Quest Journals
Journal of Software Engineering and Simulation
Volume 11 ~ Issue 2 (2025) pp: 30-36
ISSN(Online) :2321-3795 ISSN (Print):2321-3809
www.questjournals.org

Research Paper

Malware Classification Based on Deep Learning

Ran Xu
(Hongta Group Yuxi Cigarette Factory, Yuxi, China)
Corresponding Author: Ran Xu

ABSTRACT: *This paper proposes a novel malware classification method for personal computers, which I have named MalCNN. The motivation behind selecting this topic is that, in the era of the pandemic, as online work continues to grow, computer security has become an increasing concern.Currently, most malware classification methods rely on either a hard decision strategy or analysis conducted by cybersecurity professionals. However, traditional classification models and approaches are no longer suitable for the big data era. In the digital world, millions of malware programs spread every moment, making traditional classification methods inefficient due to their slow speed and inaccurate results. As a result, these methods struggle to meet modern security demands.To address these challenges, MalCNN attempts to establish a soft decision classification method based on computer vision techniques, allowing for the annotation of samples that do not meet predefined classification conditions.*

KEYWORDS: *Malware Classification; Soft Decision; Computer Vision*

R*eceived 13 Feb., 2024;* ***R****evised 25 Feb., 2025;* ***A****ccepted 27 Feb., 2025 © The author(s) 2025.*
P*ublished with open access at* ***www.questjournas.org***

I. INTRODUCTION

With the widespread adoption of computers in people's daily lives, issues related to computer security have become increasingly prominent [1]. Nowadays, computers are deeply integrated into various aspects of life, including work, travel, social interactions, and payments, making them more essential than ever. Against this backdrop, cyberattack methods have become more diverse, complex, and sophisticated [1][2]. Additionally, the number of malware samples, such as viruses and worms, has experienced explosive growth [1][3].

Conventional malware classification methods primarily rely on hard decision-making or require security experts to analyze and determine the malware family, which can no longer meet today’s security demands [1][2]. Furthermore, in the field of software security, security professionals are often in an exposed position: the security software they develop is publicly accessible, whereas attackers operate in the shadows. This means that security experts cannot fully anticipate the vulnerabilities exploited by attackers or the evasion techniques they employ to bypass security measures. As a result, hard decision classifiers struggle significantly when faced with such an overwhelming volume of malware samples.

Most computer software users today lack vigilance against malware, which can compromise the security, integrity, or usability of data and programs on a network. Moreover, modern malware increasingly exhibits characteristics such as long dormancy periods, enhanced stealth, and self-deletion mechanisms [4]. Relying solely on software usage experience or traditional hard-decision malware classification models makes accurate classification increasingly difficult.

To address this issue, this study adopts a soft decision strategy and utilizes deep learning to develop a more efficient and generalizable malware classification model, achieving higher accuracy in classification.

Deep learning has been a popular research direction in recent years. A review of the literature reveals that malware classification research in China remains relatively limited, with many existing models demonstrating suboptimal performance. Therefore, we believe that combining deep learning with malware classification has promising prospects. Compared to other soft decision classification models in this field, such as N-gram and LSTM models, the proposed MaLCNN model integrates the algorithmic classification approaches of these models while reducing computational overhead. Through multiple iterations, the model has achieved superior classification accuracy. This research provides valuable insights for the practical application of malware classification in the cybersecurity industry.

II. RELATED WORK

2.1 N-GRAM-BASED CLASSIFICATION MODEL

The N-gram algorithm [7] is based on probability theory and statistical language models. It assumes that the N-th word in a text is only related to the previous (N-1) words. According to this principle, the most probable word to follow a given sequence can be predicted. Typically, text content is segmented into fixed-size byte sequences, where each segment is called a gram. The most common values for N are 2 (bigram) and 3 (trigram). The frequency of these grams is then statistically analyzed, filtered according to predefined rules, and stored in a key gram table, which serves as a feature vector for the training text.

