

Neural Network Model for Malware Classification Based on BiD-ConvLSTM Encoder

Junyu Wu¹[0009-0000-2249-3676], Lilie Chen¹[0009-0003-4573-1560], and Yuan Liu¹[0000-0002-2576-1426](✉)

¹ School of Artificial Intelligence and Computer Science, Jiangnan University, No. 1800 Lihu Avenue, Wuxi, 214122, Jiangsu Province, China.

lyuan1800@jiangnan.edu.cn

Abstract. Malware poses a pervasive threat to computer systems and data security. Previous work has demonstrated that deep learning approaches are effective solutions to this issue. Many scholars have adopted a method of transforming the binary codes of executable malware files into images, using neural networks for classification. However, neural networks typically require inputs of fixed dimensions, while the sizes of malware files can vary significantly. Traditional methods of standardizing image sizes can lead to loss and redundancy of information. This paper proposes a malware classification model using a Bidirectional Dynamic Convolutional Long Short-Term Memory (BiD-ConvLSTM) encoder, which encodes images generated from binary codes of varying-sized malware, producing fixed-size feature images for neural network training. Images output by the encoder are trained using ResNet-50. This model achieved a maximum accuracy of 98.44% on the dataset from the Kaggle Microsoft Malware Classification Challenge (BIG 2015).

Keywords: Malware Detection, Malware Classification, Deep Learning, Neural Networks, BiD-ConvLSTM.

1 Introduction

With the exponential growth of the Internet, the prevalence of malware has escalated dramatically, establishing it as a formidable threat to cybersecurity. The spectrum of malware encompasses Trojans, worms, viruses, botnets, ransomware, adware, and spyware, all of which exploit vulnerabilities in personal computers and network systems [1]. Initially, malware was composed of simplistic code structures, which were relatively easy to detect. However, as malware developers have progressively increased the sophistication of their coding techniques, it has become increasingly difficult for even advanced detection methods to identify these threats. Malware creators often reuse segments of code to generate variants that exhibit similar characteristics, which are typically categorized under the same malware family. Identifying these malware families is critical, as it enhances the effectiveness of malware prevention strategies [2].

Malware detection can be broadly categorized into two methods: static and dynamic. In the early days, the industry commonly employed signature-based matching to identify malware. Although this method was fast, it could not identify unknown malware. Modern static detection methods have shifted towards analyzing statistical features, such as API function calls, to improve detection rates [3]. Conversely, dynamic detection involves analyzing the behavior of malware within controlled virtual environments [4]. Although more effective at identifying anomalous behavior, dynamic analysis is time-intensive and less suited for real-time detection tasks. With the rapid development of deep learning in the field of image recognition and classification, prompting researchers to apply these techniques to malware classification. This approach involves transforming malicious code into standardized image formats for deep learning classification, achieving substantial accuracy improvements. Notably, the Microsoft malware dataset featured in the 2015 Kaggle competition illustrates the potential of these methodologies [5]. However, since malware does not have a fixed size, converting malware into images of fixed sizes inevitably requires cropping or expanding the original data, leading to information loss and redundancy.

In response to these limitations, this paper proposes a malware classification model that incorporates a Bidirectional Dynamic Convolutional Long Short-Term Memory (BiD-ConvLSTM) encoder. This novel encoder adapts to varying sizes of malware, converting binary codes into uniformly sized images that preserve crucial information for neural network training. By merging convolutional and BiLSTM layers [6], the encoder effectively harnesses the contextual semantic features inherent in binary files, thereby enhancing the classification accuracy while mitigating the issues associated with traditional feature extraction techniques.

2 Related work

Visualizing malware provides a more intuitive way to observe its texture, which can reduce the impact of obfuscation techniques. Nataraj et al. [7] proposed converting malware binary files into grayscale images and then classifying them using machine learning methods. They first converted each 8 bits of a malware binary file into a decimal, corresponding to pixel values ranging from 0 to 255, and then fixed the image size based on the file size to generate grayscale images that reflect the characteristics of the malware in image features. Inspired by Nataraj et al. [7], Han et al. [8] proposed a novel method to convert malware binary files into RGB image matrices. In their approach, they first disassembled the malware binary files, then extracted and stored the opcode sequences in blocks, and each block was processed by two hash functions (Simhash and djb2) to generate the coordinates and RGB pixel information for the image matrix.

Due to the significant advantages demonstrated by deep convolutional neural networks such as VGGNet, ResNet, and DenseNet in large-scale image classification tasks [9], the classification of malware images has increasingly been conducted using deep learning. ResNet and VGG16 networks proposed by Rezaei et al. [10,11] have been applied to malicious code detection and classification, improving accuracy. Yuan et al. [12] proposed a byte-level malware classification method based on Markov images and

deep learning. The main steps involve converting malware binary files into Markov images based on byte transition probability matrices, and then using deep convolutional neural networks for Markov image classification. Shen et al. [2] introduced a feature fusion-based malicious code detection method that includes dual attention mechanisms and Bidirectional Long Short-Term Memory (BiLSTM).

