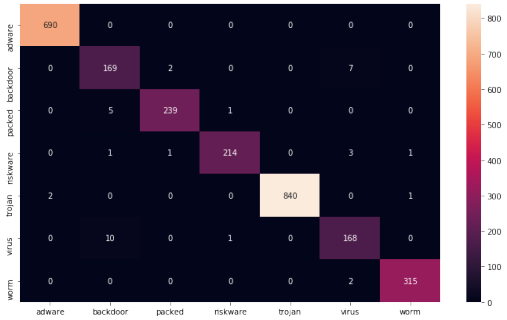

Figure 10: LSTM confusion matrix

Figure 10 shows the results of LSTM model testing, where the test results are shown using the confusion matrix. As we can see in every malware label, there is not much miss. This is because the loss rate of testing data is only 0.18%. The biggest label miss here is a virus where 30 labels are considered backdoor. The rest showed outstanding results with miss under 10. Thus, the testing accuracy obtained from the LSTM model is 98.61%.

Table 3: Details of the LSTM model results

Classification	Precision	Recall	F1-score
Adware	100%	100%	100%
Backdoor	95%	91%	91%
Packed	98%	99%	98%
Riskware	97%	99%	99%
Trojan	100%	100%	100%
Virus	94%	93%	95%
Worm	99%	99%	99%
Average	**97.57%**	**97.29%**	**97.43%**
Accuracy	**98.61%**		

Table 3 shows all the precision, recall, and f1-scores of the LSTM model from each malware. The LSTM method obtained an average precision of 97.57%, a recall of 97.29%, and an f1-score of 97.43%.

5.3. Testing Result of Nested LSTM Model

Figure 11 shows the results of Nested LSTM model testing, where the test results are shown using the confusion matrix. As we can see in every malware label, there is not much miss. It is because the loss rate of testing data is only 0.18%. The biggest label miss here is a backdoor where 66 labels are considered as a virus. The rest showed outstanding results with miss under 20. The testing accuracy obtained is 93.11%.

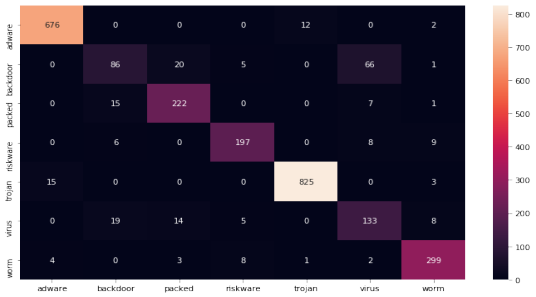

Figure 11: Nested LSTM confusion matrix

Table 4: Details of the Nested LSTM model results

Classification	Precision	Recall	F1-score
Adware	98%	100%	97%
Backdoor	55%	72%	67%
Packed	93%	71%	83%
Riskware	88%	87%	89%
Trojan	100%	99%	97%
Virus	65%	75%	76%
Worm	92%	92%	93%
Average	**84.43%**	**85.14%**	**86.00%**
Accuracy	**93.11%**		

Table 4 shows all the Precision, Recall, and F1-scores of the model from each classification. The Nested LSTM method obtained an average precision of 84.43%, a recall of 85.14%, and an f1-score of 86.00%.

5.4. Testing Result of Support Vector Machine Model

We also developed Support Vector Machine (SVM) so that it can be used as a comparison or benchmark. To see which ones perform better using the same word embedding method.

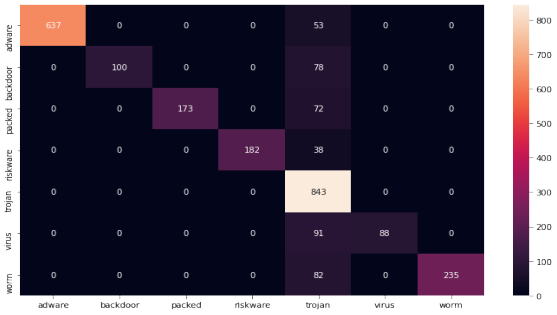

Figure 12: SVM confusion matrix

Figure 12 shows the results of the SVM testing model, where the test results are shown using the confusion matrix. As we can see in every malware label, there is much miss. For example, the Virus miscalculated up to 91 labels. Followed by almost every label more than 30 miss labels because of a low level of accuracy, and this is why we propose to use deep learning methods because this type of data is not suitable for SVM. The testing accuracy obtained is only 84.50%