The N-gram algorithm was initially considered for this study based on the assumption that .byte files in malware may exhibit statistical patterns similar to natural language text. After reviewing relevant research, we selected the N-gram algorithm for testing. The work of [5] and others treated malware assembly code as "words" in a text. Under this approach, an entire malware sample could be viewed as a "document," where different "documents" have distinct writing styles and purposes—similar to a text classification task. By applying the N-gram model from natural language processing (NLP) to malware classification, predictions could be made based on these extracted patterns.

Experimental results showed that setting N = 3 (trigram) achieved the best performance, with an accuracy of up to 96%. This confirms that the N-gram algorithm is applicable to malware classification and provides valuable insights for our study. We determined that assembly code features are relevant for malware classification, making NLP-based processing a feasible approach. However, despite achieving 96% accuracy, there is still room for improvement.

One of the main limitations of this method is its difficulty in adapting to different computing architectures. This is because assembly syntax and mnemonics vary significantly across architectures. For example, an N-gram model trained on x86 assembly code cannot be directly transferred to ARM-based architectures, leading to limited applicability.

Today, devices using ARM assembly architecture, such as mobile phones and edge computing devices, account for a significant portion of computing platforms. Security concerns for these devices are just as critical as for traditional x86-based computers. Therefore, focusing only on x86-based malware classification is not a viable solution.

To overcome these limitations, we propose a cross-architecture malware classification approach that works on x86, ARM, and other platforms. This is why our model does not use assembly code as input features. Instead, our proposed MaLCNN model is designed to classify malware across different architectures without requiring separate models for each platform. This eliminates the need to train multiple models repeatedly for different architectures, improving efficiency and scalability in malware classification.

2.2 LSTM-BASED CLASSIFICATION MODEL

Long Short-Term Memory (LSTM) [6] is one of the commonly used deep learning algorithms, capable of effectively addressing gradient explosion and gradient vanishing problems in recurrent neural networks (RNNs). Compared to standard RNNs, LSTM performs better in handling long-sequence information.

The internal structure of LSTM can be divided into three main components:

Forget Gate – The network selectively forgets certain information from the previous node while retaining the most important features.

Selective Memory – The network processes the current input (Xt in the diagram) and selectively remembers important parts while discarding less relevant information.

Output Gate – After the above processing, the network determines which information will be passed to the next node.

By incorporating these mechanisms, LSTM is able to capture long-term dependencies, making it well-suited for sequential data processing, such as malware classification based on bytecode or API call sequences.

[8] and colleagues proposed a method that utilizes LSTM for malware classification. Similar to the N-gram approach, they treated assembly code as words, mapping an entire malware sample into a "document." This transformation allowed malware classification to be framed as a text classification problem, making it possible to apply LSTM for classification.

However, a significant drawback of this method is that it can only predict malware based on a single assembly architecture. A model trained on x86 assembly cannot be applied to ARM-based machines, as the syntax and mnemonics of assembly code differ significantly between architectures. This limits its applicability, making it less than ideal as a universal solution.

As discussed in Chapter 4, LSTM outperforms the N-gram algorithm in malware classification. The selective memory mechanism and textual representation of malware in LSTM provided valuable insights for our research. Inspired by this, our study continues to adopt mapping and feature extraction techniques while focusing on improving classification accuracy and reducing computational overhead.