Since training deep neural networks requires consistent data sizes, most of the learning-based approaches mentioned above convert binary files into fixed-size images. However, existing methods often overlook the loss of some byte information during the conversion process. Additionally, there is a significant amount of redundant information in the conversion process.

3 Proposed Method

The malware classification model proposed in this paper is structured into two main parts: an encoder based on Bidirectional Dynamic Convolutional Long Short-Term Memory (BiD-ConvLSTM) and a classifier composed of the ResNet network [13,14].

3.1 Encoder

To effectively tackle the challenge of encoding variable-sized malware files into consistent, fixed-size image representations suitable for deep learning analysis, this paper introduces an innovative encoder design: the Bidirectional Dynamic Convolutional Long Short-Term Memory (BiD-ConvLSTM) encoder. As depicted in Figure \ref{figure1}, this encoder is adept at handling the dynamic and diverse nature of malware binaries. The process begins by transforming binary files of malware into a sequence of images, each images maintaining fixed dimensions and serving as a time step input to the BiD-ConvLSTM encoder. This transformation is crucial as it ensures that each segment of the binary data is uniformly represented, facilitating more reliable processing.

Within the BiD-ConvLSTM, feature extraction techniques and attention mechanisms are employed. These methods are critical for adjusting the scale and dimensionality of the data, ensuring that each output feature image adheres to a predefined size, regardless of the original input dimensions. This standardization is pivotal, as it guarantees that the output from the encoder is a set of feature-rich, fixed-size images, perfectly suited for further classification tasks. Consequently, the encoder's output can seamlessly integrate with various types of classifiers, including deep neural networks like ResNet or VGG, which require consistent input sizes for optimal performance. By providing standardized, high-quality feature images, the BiD-ConvLSTM encoder plays a crucial role in enhancing the overall effectiveness and accuracy of malware classification systems. This approach not only simplifies subsequent processing stages but also maximizes the potential for accurate malware detection and classification across diverse datasets.

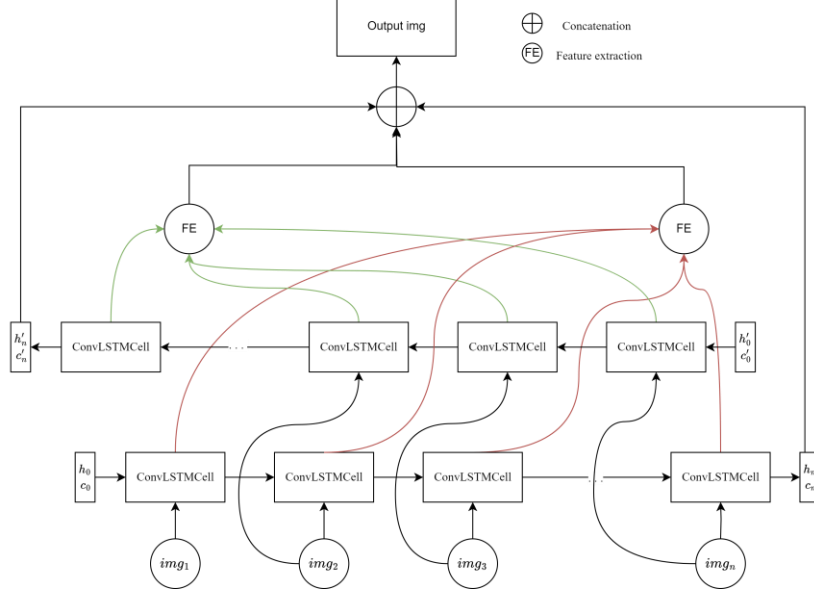


Fig. 1. BiD-ConvLSTM encoder

ConvLSTM. The ConvLSTMCell, as proposed by Shi [6], replaces the fully connected layers in the FC-LSTM cell with convolution operations, thus substituting matrix multiplication with convolutions. LSTM, being a specialized type of RNN, has been proven to be stable and effective in modeling long-range dependencies across various studies [15,16]. FC-LSTM can be regarded as a multivariate version of LSTM, where the inputs, outputs, and states are all 1D vectors. The essential operations of FC-LSTM, as outlined in the work of Graves [17], are detailed in equation (1), where σ is the sigmoid function, i , f , o , and c denote the input gate, forget gate, output gate, and memory cell, respectively, and \circ indicates the Hadamard product.

