Tuijin Jishu/Journal of Propulsion Technology
ISSN: 1001-4055
Vol. 45 No. 2 (2024)

A RobustEmbedded Long-Short Term Memory (eLSTM) Learning Based Framework for Classification of Malware

[1] Lingaraj Sethi, [2] Prashanta Kumar Patra
[1] Biju Patnaik University of Technology, Rourkela, Odisha, India.
[2] SOA University, Bhubaneswar, Odisha, India.

Abstract: — Malware poses a significant threat to digital security, as it is designed to access or manipulate information from systems without user consent or authorization. As the digital landscape evolves, malware attacks by cyber-criminals are becoming increasingly sophisticated and frequent. To effectively counter these threats, this paper introduces a novel framework that combines Long-Short Term Memory (LSTM) and Convolutional Neural Networks (CNN) for malware detection and classification. The proposed framework leverages the strengths of LSTM and CNN neurons to process and learn from locally available data that exhibits malware-like characteristics. By abstracting and correlating relevant data, the model is capable of identifying patterns and anomalies associated with malware activity. This combination of LSTM's ability to capture temporal dependencies and CNN's feature extraction capabilities enhances the accuracy and robustness of malware detection. Through rigorous experimentation and evaluation, the framework demonstrates superior performance in classifying malware compared to traditional methods. The results suggest that this approach provides a powerful tool for identifying and mitigating potential malware threats, contributing to a safer and more secure digital environment.

Keywords: CNN, LSTM, Malware detection

1. Introduction

To detect and classify the malware is quite a cumbersome process. Malware is characterized by different characters and features making their grouping in different families of malware [Choudhary et al.]. To safeguard our system from this malware, various pattern matching techniques are there. But attackers had tried various metamorphism techniques to muddle these pattern matching techniques. This becomes a big challenge to handle these metamorphism techniques for detectors. There are various automatic malware generations technique that are producing new kinds of malware every second, if the previously generated malwares are detected by our conventional detecting system. Due to this problem, it is must to have modern day techniques that not only ranging to detect the malware attack on targeted system but analyze that malign data, to combat any future attack of evasion. Malware detection techniques are differently grouped. They are classified on the technique and character of malware used. Some of them are anomaly feature detection, signature and machine learning-based algorithm for malware detection [Tahir et al]. In signature-based technique, samples of malware are taken and a signature of it is obtained. This signature is used in the system and if our system found that signature then the malware will get detected. In the case of anomaly-based technique, look for the activities that fall outside the working range of the computer [Majumdar et al.]. The anomaly-based technique gives false detection many times as any variation in computer activities results in malware detection while in signature-based technique, a code metamorphism technique can easily confuse the system and malware can enter the system so to overcome these difficulties, the machine learning technique in evolved with time. To combat this evasion, traditional techniques are no longer effective. So, instead of traditional techniques, a specific technique that evolves from deep learning is used to solve this issue in cyberspace. Deep learning is a Machine Learning process that provides specific extraction

features i.e. high-level extraction from low-level data. This feature enables developers to get rid of featuring engineering parameters manually after analysis done by individuals. In this paper, an attempt has been made to develop a model to detect and classify the model that evolves from LSTM (Long Short-Term Memory) based deep learning technology.

1.1 Recurrent Neural Network

In Neural Networks that are feed on a forwarding basis, samples are treated independently of each other. Due to this reason, the feedforward neural network has no practical uses in the case of having samples related to each other or depending on each other. Thus, it requires a different type of architecture having memory, as required on time-sequential data or time series. RNN uses memory to the network (Mikolov et al.). Figure 1 illustrates the RNN. This RNN not only depends on the current input but requires information of previous time steps. That is indicated by the feedback loop:

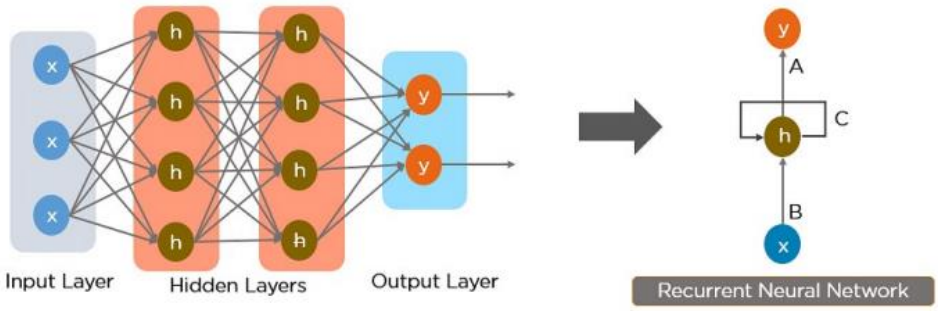

Fig. 1 Working Principle of RNN

While in the case of a feed-forward network, required set of information moves only in a forward direction, and information is available for each subsequent step of time steps in RNN. A Recurrent Neural Network (RNN) is a type of neural network designed to handle sequential data, such as time series, speech, or text. Unlike traditional feedforward neural networks, which process inputs independently, RNNs maintain an internal state that allows them to capture dependencies and patterns across time steps in a sequence.

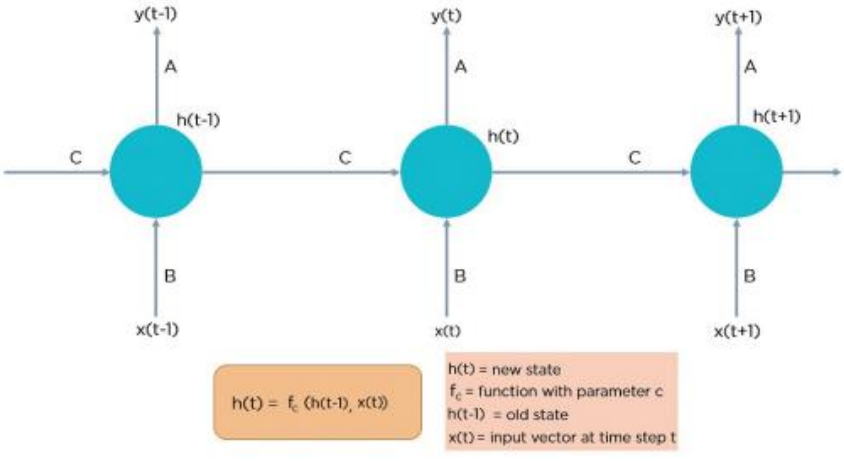

Fig. 2 Flow diagram of RNN

Here's an overview of the working principle of RNNs:

- Input Sequence: An RNN processes a sequence of data one time step at a time. The input at each time step can be a single element (e.g., a word in a sentence) or a vector representing multiple features.

- Recurrent Connections: At each time step, the RNN maintains a hidden state (also known as memory or context) that captures information from previous time steps. This hidden state is updated at each time step based on the current input and the previous hidden state.
- Hidden State Update: The hidden state at each time step is computed using a combination of the input and the previous hidden state. Mathematically, this can be represented as:

$$h_t = \sigma\left(W_h h_{t-1} + W_x x_{t+b}\right) h_t$$

Where:

ht is the hidden state at time step t

ht−1 is the hidden state from the previous time step. o xt is the input at time step t.

Wh, Wx, and bb are learned parameters (weights and biases).

σ is a non-linear activation function, such as tanh or ReLU.

- • Output: The RNN can produce an output at each time step based on the hidden state. This output may be a prediction, a classification, or some other form of result depending on the task. Mathematically, the output can be represented as:

$$y_t = \text{softmax}\left(W_y h_t + b_y\right) y_t$$

Where:

yt is the output at time step t.

Wy and by are learned parameters.

The softmax function is used for classification tasks.