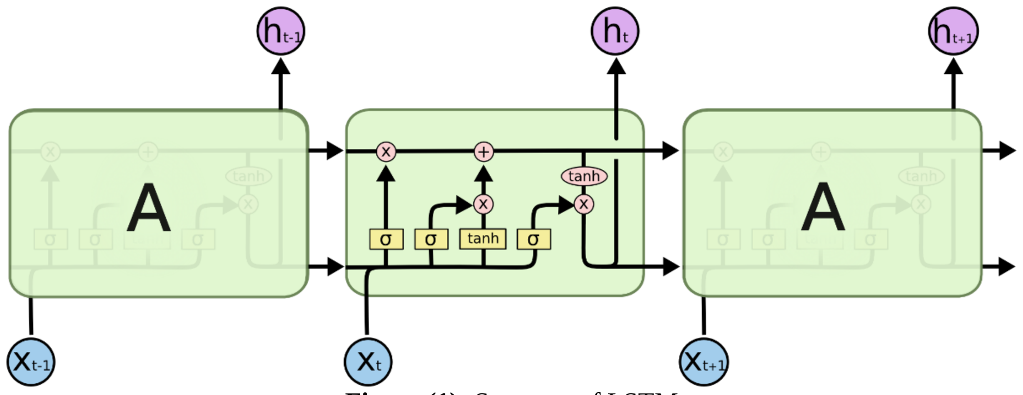

Figure (1): Structure of LSTM

III. MALCNN

3.1 DATASET

For our study, we use BIG2015, an open-source dataset provided by Microsoft [9]. This dataset consists of nine malware families, specifically: Ramnit, Lollipop, Kelihos_ver3, Vundo, Simda, Tracur, Kelihos_ver, Obfuscato.ACY, and Gatak. Each sample in the dataset has been manually analyzed and classified by Microsoft security experts, ensuring accurate malware family labels.

A detailed overview of the dataset is provided in Table 1. After data preprocessing and cleaning, the dataset contains 10,868 samples.

Notably, BIG2015 does not provide executable files; instead, it offers binary-formatted, non-executable .byte files and .asm assembly files corresponding to each malware sample. In our experiments, we only use the .byte files for feature mapping to build our classification model.

IV. CONCLUSION

Table 1: Malware Categories and Distribution in the Dataset

Malware Family	Sample Count
Ramnit	1541
Lollipop	2478
Kelihos_ver3	2942
Vundo	475
Simda	42
Tracur	751
Kelihos_ver	398
Obfuscato.ACY	1228
Gatak	1013

The BIG2015 dataset does not store malware samples in categorized folders; instead, all samples are placed in a single directory, while label information is provided separately in an Excel file. To partition the dataset, we extract filenames based on their corresponding categories from the Excel file.

For dataset splitting, we randomly select 20% of the samples from each category as the test set, while the remaining 80% are used for training. This ensures a balanced distribution across different malware families for both training and evaluation.

3.2 FEATURE ENGINEERING

MalCNN adopts a computer vision-based approach to train a soft decision model. The first step in feature engineering is to map a malware sample into an image while ensuring that essential features are extracted efficiently without compromising processing speed. To achieve this, several key issues need to be considered:

3.2.1 Relationship Between Image Size and Source File

Malware samples vary significantly in size due to differences in type and obfuscation techniques. If only a portion of a file is used for mapping, important information may be lost. On the other hand, mapping the entire file could reduce processing efficiency and introduce excessive redundancy.

To address this issue, we map the entire malware file into a grayscale image by reading the malware file in binary format. Specifically, we read 1024 bits per row, forming a grayscale image, and pad with zeroes if the last row is incomplete. This approach preserves as many features as possible while maintaining processing efficiency.

The resulting grayscale images vary in size. Initially, we considered using RoI Pooling [10] to standardize image dimensions. However, experimental results showed poor performance with this method. We hypothesize that this is due to the pixel values in malware images being binary (0 or 1) rather than continuous grayscale values, causing RoI Pooling's max-pooling operation to distort key features.

Through experimentation, we found that resizing the images directly to 224×224 pixels yielded the best classification results.

3.2.2 Handling Anomalies and Missing Values

Some anomalies and missing values may exist in our dataset, such as Malware samples whose filenames cannot be found in the label table or Missing label information.

These issues need to be addressed to prevent disruptions in feature engineering, which could negatively impact model training and evaluation.

Since the original dataset is extremely large [9] and the proportion of affected samples is less than 0.01%, we adopt a simple exclusion approach—ignoring these problematic samples. This method is efficient and has negligible impact on model performance.

Additionally, missing values in our dataset primarily affect labels or filenames rather than numerical features. Thus, common imputation techniques (such as mean or median filling) are not applicable in this case.

3.3 Model Architecture

In this study, we selected ResNeSt as the model, an improved version of ResNet [14][15]. Unlike ResNet, ResNeSt introduces a network slicing mechanism, as shown in the figure. During inference, the model no longer follows a single sequential computation path. Instead, the input is duplicated into k copies, each sent into a separate Cardinal for independent computation. Within each Cardinal, the input is further split into r slices, processed separately, and then merged using Concatenate to combine the results.