$$\begin{aligned}
 i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci} \circ c_{t-1} + b_i) \\
 f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf} \circ c_{t-1} + b_f) \\
 c_t &= f_t \circ c_{t-1} + i_t \circ \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
 o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co} \circ c_t + b_o) \\
 h_t &= o_t \circ \tanh(c_t)
 \end{aligned} \tag{1}$$

The primary limitation of FC-LSTM in handling spatiotemporal data is its reliance on fully connected layers for the transitions from input-to-state and state-to-state, which fail to encode spatial information. This means FC-LSTM cannot adequately account for the significance of the same opcode appearing at different positions within malware. To address this issue, we employ the ConvLSTM configuration proposed by Shi [6]. The fundamental operations of the ConvLSTMCell are displayed in equation (2), with $*$ denoting the convolution operation. This modification transforms the input from a 1D

sequence to a 2D matrix, while other parameters remain consistent with those of the FC-LSTM.

$$\begin{aligned}
i_t &= \sigma(W_{xi} * X_t + W_{hi} * H_{t-1} + W_{ci} \circ C_{t-1} + b_i) \\
f_t &= \sigma(W_{xf} * X_t + W_{hf} * H_{t-1} + W_{cf} \circ C_{t-1} + b_f) \\
c_t &= f_t \circ C_{t-1} + i_t \circ \tanh(W_{xc} * X_t + W_{hc} * H_{t-1} + b_c) \\
c_t &= f_t \circ C_{t-1} + i_t \circ \tanh(W_{xc} * X_t + W_{hc} * H_{t-1} + b_c) \\
o_t &= \sigma(W_{xo} * X_t + W_{ho} * H_{t-1} + W_{co} \circ C_t + b_o) \\
h_t &= o_t \circ \tanh(C_t)
\end{aligned} \tag{2}$$

Feature extraction. Due to the varying sizes of malware, ConvLSTM encounters different lengths of time steps when processing input data. To harness the power of deep learning techniques and better utilize the inherent characteristics of malware for effective family classification, this paper employs feature extraction and attention mechanisms. These methodologies are critical for standardizing the dimensions of the output data, ensuring that the neural network can interpret and process the information consistently, regardless of the input size variability.

In our approach, we focus on capturing the essential statistical features from each time step of the ConvLSTM output. Specifically, we extract the maximum and mean values across each time step H_t . These values are crucial as they represent the peak and average activations within the time step, providing a snapshot of the most and generally active features respectively. The maximum value highlights areas with the highest activation, suggesting features of significant importance, while the mean value offers a sense of the overall activity level, helping to understand the background behavior of the data. The extraction and retention of these values are formalized in equation (3), where we compute and output the maximum and mean values for each time step:

$$\begin{aligned}
H_{Max[i,j]} &= \max(H_{t[i,j]}, H_{Max[i,j]}) \\
H_{avg} &= \frac{\sum H_t}{t}
\end{aligned} \tag{3}$$

The outputs H_n along with the outputs from the feature extraction module, H_{Max} and H_{avg} , are concatenated to form the output of the unidirectional dynamic ConvLSTM encoder. Executing the process in the reverse direction yields the backward encoder output, and concatenating these outputs allows the BiD-ConvLSTM to encode the variable-length matrix sequence.

This paper also replaces the above feature extraction module with an attention mechanism, as shown in Figure 2, directly concatenating the bidirectional H_n as the encoder output. The Spatial Attention Module (SAM) [21] used here primarily focuses on introducing attention to the spatial positions of input data within deep learning models. The core idea of SAM is to dynamically allocate attention across different regions of the input, selectively emphasizing relevant features and suppressing less useful ones. This is achieved by processing the input feature maps through average pooling and max pooling operations along the channel axis. The average pooling captures the average

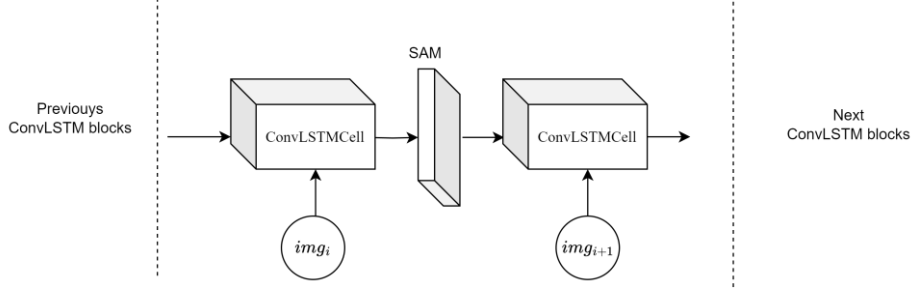


Fig. 2. Use SAM's BiD-ConvLSTM encoder

presence of features, providing a global context, while max pooling highlights the most dominant features at specific locations. These pooled features are concatenated to form a comprehensive feature descriptor. To compute spatial attention, average pooling and max pooling operations are first applied along the channel axis and then concatenated to create an effective feature descriptor. Subsequently, a convolutional layer is applied to this concatenated descriptor to generate a spatial attention map $M_s(F) \in \mathbb{R}^{H \times W}$, as calculated in equation (4):