- Training: RNNs are trained using backpropagation through time (BPTT), an extension of the standard backpropagation algorithm. This process involves computing the gradients of the loss function with respect to the model's parameters and updating them using an optimization algorithm like stochastic gradient descent.
- Challenges: While RNNs are powerful for handling sequential data, they face challenges such as vanishing and exploding gradients, which can affect learning and lead to poor performance on long sequences. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) architectures are variations of RNNs that address these issues and improve the network's ability to capture long-term dependencies. Overall, RNNs are a versatile class of neural networks that excel at processing and learning from sequences of data, making them well-suited for tasks such as natural language processing, speech recognition, and time series forecasting.

a. Long Short-Term Memory

The basic Recurrent Neural Network (RNN) model is known to suffer from the vanishing gradient problem during backpropagation, which can hinder the network's ability to learn long-range dependencies and lead to significant memory utilization. To address these challenges, more complex RNN models with specific gating mechanisms have been developed. Among these, the Long Short-Term Memory (LSTM) model is the most renowned gated architecture. LSTM mitigates the vanishing gradient problem by decoupling the network's memory from its output, allowing for better handling of long-term dependencies. Additionally, LSTM employs additive updates to the memory state, rather than multiplicative ones, thereby providing more stable learning dynamics. Bidirectional LSTMs are an extension of the standard LSTM model that processes data in both forward and backward directions. This bidirectional processing enables the network to capture context from both past and future inputs, enhancing its performance on various sequential data tasks. Overall, these advanced RNN models provide more robust solutions for learning from sequential data, offering improved performance and efficiency.

b. Conventional Neural Network (CNN)

Overview: Convolutional Neural Networks (CNNs) are a type of deep learning model designed to efficiently handle data with a local structure, such as images. CNNs have been particularly successful in image classification and other computer vision tasks.

- Convolutional Layers: CNNs contain convolutional layers, which apply a series of filters to the input data. These filters detect specific features, such as edges or patterns, within the data.
- Deeper Layers: As the network goes deeper, the layers detect more complex and abstract features.
- Pooling Layers: Pooling layers (e.g., max pooling) reduce the dimensionality of the data and the computational load, which helps prevent overfitting and improves generalization.

1.4 TensorFlow Layers

- Overview: TensorFlow is a popular deep learning framework that provides various tools and layers to build neural networks.
- Layer Types: Different layer types in TensorFlow, such as Dense, Dropout, LSTM, and Embedding layers, allow for creating diverse and complex neural network architectures.

1.5 Input Layers

Purpose: The input layer is the starting point of a neural network, where the initial data enters the network.

1.6 Dropout Layer

- Function: Dropout is a regularization technique that randomly disables a proportion of neuron connections during training.
- Purpose: This technique helps reduce overfitting and improves the generalization of the model.

1.7 LSTM Layer

- Overview: The Long Short-Term Memory (LSTM) layer is a recurrent neural network (RNN) variant designed to capture long-term dependencies in sequential data.
- Function: LSTM layers handle forward and backward propagation in a single layer, making them efficient for tasks such as time series analysis and natural language processing.

1.8 Bidirectional Layer

- • Purpose: Bidirectional layers allow RNNs to process data in both forward and backward directions.
- • Function: This bidirectional processing improves the network's ability to capture context from both past and future inputs in sequential data tasks.

1.9 Dense Layer

- Function: Dense layers, also known as fully connected layers, connect each neuron to every neuron in the previous and next layer.
- Purpose: These layers perform complex transformations and are commonly used in neural networks.

1.10 Embedding Layer

- Function: Embedding layers map positive integer indices (e.g., words in a vocabulary) to dense vectors of floating-point values.
- Purpose: This layer is widely used in natural language processing tasks to convert discrete data (e.g., words) into dense representations.

1.11 CONV 1D Layer

- Overview: CONV 1D layers apply one-dimensional convolutional filters to sequential data.

- Purpose: This layer is useful for tasks such as processing time series data or natural language processing, where data can be represented as sequences.

2. Objectives

The primary objective of the research is to Model hyperparameter tuning reveals best-practice parameters, and the ensemble confusion matrix delves into classification efficacy.