This architecture enhances accuracy because even if a particular Cardinal learns incorrect features during training, other Cardinals can still capture the correct patterns, improving overall classification performance. Additionally, within each Cardinal, ResNeSt incorporates a split-attention mechanism [14], which enables the model to learn relationships between one or multiple channels, further refining feature extraction and classification accuracy.

For this experiment, we employed the ResNeSt50 network, balancing computational efficiency with high classification performance.

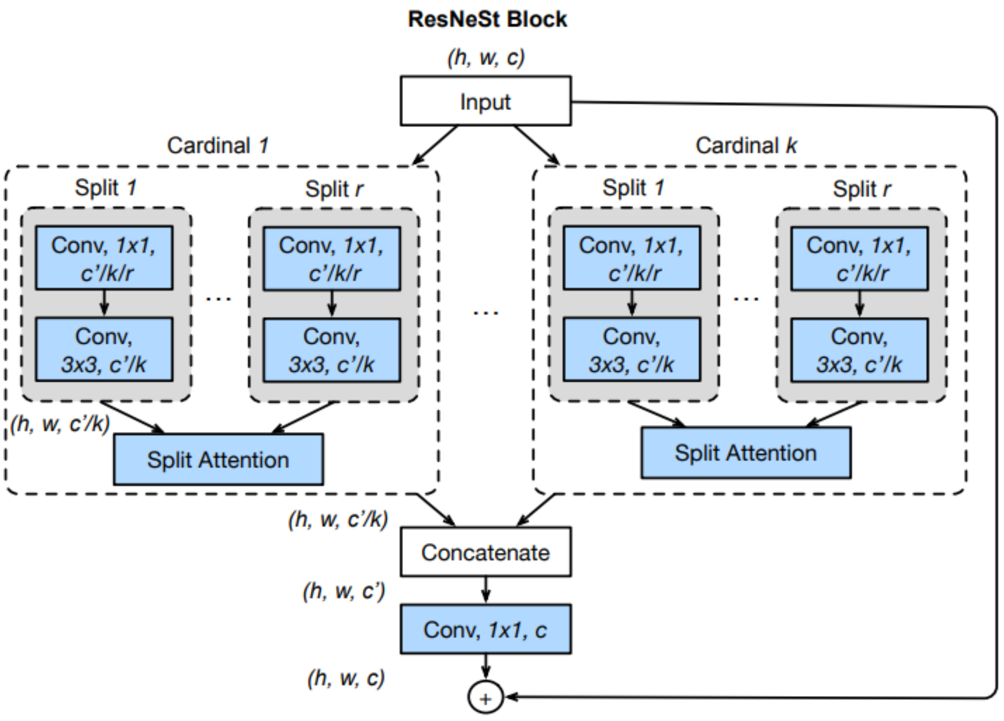

Figure (1): Architecture of ResNeSt Network

V. EXPERIMENT RESULT

4.1 EXPERIMENTAL ENVIRONMENT

Our experimental environment configuration is shown in Table 2.

Table 2: Experimental Environment Configuration

Component	Version
Pytorch	1.10.1
Numpy	1.21.5
CUDA	11.2
CuDNN	8.1.1
Pandas	1.3.4

The training hyperparameters are shown in Table 3.

Table 3: Training Parameter Information

Hyperparameter	Value
Learning Rate	0.01
Epoches	50
Weight Decay	0.0001
Batch Size	64
Shuffle	True
Drop Last	True

4.2 EXPERIMENTAL SETUP

Since the methods proposed in [6][7] cannot be migrated to ARM-based machines, and the dataset in [9] does not provide ARM-related data, we did not conduct comparative experiments. Instead, we performed experiments solely on the x86 platform, using model accuracy on the test set as the evaluation metric. The experiments were conducted on a Windows 10 system with an Nvidia RTX 3060Ti GPU.