$$\begin{aligned} M_s(F) &= \sigma(f^{7 \times 7}([AvgPool(F), MaxPool(F)])) \\ &= \sigma(f^{7 \times 7}([F_{avg}^s, F_{max}^s])) \end{aligned} \quad (4)$$

where σ represents the sigmoid function, and $f^{7 \times 7}$ indicates a convolution operation with a 7×7 filter. The map indicates which spatial locations the model should focus on or suppress. Finally, this spatial attention map is multiplied by the original feature map to produce a weighted feature map, which allows the model to focus more on regions containing significant information in subsequent processing.

3.2 Classifier

In this study, we utilize the Deep Residual Network (ResNet), originally proposed by He et al. [8], along with its subsequent improvements detailed in [9], to serve as the classifier. ResNet has established itself as a series of extremely deep architectures that exhibit compelling accuracy and robust convergence behavior. The cornerstone of ResNet is the introduction of the concept of residual learning. Each residual block within ResNet does not output direct feature mappings. Instead, these blocks output the residual of their inputs—essentially the difference or discrepancy between the input and the output. This architectural design requires the network to learn this residual mapping to an identity function, rather than learning a straightforward non-linear mapping.

This innovative approach makes it significantly easier for the network to learn identity mappings, particularly beneficial when the network depth increases. By learning these residuals, the network can enhance its ability to propagate gradients throughout its depth, effectively avoiding the performance degradation typically associated with increased depth in traditional architectures.

For our classification tasks, we employ the ResNet50 model, a specific variant of ResNet. The detailed architecture of ResNet50 is outlined in Table 1. This model con-

Table 1. Architectures for ResNet50.

layer name	conv1	conv2	conv3	conv4	conv5	output
output size	$\frac{W}{2} \times \frac{H}{2}$	$\frac{W}{4} \times \frac{H}{4}$	$\frac{W}{8} \times \frac{H}{8}$	$\frac{W}{16} \times \frac{H}{16}$	$\frac{W}{32} \times \frac{H}{32}$	1×1
	$7 \times 7, 64, \text{stride } 2$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	average pool classes number-d fc softmax
	$3 \times 3, \text{max pool, stride } 2$					

figuration is particularly chosen for its balance between depth and complexity, making it highly effective for tasks requiring detailed feature extraction without excessive computational demands. For images inputted as $N \times W \times C$, where N represents the batch size, W the width and height, assuming a square format, and C the number of color channels, the initial layer, known as conv1, consists of a 7×7 convolutional layer with a stride of 2. This configuration efficiently captures preliminary feature maps, generating 64 distinct feature maps. This large kernel size combined with a stride reduces the spatial dimensions early, allowing the network to focus on higher-level features by covering more area per filter.

Following this, the network applies a 3×3 max pooling layer with a stride of 2. This step further downsamples the feature maps, reducing their dimensions to make the network computationally efficient and enhance the robustness of the feature detection by providing spatial invariance—features can be recognized irrespective of slight shifts and distortions.

The core of ResNet50 comprises four residual blocks, each containing multiple residual units as illustrated in Figure 3. These units are designed to perform complex transformations while maintaining the integrity of the input data through the block. Each residual unit features a combination of 3×3 and two 1×1 convolutional layers. The initial 1×1 convolution acts as a bottleneck, reducing dimensionality and thus computational complexity, while the final 1×1 convolution restores the dimensions, preparing the output to be added back to the input.

Each unit's output is added to its input, utilizing skip connections that allow gradients to flow through the network directly, mitigating the vanishing gradient problem common in deep networks. This addition of the input (or skip connection) to the output facilitates the learning of residuals—modifications to the identity mapping of the input rather than a complete transformation, enhancing training effectiveness and performance sustainability.

In the first unit of each residual block, downsampling is performed to progressively decrease the size of the feature maps, focusing on more abstract feature representations at higher layers. Following the final residual block, ResNet50 employs a global average pooling layer, which converts the spatial feature maps into a single vector per feature map, effectively summarizing the prominent features in each channel.