Table 5: Details of the SVM model results

Classification	Precision	Recall	F1-score
Adware	94%	100%	97%
Backdoor	50%	100%	67%
Packed	70%	100%	83%
Riskware	85%	100%	92%
Trojan	100%	66%	80%
Virus	40%	100%	57%
Worm	78%	100%	87%
Average	**73.86%**	**95.14%**	**80.43%**
Accuracy	**84.50%**		

Table 5 shows all the results of the classification label, precision, recall, and f1-scores of the SVM model from each classification. The SVM method obtained an average precision of 73.86%, a recall of 95.14%, and an f1-score of 80.43%.

5.5. Summary of Testing Results

This section presented the result of all methods, including our proposed method, and compared it to other existing methods, which is SVM. Our proposed methods can overcome others. The results are shown in Table 6.

Table 6: Results comparison of all models

Method	Accuracy	Precision	Recall	F1-score
LSTM	**98.61%**	**97.57%**	**97.29%**	**97.43%**
Nested LSTM	93.11%	84.43%	85.14%	86.00%
SVM	84.50%	73.86%	95.14%	80.43%
CNN	96.60%	95.71%	95.12%	95.62%
DRN	86.54%	84.97%	84.23%	84.67%
DBN	92.60%	96.30%	89.60%	92.80%
DGCNN	96.87%	88.79%	92.83%	90.76%
DL4MD	95.64%	93.06%	94.60%	94.52%

Table 6 shows the overall comparison of all methods. We also make plot early stopping in LSTM and Nested LSTM train process so that we can take the best accuracy model when the training process happens. Overall, the table above shows that the LSTM model produces the best accuracy of 98.61% among the three methods, although the difference in accuracy does not differ significantly from the Nested LSTM. Both LSTM and Nested LSTM methods are still better than SVM methods. It shows that the deep learning method is far more accurate compared to ordinary machine learning methods.

6. Conclusion and Future Work

In this paper, we investigate the effectiveness of malware system call sequences that transformed into vectors and use word2vec as word embedding and then enter the LSTM layer repeatedly for the classification process with non-linear activation functions like Softmax. We have also carried out various experiments with different parameters, network structures, and added early stopping plots to get the best model accuracy in the training process. The design of the model is also evaluated using different methods such as Nested LSTM and SVM as benchmarks. From the three models, it can be concluded that the LSTM method gets the highest accuracy reaching 98.61% in the real-world data set. Overall, LSTM is included as the most effective approach to learning long-range dependencies in cybersecurity tasks and more appropriate methods for detecting malware through system call sequences.

From this research, it can be concluded that the experimental results with the LSTM network are straight forward. Still, we have not tried to use more complex LSTM networks, such as use many different layers, use more automatic decoders or use word embedding techniques other than word2vec. It because such a network architecture will cost us more and more complex preprocessing, network architecture, and a clean dataset probably will improve the results.

References

[1] N. Aziz, Z. Yunos, and R. Ahmad, "A management framework for developing a malware eradication and remediation system to mitigate cyberattacks," in *Lecture Notes in Electrical Engineering*, 481, 513–521, 2019.

[2] R. Bavishi, M. Pradel, and K. Sen, "Context2Name: A Deep Learning-Based Approach to Infer Natural Variable Names from Usage Contexts," 2018.

[3] C. Raghuraman, S. Suresh, S. Shivshankar, and R. Chapaneri, "Static and dynamic malware analysis using machine learning," in *Advances in Intelligent Systems and Computing*, 1045, 793–806, 2020.

[4] Abbasi, "Leveraging behavior-based rules for malware family classification," Dec. 2019.

[5] H. Lim, "Detecting Malicious Behaviors of Software through Analysis of API Sequence k-grams," *Comput. Sci. Inf. Technol.*, 4, no. 3, 85–91, 2016.

[6] Y. Ki, E. Kim, and H. K. Kim, "A novel approach to detect malware based on API call sequence analysis," *Int. J. Distrib. Sens. Networks*, 2015.

[7] V. Zenkov and J. Laska, "Dynamic data fusion using multi-input models for malware classification," 2019.