4.3 EXPERIMENTAL RESULTS

At the beginning of training, the learning rate was set to 0.01, as the model weights were initialized. A larger learning rate in the early iterations allows the model to explore a broader parameter space to find an optimal region for subsequent fine-tuning [24].

As shown in Figure 2, the learning rate decreased rapidly at the start. If a high learning rate were maintained for too long, the model could overshoot the optimal region, so it was necessary to gradually reduce the learning rate to allow finer exploration in a promising region. However, between 20 and 25 iterations, we observed a temporary increase in model loss, which we suspect was due to the learning rate being too high at that stage, causing the model to jump out of a local minimum, leading to an increase in loss. As the learning rate continued to decrease in later iterations, the model resumed learning, minimizing loss and converging to a better set of parameters.

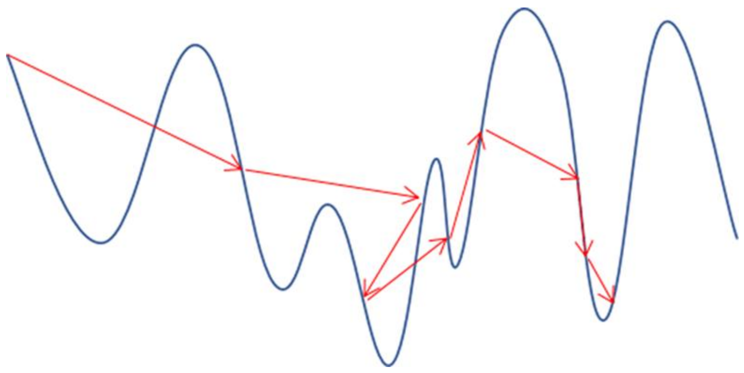

Figure (2): Illustration of the Impact of Learning Rate

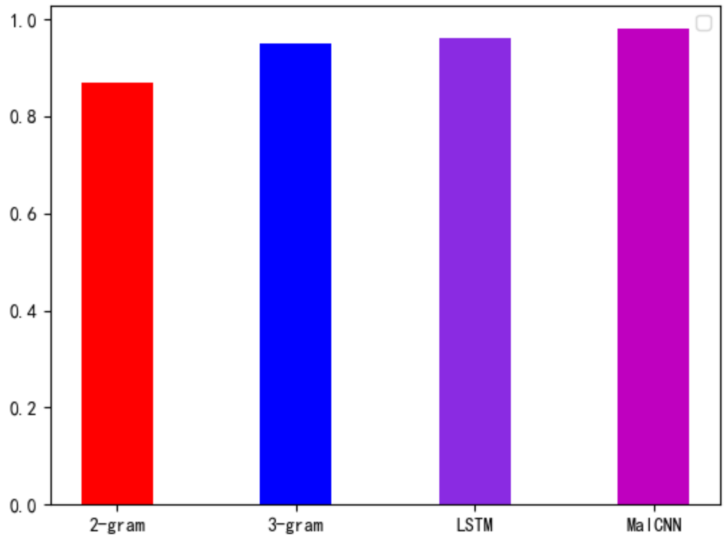

Figure (3): Comparison of Model Accuracy Across Different Models

Table 4: Model Accuracy

Models	2-gram	3-gram	LSTM	MaLCNN
Accuracy	87%	95%	96%	98.4%

The reference models address malware classification by converting assembly code into a text classification problem. In Table 4, we present the final test accuracy of all models. Among them, the N-gram model performed best when n = 3, while LSTM achieved a high accuracy of 96%.

However, the N-gram method requires maintaining a large word frequency matrix [6], making search and update operations highly costly, leading to a significantly higher model complexity compared to MalCNN. Additionally, LSTM, as an RNN-based model, typically has a higher computational complexity than MalCNN, which belongs to the CNN model family.

Therefore, MalCNN has a key advantage of lower model complexity, reducing computational overhead far more efficiently than the other three methods.