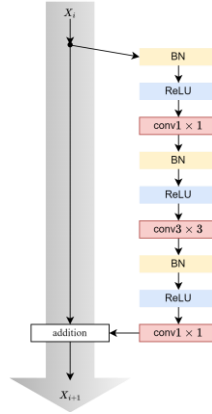


Fig. 3. ResNet50 Residual Unit

The output of the global average pooling layer feeds into a fully connected layer, which is tailored to the specific number of classes in the classification task. This layer transforms the global feature descriptors into class-specific scores. Finally, a softmax function is applied to these scores, converting them into probabilities for each class. The softmax output provides the likelihood of the image belonging to each class, facilitating effective classification based on the learned features throughout the network.

4 Experiments

The experimental environment of this paper is as follows: Operating System: Ubuntu 22.04; Programming Language Environment: Python 3.10.13 and PyTorch 2.0.1 with ROCm 5.7.1; CPU: Intel(R) Core(TM) i5-12600KF; 32GB Memory; GPU: Radeon RX 7900 XTX.

4.1 Data Set

To evaluate the performance of the malware classification model proposed in this study, we utilized the dataset from the Microsoft Malware Classification Challenge (BIG 2015) hosted by Kaggle [5]. This dataset comprises malicious software samples from nine different families, totaling 10,868 samples. Each malware file in the dataset is uniquely identified by a 20-character hash value, ensuring that each sample is distinct.

The dataset includes two types of files for each malware sample. The first type, a bytes file, contains the hexadecimal representation of the malware's binary content, excluding any file headers to ensure that the data remains non-executable and harmless. The second type, an asm file, serves as a metadata manifest. It is essentially a log that contains various metadata information extracted from the binary files using the IDA disassembler tool. This metadata includes detailed information such as function calls and strings found within the binary files, providing a comprehensive overview of each sample's structure and behavior.

For the purposes of this research, the dataset was split into a training set and a validation set in a 9:1 ratio, ensuring that a significant portion of the data was used for training the model while still reserving enough samples for an effective evaluation of the model's performance. The specific details of this split are provided in Table 2. This

Table 2. The number of each malware family in the training set and validation set in the dataset.

Family name	Train Samples	Validation Samples
Ramnit	1385	155
Lollipop	2230	248
Kelihos_ver3	2647	295
Vundo	427	48
Simda	37	5
Tracur	675	76
Kelihos_ver1	358	40
Obfuscator.ACY	1105	123
Gatak	911	102

structured approach to testing allows us to thoroughly assess the efficacy of the proposed model across a diverse set of malware samples, enabling robust validation of the model's ability to generalize across different types of malware and effectively classify them into their respective families.

The model proposed in this paper solely utilizes data from malware binary files and does not employ datasets that include IDA file format data.

4.2 Training

To commence the evaluation of the proposed malware classification model, the binary data of the malware samples from the dataset is initially processed. Each binary file is converted into multiple grayscale images of dimensions $W \times H$, where $W = H = 256$. The method for this conversion is detailed in Algorithm 1. This transformation is designed to visually represent the binary data in a format that can be effectively processed by convolutional neural networks.

Each of these grayscale images is then treated as a separate time step and fed into the BiD-ConvLSTM encoder. This encoder is specifically designed to handle sequential data, making it well-suited for analyzing the series of images as it can maintain contextual continuity across the frames. The output from the BiD-ConvLSTM encoder is a set of feature images that encapsulate the essential characteristics extracted from the binary data.

These feature images are subsequently used as inputs for the ResNet50 classifier. ResNet50, known for its deep network capabilities and resilience to vanishing gradients, processes these images to predict the probability of each malware class. The

predictions are made using a softmax function, which outputs the likelihood of each class, providing a probabilistic classification of the input malware image.

Algorithm 1: Algorithm for converting binary files to images

input : Binary files of malware. **output:** A set of malware images.

```

1  FileData = File.read();
2  FileData = [int(hexV alue, 16) for hexV alue in FileData];
3  W = H = 256;
4  l = len(FileData);
5  if l < W × H then
6  |   FileData.add([0] * (W * H - l));
7  |   l = W × H;
8  SeqLen =  $\frac{l}{W \times H}$ ;
9  index = 0;
10 for i ← 0 to SeqLen do
11 |   for j ← 0 to H do
12 | |   for k ← 0 to W do
13 | | |   if index < l then
14 | | | |   images[i, j, k] = FileData[index];
15 | | | |   index ++;
16 Generate images;

```

The model's performance is refined through the use of Focal Loss [19], a specialized loss function designed to address class imbalance by focusing more on hard-to-classify examples. This is particularly useful in datasets where certain malware families are underrepresented.

Training the model involves backpropagation and the use of the Adam optimization algorithm, known for its efficiency in handling sparse gradients and adaptive learning rate capabilities. The initial learning rate is set at 0.01, and a cosine annealing strategy with warm restarts is employed to adjust the learning rate dynamically during training. This approach helps in fine-tuning the learning process, allowing for faster convergence and potentially better generalization on unseen data.