Deep learning is evolved from the traditional Machine Learning process. It has attracted the attention of researchers as it is a hotspot in the cyber safety industry. Deep learning has achieved many recognized applications in the industry. Athiwaratkun et al. have reviewed various models of malware classifications. In these models, one model is found to have two-stage classifiers. In this model, the initial stage layer is either LSTM or GRU and the one stage is single layer MLP. Another model found in his review has a single stage of nine CNN layers. When these models are trained evaluated systematically then it has an accuracy of up to 80%. Zhang et al. in their work have adopted a novel deep learning model that contains both CNN and LSTM layers. For training these models, an API call sequence is being used. The CNN of this model has filters of increasing size, with an output of each filter is working as input to the LSTM layer. The output of this LSTM layer is used as input for the dropout layer with final fully connected layers for classifications. The outcome obtained from the dense layer is the required prediction of the given model. This model has an accuracy of about 100%. Mishra et al. Use the BiLSTM model for classifying the various malware in a given cloud system. It has CNN layers and it is trained on-call sequence. The authors have achieved an accuracy of up to the 90%. They also found that putting BiLSTM on single-layer LSTM results in the worst accuracies in all possible cases studied by them. Lu et al. have proposed a work that uses opcodes obtained by disassembled executables. They also used the technique of embedding the word as a feature engineering step. Natural Language Processing applications also deploy techniques of word embedding. The result obtained from word embedding is used as input for LSTM layers. For detection purposes, this model has achieved an AUC of 0.99 while in the case of classification AUS is 0.987. A brief comparison of various malware detection techniques based on opcodes has been discussed in table 1. In table 1, the various author used different kinds of algorithms on windows and IoT. The result obtained in terms of accuracy, recall % and Precision & has been discussed below.

Table 1. Comparative study of various op-code based techniques

Author's/ Algorithm	Dataset used on	Accuracy in %	Recall in %	Precision in %
Santos et al.	Windows	95.91	77.70	86.25
LSTM+CNN+MF [Yan et al.]	Windows	99.88	99.14	-
Hashemi et al.	Windows	96.87	81.55	91.09
Azmoodeh et al.	ARM based on IOTs.	91.91	94.20	83.33

These findings show a range of performance across different algorithms and datasets:

- Santos et al. achieved a high accuracy of 95.91% on a Windows dataset, with recall and precision rates of 77.70% and 86.25% respectively.
- LSTM+CNN+MF [Yan et al.] showed exceptional performance with an accuracy of 99.88% and a recall rate of 99.14% on a Windows dataset. Precision data is not specified.
- Hashemi et al. obtained an accuracy of 96.87% with recall and precision rates of 81.55% and 91.09% respectively on a Windows dataset.
- Azmoodeh et al. worked with ARM-based IoT datasets and achieved an accuracy of 91.91%, a high recall rate of 94.20%, and a precision rate of 83.33%.

Overall, these findings highlight the varying performance of different algorithms and approaches across different types of datasets, illustrating the potential for further improvements in malware detection methods.

3. Methods

The Embedded Long Short-Term Memory (LSTM) model is used to classify malware based on data from the works of Prajapati et al. and Nappa et al., encompassing data from 20 families of malware. Deep learning models often require frequent tuning of data, and in this approach, the embedded and dense layers are randomly initialized at each instance of training. This results in variations in accuracy each time the model is trained. The embedded layer is positioned between the input layer and the LSTM layer in the architecture, as depicted in figure 3.