[8] M. Imran, M. T. Afzal, and M. A. Qadir, "Malware classification using dynamic features and Hidden Markov Model," in *Journal of Intelligent and Fuzzy Systems*, 31(2), 837–847, 2016.

[9] A. F. Agarap, "Towards Building an Intelligent Anti-Malware System: A Deep Learning Approach using Support Vector Machine (SVM) for Malware Classification," 2017.

[10] M. F. Rafique, M. Ali, A. S. Qureshi, A. Khan, and A. M. Mirza, "Malware Classification using Deep Learning based Feature Extraction and Wrapper based Feature Selection Technique," 1–20, 2019.

[11] Y. Lu, G. Jonathan, and L. Jiang, "Deep Learning Based Malware Classification Using Deep Residual Network," 2019.

[12] A. Oliveira, U. N. De Julho, and U. N. De Julho, "Behavioral Malware Detection Using Deep Graph Convolutional Neural Networks," 1–17, 2019.

[13] Z. Zhang, C. Chang, P. Han, and H. Zhang, "Packed malware variants detection using deep belief networks," *MATEC Web Conf.*, 309, 02002, 2020.

[14] K. HE and D.-S. KIM, "Malware Detection with Malware Images using Deep Learning Techniques," 2018.

[15] W. Hardy, L. Chen, S. Hou, Y. Ye, and X. Li, "DL4MD: A Deep Learning Framework for Intelligent Malware Detection," *Proc. Int. Conf. Data Min.*, 61–67, 2016.

[16] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations ofwords and phrases and their compositionality," *Adv. Neural Inf. Process. Syst.*, 3111-3119, 2013.

[17] D. Meyer, "How exactly does word2vec work ?," *Uoregon.Edu,Brocade.Com*, 1–18, 2016.

[18] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, 9, no. 8, 1735–1780, 1997.

[19] A. Sherstinsky, "Fundamentals of Recurrent Neural Network (RNN) and

Long Short-Term Memory (LSTM) network," *Phys. D Nonlinear Phenom.*, 404, p. 132306, Mar. 2020.
[20] F. Miedema, "Sentiment Analysis with Long Short-Term Memory networks," 1–17, 2018.
[21] J. R. A. Moniz and D. Krueger, "Nested LSTMs," *J. Mach. Learn. Res.*, 77, 530–544, 2017.
[22] Y. Ahuja and S. Kumar Yadav, "Multiclass Classification and Support Vector Machine," *Global Journal of Computer Science and Technology Interdisciplinary*, 12(11), 14–19, 2012.
[23] C. Brew, "Classifying ReachOut posts with a radial basis function SVM," 2016.
[24] L. Wang, B. Wang, J. Liu, Q. Miao, and J. Zhang, "Cuckoo-based malware dynamic analysis," *Int. J. Performability Eng.*, 15(3), 772–781, 2019.
[25] "VirusShare.com." [Online]. Available: https://virusshare.com/. [Accessed: 18-Apr-2020].
[26] G. D. Webster, Z. D. Hanif, A. L. P. Ludwig, T. K. Lengyel, A. Zarras, and C. Eckert, "SKALD: A scalable architecture for feature extraction, multi-user analysis, and real-time information sharing," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, 9866 LNCS, 231–249, 2016.
[27] T. K. Lengyel, S. Maresca, B. D. Payne, G. D. Webster, S. Vogl, and A. Kiayias, "Scalability, fidelity and stealth in the DRAKVUF dynamic malware analysis system," *ACM Int. Conf. Proceeding Ser.*, 2014-Decem, no. December, 386–395, 2014.
[28] S. Jamalpur, Y. S. Navya, P. Raja, G. Tagore, and G. R. K. Rao, "Dynamic Malware Analysis Using Cuckoo Sandbox," in *Proceedings of the International Conference on Inventive Communication and Computational Technologies, ICICCT 2018*, 2018, 1056–1060, 2018.
[29] "Kaspersky Cyber Security Solutions for Home & Business | Kaspersky." [Online]. Available: https://www.kaspersky.com/. [Accessed: 18-Apr-2020].
[30] "VirusTotal." [Online]. Available: https://www.virustotal.com/gui/home/upload. [Accessed: 26-Apr-2019].
[31] A. K. Santra and C. J. Christy, "Genetic Algorithm and Confusion Matrix for Document Clustering." 2012.