Overall, this methodology not only systematically transforms and processes the malware data but also optimizes the learning process to enhance model accuracy and reliability in classifying diverse malware types.

4.3 Result

The paper employs four commonly used evaluation metrics: Accuracy, Precision, Recall, and F1-Score. Given the class imbalance present in the BIG2015 dataset, these metrics are calculated using a Macro-average approach. This method involves calculating the metrics individually for each class and then averaging the results. Macro-

averaging treats all classes equally, which is particularly important in datasets where some classes are underrepresented. This approach significantly highlights the impact of rare classes and provides a more accurate reflection of the model's performance under sample imbalance conditions.

To assess the impact of various components on the model's performance, the paper conducts comparative experiments. These experiments are designed to evaluate the effect of using a dynamic ConvLSTM instead of a BiD-ConvLSTM, and the inclusion of Feature Extraction (FE) and Spatial Attention Module (SAM) on the overall effectiveness of the model. The results of these comparative tests are detailed in Table 3. This methodical examination allows for a comprehensive understanding of how each component influences the accuracy and robustness of the malware classification model, ensuring that the final model configuration is optimized for the best performance on the challenging and diverse dataset.

Table 3. Results of the impact of various modules on model performance in the BIG 2015 dataset.

Method	Accuracy	F1-score	Precision	Recall
Dynamic ConvLSTM	95.70%	90.62%	89.55%	92.18%
BiD-ConvLSTM	97.16%	93.50%	94.46%	92.78%
BiD-ConvLSTM + FE	98.44%	97.05%	98.33%	96.05%
BiD-ConvLSTM + SAM	98.08%	96.56%	97.75%	95.68%

The experiments demonstrate that using only the dynamic ConvLSTM method, which extracts features in a single direction, fails to effectively combine global information and lacks a feature extraction mechanism to capture more rich and critical information. As a result, this configuration yielded the lowest accuracy.

In contrast, employing the BiD-ConvLSTM approach not only enhances the extraction of local features but also captures sequential information across both directions. This bidirectional approach significantly improves the model's ability to adapt to the task of malware classification, leading to a notable increase in accuracy.

The addition of the Feature Extraction (FE) module further enhances the model's performance. By enabling the model to better utilize the information present in the input data and extract more distinctive features, the FE module significantly increases the accuracy of classifying different malware families, achieving the highest accuracy observed in the experiments.

The integration of the Spatial Attention Mechanism (SAM) module allows the model to focus more intently on the parts of the data that are most relevant to the classification task. This targeted attention helps to improve the model's overall performance, with accuracy levels slightly below those achieved with the FE module but still significantly higher than the models without these enhancements.

Overall, these results underscore the importance of incorporating both bidirectional learning and advanced feature processing techniques such as feature extraction and attention mechanisms in improving the performance of machine learning models, especially in complex classification tasks like malware identification.

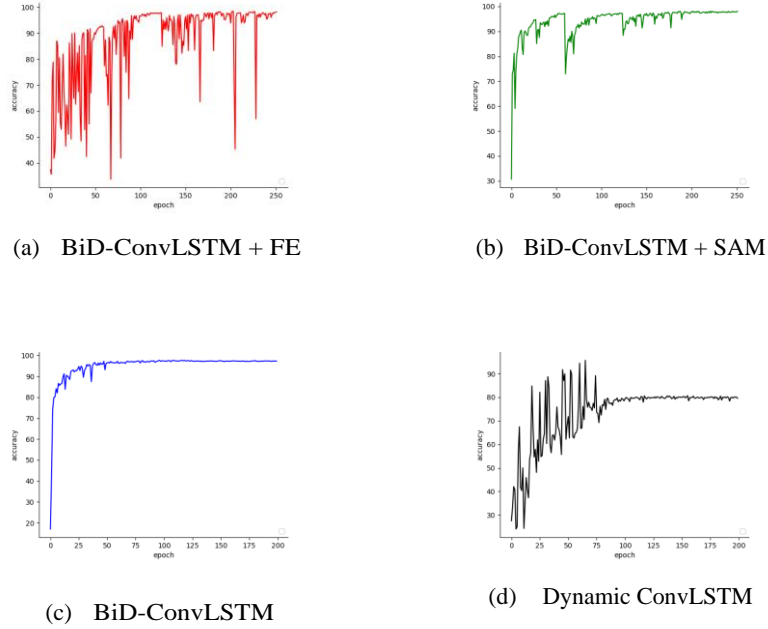
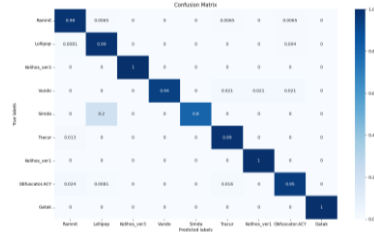