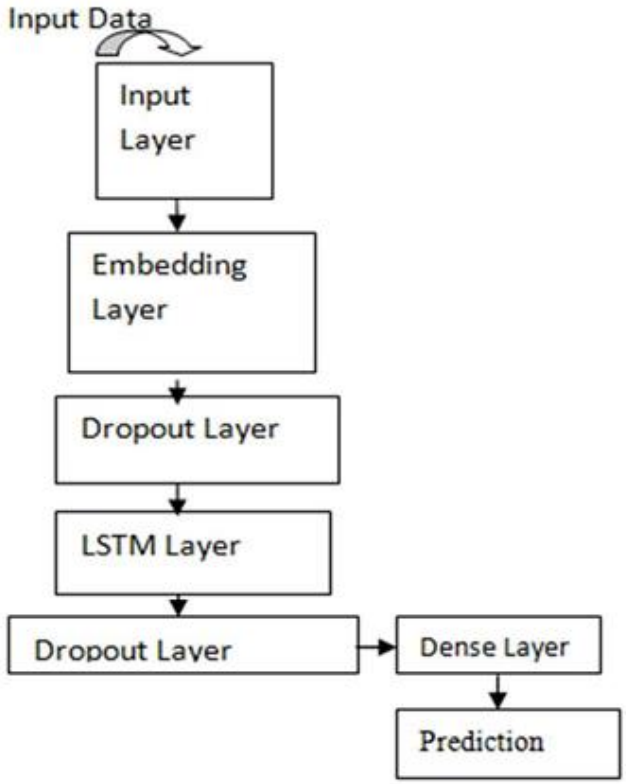

Fig.3 Architecture of Embedded LSTM Model

The methodology can be implemented with the following set of algorithms:

Algorithm 1: Predict

Steps:

1. Import necessary libraries.

2. Load and visualize the dataset.

3. Preprocess the dataset.

4. Prepare the dataset for LSTM.

5. Build and train the LSTM model.

6. Make predictions and evaluat the model

The above algorithm uses LSTM for prediction using loading, preprocessing, building and training the dataset using a given training dataset.

Algorithm 2. CNNLSTM

Steps:

1. Input training_x, training_y

2. Initialize hyper parameters

3. Normalize inputs using Max-Min normalization

4. Apply dropout with dropout rate

5. Apply LSTM with specified epochs and batchsize

6. Apply dopout with same dropout rate as in step 4

7. Apply neural network with linear activation function

8. Call Predict This algorithm uses our proposed model according to the various set of steps used and calls Algorithm 1 for prediction.

4. Simulation & Results

The proposed work have used X-IIoTID and UNSW-NBI5 dataset used by multiple researchers in the field.

Table-2: Features of the UNSW-NB15 dataset.

Name of Feature	Feature Value	Feature type
Service	nominal	primary
State	nominal	primary
Sinpkt	float	time
Dpkts	int	primary
Dbytes	int	primary
Synack	float	time
Dttl	int	primary
Dload	float	primary
Ct_dst_src_ltm	int	connection
Sload	float	primary
Ct_src_dport_ltm	int	connection
Dloss	int	primary
Sloss	int	primary
Dur	float	primary

4.1 Performance Measures

Different performance measurement metrices like precision, recall, F-Score and accuracy were used in the work. FP represents number of traffic, TP represents number of attacks, TN measures number of traffic measured successfully. FN represents the number of attacks misclassified as normal traffic.

$$Accuracy = (TP + TN) / (TP + TN + FP + FN) \quad (1)$$

$$Precision = TP / (TP + FP) \quad (2)$$

$$Recall = TP / (TP + FN) \quad (3)$$

$$F1\ Score = 2 * (Precision * Recall) / (Precision + Recall) \quad (4)$$

4.2 Results

The following results were obtained as a part of binary amd multi-clss classified for LSTM model, CNN model and our proposed CNN+LSTM model.

Table 3: Results using CNN model

Metrics	Binary Classifier	Multi-class Classifier
Training accuracy	91.22%	91.12%
Validation accuracy	91.15%	91.09%
Testing accuracy	91.14%	91.10%
Training loss	41.83%	11.83%
Loss in Validation	11.72%	11.70%
Loss in Test	6.41%	6.41%

Analysis of table 3 represents:

- Accuracy Metrics: The CNN model exhibits consistent accuracy across training, validation, and testing sets for both binary and multi-class classification tasks.
- Loss Metrics: The CNN model's training loss is much higher for the binary classifier compared to the multi-class classifier. However, the loss metrics show significant improvement in the validation and testing stages, particularly in binary classification. This suggests that the model is overfitting during training but performs better with unseen data.
- Performance: The model generally performs well, with consistent accuracy and loss across different datasets, particularly in testing.