Fig. 4. accuracy graph

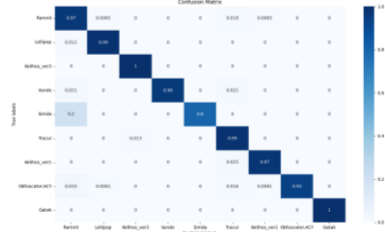
Figure 4 illustrates the trends in model accuracy as training progresses across various epochs. The graph indicates that the accuracy displays a periodic pattern, primarily due to the implementation of a cosine annealing strategy for adjusting the learning rate. This method periodically resets the learning rate to a higher value and then gradually decreases it, helping to prevent the model from settling into local minima too early during training. The encoder with the Feature Extraction (FE) module, which collects the most information, shows the most pronounced fluctuations in accuracy. This is because the richer feature set provided by the FE module enables more dynamic adjustments in learning, leading to significant shifts in model performance as the learning rate changes.

On the other hand, models without the FE and SAM (Spatial Attention Mechanism) modules exhibit signs of overfitting much earlier in the training process. This is attributed to the reduced complexity and diversity of features available to the model, which limits its ability to generalize from the training data. The situation is even more severe for models that do not utilize a bidirectional approach, leading to the cessation of training at around 200 epochs to prevent counterproductive overfitting.

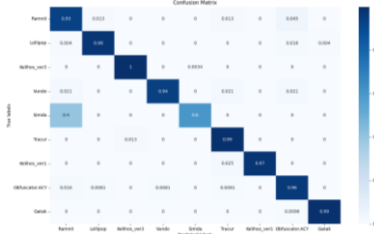
The incorporation of the SAM helps the encoder focus more intently on the most relevant parts of the data for the classification task. Unlike the FE module, SAM does not increase the amount of information processed but rather optimizes the model's attention towards significant features. This targeted focus results in the most stable change in accuracy, avoiding drastic fluctuations and maintaining a steady improvement in performance throughout the training cycles.



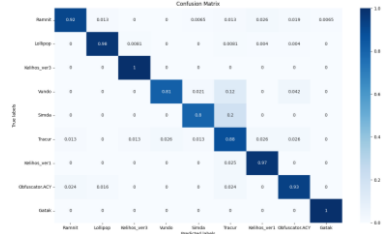
(e) BiD-ConvLSTM + FE



(f) BiD-ConvLSTM + SAM



(g) BiD-ConvLSTM



(h) Dynamic ConvLSTM

These observations underscore the effectiveness of the combined use of learning rate scheduling and architectural enhancements in managing model training dynamics and improving classification accuracy in complex tasks such as malware detection.

Figure 5 presents the confusion matrix for the malware classification model proposed in this study. The matrix reveals that while the use of Focal Loss has significantly alleviated the issues associated with class imbalance within the dataset, the model's performance is still notably affected. This imbalance impact is particularly evident in the F1-Score, which reflects both the precision and recall of the model. The F1-Score is crucial for understanding how well the model identifies and classifies each class, especially in datasets where some classes are underrepresented.

Additionally, the paper compares the proposed method with other approaches that have also used the BIG 2015 dataset. The comparative results are summarized in Table 4. The data in the table indicates that our method achieves an exceptional accuracy of 98.44%, outperforming other techniques. This high level of accuracy underscores the effectiveness and efficiency of our method in detecting and classifying malware, demonstrating that our enhancements—such as the implementation of BiD-ConvLSTM, feature extraction modules, and strategic application of the Focal Loss—contribute significantly to superior performance.

These findings highlight the advanced capability of the proposed model to handle the complexities of malware detection, proving its robustness against diverse and challenging datasets. The integration of sophisticated machine learning techniques ensures

that our model not only tackles the inherent issues of imbalance but also excels in accurate classification across various malware families.

Table 4. Comparison of the detection results on the BIG 2015 dataset.

Ref.	Models	Accuracy
Kim[20]	CNN,GRU,DNN	92.6%
Cho[21]	RNN-CNN	96%
Alaeiyan[22]	Multi-label fuzzy clustering	97.56%
Lin[23]	bit-level 1D CNN	96.32%
Shen[2]	BiLSTM+SAM+CAM	97.75%
This paper	Our approach	98.44%

5 Conclusion

This paper introduces an encoder that converts malware into images for deep neural network learning and utilizes ResNet50 for classification. This encoder effectively extracts features from malware of any size to generate feature images suitable for neural network training, eliminating the need for complex feature engineering and preprocessing. The encoder employs ConvLSTM to encode binary data of malware and captures dependencies before and after the data by integrating both forward and reverse directions. It also focuses on critical parts of the data using feature extraction and attention mechanisms. This encoder effectively resolves the issue of inconsistent sample sizes in malware, avoiding information loss due to normalization and significantly improving the classification accuracy and generalization ability for malware variants. The effectiveness of this method has been validated on a malware test dataset.

However, there are still areas where the encoder could be improved. For example, it cannot directly encode binary files of malware; it still requires manual conversion of binary data into multiple fixed-size images. Additionally, although the introduction of Focal Loss helps combat the problem of dataset imbalance, it has not completely resolved the issue, as there is still a noticeable decrease in accuracy for classes with fewer samples. In future work, we plan to further optimize the model structure and explore other feature extraction methods to address these issues.

References

1. Gopinath, M., Sethuraman, S.C.: A comprehensive survey on deep learning based malware detection techniques. *Computer Science Review* **47**, 100529 (2023)
2. Shen, G., Chen, Z., Wang, H., Chen, H., Wang, S.: Feature fusion-based malicious code detection with dual attention mechanism and bilstm. *Computers & Security* **119**, 102761 (2022)

3. Alazab, M., Venkataraman, S., Watters, P.: Towards understanding malware behaviour by the extraction of api calls. In: 2010 second cybercrime and trustworthy computing workshop. pp. 52–59. IEEE (2010)
4. Damodaran, A., Troia, F.D., Visaggio, C.A., Austin, T.H., Stamp, M.: A comparison of static, dynamic, and hybrid analysis for malware detection. *Journal of Computer Virology and Hacking Techniques* **13**, 1–12 (2017)
5. Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., Ahmadi, M.: Microsoft malware classification challenge. arXiv preprint arXiv:1802.10135 (2018)
6. Shi, X., Chen, Z., Wang, H., Yeung, D.Y., Wong, W.K., Woo, W.c.: Convolutional lstm network: A machine learning approach for precipitation nowcasting. *Advances in neural information processing systems* **28** (2015)
7. Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B.S.: Malware images: visualization and automatic classification. In: Proceedings of the 8th international symposium on visualization for cyber security. pp. 1–7 (2011)
8. Han, K., Lim, J.H., Im, E.G.: Malware analysis method using visualization of binary files. In: Proceedings of the 2013 Research in Adaptive and Convergent Systems, pp. 317–321 (2013)
9. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet large scale visual recognition challenge. *International journal of computer vision* **115**, 211–252 (2015)
10. Rezende, E., Ruppert, G., Carvalho, T., Ramos, F., De Geus, P.: Malicious software classification using transfer learning of resnet-50 deep neural network. In: 2017 16th IEEE international conference on machine learning and applications (ICMLA). pp. 1011–1014. IEEE (2017)
11. Rezende, E., Ruppert, G., Carvalho, T., Theophilo, A., Ramos, F., Geus, P.d.: Malicious software classification using vgg16 deep neural network’s bottleneck features. In: Information Technology-New Generations: 15th International Conference on Information Technology. pp. 51–59. Springer (2018)
12. Yuan, B., Wang, J., Liu, D., Guo, W., Wu, P., Bao, X.: Byte-level malware classification based on markov images and deep learning. *Computers & Security* **92**, 101740 (2020)
13. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
14. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14. pp. 630–645. Springer (2016)
15. Hochreiter, S., Schmidhuber, J.: Long short-term memory. *Neural computation* **9**(8), 1735–1780 (1997)
16. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. *Advances in neural information processing systems* **27** (2014)
17. Woo, S., Park, J., Lee, J.Y., Kweon, I.S.: Cbam: Convolutional block attention module. In: Proceedings of the European conference on computer vision (ECCV). pp. 3–19 (2018)
18. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. In: Proceedings of the IEEE international conference on computer vision. pp. 2980–2988 (2017)

20. Kim, C.H., Kabanga, E.K., Kang, S.J.: Classifying malware using convolutional gated neural network. In: 2018 20th International conference on advanced communication technology (ICACT). pp. 40–44. IEEE (2018)
21. Cho, Y.: Dynamic rnn-cnn based malware classifier for deep learning algorithm. In: 2019 29th International Telecommunication Networks and Applications Conference (ITNAC). pp. 1–6. IEEE (2019)
22. Alaeiyan, M., Dehghantanha, A., Dargahi, T., Conti, M., Parsa, S.: A multilabel fuzzy relevance clustering system for malware attack attribution in the edge layer of cyber-physical networks. *ACM Transactions on Cyber-Physical Systems* **4**(3), 1–22 (2020)
23. Lin, W.C., Yeh, Y.R.: Efficient malware classification by binary sequences with one-dimensional convolutional neural networks. *Mathematics* **10**(4), 608 (2022)