While the CNN model offers strong performance, particularly in terms of testing accuracy and loss, the high training loss in binary classification suggests there may be room for further optimization or regularization techniques to reduce overfitting during training.

Table 4: Results using with LSTM model

Metrics	Binary Classifier	Multi-class Classifier
Training accuracy	90.53%	90.46%
Validation accuracy	90.05%	90.08%
Testing accuracy	90.19%	90.08%
Training loss	12.75%	12.75%
Loss in Validation	12.71%	12.72%
Loss in Test	8.32%	8.33%

Analysis from table 4 illustrates:

- • The LSTM model shows consistent accuracy across training, validation, and testing datasets for both binary and multi-class classification tasks.
- • The training, validation, and testing accuracies are relatively close, indicating the model generalizes well and avoids overfitting.
- • Loss metrics suggest that while the training loss is quite high, the testing loss is much lower, indicating that the model performs better on new, unseen data.
- • The LSTM model performs well overall, though there may be opportunities for further improvements, particularly in reducing training loss.

Table 5: Results using with CNN+LSTM model

Metrics	Binary Classifier	Multi-class Classifier
Training accuracy	93.74%	93.24%
Validation accuracy	93.11%	93.11%
Testing accuracy	93.21%	92.90%
Training loss	5.19%	6.07%
Loss in Validation	5.89%	5.98%
Loss in Test	6.21%	6.28%

Analysis of table 5 presents:

- The model shows consistently high accuracy across training, validation, and testing datasets for both binary and multi-class classification tasks.
- The loss metrics are slightly higher for testing datasets compared to training and validation, suggesting some overfitting or slight generalization issues.
- Overall, the model performs well, with high accuracy and low loss, making it suitable for the tasks at hand.

Table 6: Analysis of Performance by comparing proposed models with exsting research works on UNSW-NB15 dataset.

Paper	Model	Binary Classification	Multi-class Classification
12	DM	80.72%	72.26%
5	GE	86.64%	78.32%
15	DNN	87.62%	85.38%
2	DEL	90.90%	88.90%
Proposed model	CNN	90.09%	90.10%
Proposed Model	LSTM	97.14%	97.10%
Proposed Model	CNN + MTM	93.28%	92.90%

The study shown in table 6, presents an analysis of performance by comparing proposed models with existing research works on the UNSW-NB15 dataset. This dataset is commonly used for cybersecurity research, particularly for intrusion detection systems. Various models were evaluated in both binary and multi-class classification tasks. The existing research works compared include DM, GE, DNN, and DEL models, with the following classification results:

- DM achieved an accuracy of 80.72% for binary classification and 72.26% for multi-class classification.
- GE attained 86.64% and 78.32% for binary and multi-class classification, respectively.
- DNN reached 87.62% for binary classification and 85.38% for multi-class classification.
- DEL performed at 90.90% for binary classification and 88.90% for multi-class classification. The proposed models introduced in the study include:
- CNN, which achieved an accuracy of 90.09% in binary classification and 90.10% in multi-class classification.

- LSTM, which outperformed other models, achieving 97.14% and 97.10% accuracy in binary and multi-class classification, respectively.
- CNN + MTM, which achieved an accuracy of 93.28% in binary classification and 92.90% in multi-class classification. The results demonstrate that the proposed LSTM model significantly outperforms both existing models and other proposed models in both binary and multi-class classification tasks. These findings highlight the effectiveness and potential of advanced deep learning techniques in cybersecurity applications.

5. Discussion

This work presents an innovative approach to enhancing the Long Short-Term Memory (LSTM) model by integrating an embedding layer. The model employs a hybrid CNN-LSTM technique, implemented using TensorFlow, and is tested on a standardized system. By incorporating the embedding layer, this approach overcomes key challenges associated with manual data feature engineering. The model leverages the capabilities of CNN to extract spatially local correlations and the strengths of LSTM to learn from API calls, resulting in improved accuracy in detection classification tasks. Further evaluation can be conducted by constructing alternative models, including LSTM without embedding, Bidirectional LSTM (BiLSTM) with embedding, and BiLSTM without embedding. This study demonstrates the potential for combining CNN and LSTM in a unified model to achieve superior performance in complex classification scenarios.

Refrences

[1] B. Athiwaratkun and J. W. Stokes (2017). Malware classif ication with LSTM and GRU language models and a character-level CNN. In 2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, pages 2482–2486.

[2] A. Azmoodeh , A. Dehghantanha , and Choo K.-K. R., “Robust Malware Detection for Internet Of (Battlefield) Things Devices Using Deep Eigenspace Learning,” IEEE Trans. Sustain. Comput., 2018, doi: 10.1109/TSUSC.2018.2809665

[3] J. Cheng, L. Dong, and M. Lapata, (2016). Long shortterm memory-networks for machine reading.< https: //arxiv.org/abs/1601.06733>.

[4] S. Choudhary, and A. Sharma, (2020). Malware detection &classification using machine learning. In 2020 International Conference on Emerging Trends in Communication, Control and Computing, ICONC3, pages 1–4.

[5] H. Hashemi, A. Azmoodeh, A. Hamzeh, and S. Hashemi, “Graph embedding as a new approach for unknown malware detection,” J. Comput. Virol. Hacking Tech., 2017.

[6] R. Lu, (2019). Malware detection with LSTM using opcode language.

[7] T. Mikolov, S. Kombrink, L. Burget, J. ˇCernock`y, , and S. Khudanpur (2011). Extensions of recurrent neural network language model. In 2011 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, pages 5528–5531.

[8] P. Mishra, K. Khurana,S. Gupta, and M.K. Sharma, (2019). Vmanalyzer: Malware semantic analysis using integrated CNN and bi-directional LSTM for detecting VM-level attacks in cloud. In 2019 Twelfth International Conference on Contemporary Computing, IC3, pages 1–6.

[9] A. Mujumdar, G Masiwal, and D.B. Meshram (2013). Analysis of signature-based and behavior-based antimalware approaches. International Journal of Advanced Research in Computer Engineering and Technology, 2(6).

[10] A. Nappa, M.Z. Rafique, and J. Caballero, (2015). The MALICIA dataset: Identification and analysis of drive-by download operations. International Journal of Information Security, 14(1):15–33.

[11] P. Prajapati, and M. Stamp, (2021). An empirical analysis of image-based learning techniques for malware classification. In Stamp, M., Alazab, M., and Shalaginov, A., editors, Malware Analysis using Artificial Intelligence and Deep Learning. Springer.

[12] I. Santos, F. Brezo, XUgarte-Pedrero, and P.G. Bringas, “Opcode sequences as representation of executables for data mining-based unknown malware detection,” Inf. Sci. (Ny)., 2013.

[13] R. Tahir, (2018). A study on malware and malware detection techniques. International Journal of Education and Management Engineering, 8(2):20–30.
[14] N. Tavakoli, (2019). Modeling genome data using bidirectional LSTM. In 2019 IEEE 43rd Annual Computer Software and Applications Conference, volume 2 of COMPSAC, pages 183–188. IEEE.
[15] J. Yan , Y. Qi , and Q. Rao, "Detecting Malware with an Ensemble Method Based on Deep Neural Network," Secur. Commun. Networks, vol. 2018, pp. 1–16, 2018.
[16] J. Zhang, (2020). Deepmal: A CNN-LSTM model for malware detection based on dynamic semantic behaviors. In 2020 International Conference on Computer Information and Big Data Applications, CIBDA, pages 313–316.