VI. CONCLUSION

In this paper, we proposed a malware classification model based on computer vision methods. We explored various feature engineering techniques for processing raw binary .bytes files and identified an approach well-suited for our research. We then selected ResNeSt as the base model and, after conducting experiments and reviewing related literature, incorporated AutoAugment, mix-up, and other training techniques to improve model accuracy and robustness. Additionally, we optimized Label Smoothing to ensure compatibility with other training strategies.

Compared to other models and methods, our approach achieved higher classification accuracy while maintaining lower model complexity. However, we believe there is still room for improvement. In future work, we plan to integrate adversarial training to evaluate the model's robustness against state-of-the-art attack algorithms and explore potential solutions to further enhance security and reliability.

REFERENCES

[1]. Sun C C, Hahn A, Liu C C. Cyber security of a power grid: State-of-the-art[J]. International Journal of Electrical Power & Energy Systems, 2018, 99: 45-56.

[2]. Von Solms R, Van Niekerk J. From information security to cyber security[J]. computers & security, 2013, 38: 97-102.

[3]. Pascanu R, Stokes J W, Sanossian H, et al. Malware classification with recurrent networks[C]//2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2015: 1916-1920.

[4]. Skoudis E, Zeltser L. Malware: Fighting malicious code[M]. Prentice Hall Professional, 2004.

[5]. Raff E, Zak R, Cox R, et al. An investigation of byte n-gram features for malware classification[J]. Journal of Computer Virology and Hacking Techniques, 2018, 14(1): 1-20.

[6]. Hochreiter S, Schmidhuber J. Long short-term memory[J]. Neural computation, 1997, 9(8): 1735-1780.

[7]. Brown P F, Della Pietra V J, Desouza P V, et al. Class-based n-gram models of natural language[J]. Computational linguistics, 1992, 18(4): 467-480.

[8]. Athiwaratkun B, Stokes J W. Malware classification with LSTM and GRU language models and a character-level CNN[C]//2017 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, 2017: 2482-2486.

[9]. Ronen R, Radu M, Feuerstein C, et al. Microsoft malware classification challenge[J]. arXiv preprint arXiv:1802.10135, 2018.

[10]. Ren S, He K, Girshick R, et al. Faster r-cnn: Towards real-time object detection with region proposal networks[J]. Advances in neural information processing systems, 2015, 28.

[11]. Sutton R S, Barto A G. Reinforcement learning: An introduction[M]. MIT press, 2018.

[12]. Cubuk E D, Zoph B, Mane D, et al. Autoaugment: Learning augmentation strategies from data[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019: 113-123.

[13]. Rivest J F, Soille P, Beucher S. Morphological gradients[J]. Journal of Electronic Imaging, 1993, 2(4): 326-336.

[14]. Zhang H, Wu C, Zhang Z, et al. Resnest: Split-attention networks[J]. arXiv preprint arXiv:2004.08955, 2020.

[15]. He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[16]. Madry A, Makelov A, Schmidt L, et al. Towards deep learning models resistant to adversarial attacks[J]. arXiv preprint arXiv:1706.06083, 2017.

[17]. Zhang H, Cisse M, Dauphin Y N, et al. mixup: Beyond empirical risk minimization[J]. arXiv preprint arXiv:1710.09412, 2017.

[18]. Zhang L, Deng Z, Kawaguchi K, et al. How does mixup help with robustness and generalization?[J]. arXiv preprint arXiv:2010.04819, 2020.

[19]. Ying X. An overview of overfitting and its solutions[C]//Journal of Physics: Conference Series. IOP Publishing, 2019, 1168(2): 022022.

[20]. Müller R, Kornblith S, Hinton G E. When does label smoothing help?[J]. Advances in neural information processing systems, 2019, 32.

[21]. Hinton G, Vinyals O, Dean J. Distilling the knowledge in a neural network[J]. arXiv preprint arXiv:1503.02531, 2015, 2(7).

[22]. Loshchilov I, Hutter F. Decoupled weight decay regularization[J]. arXiv preprint arXiv:1711.05101, 2017.

[23]. Kingma D P, Ba J. Adam: A method for stochastic optimization[J]. arXiv preprint arXiv:1412.6980, 2014.

[24]. You K, Long M, Wang J, et al. How does learning rate decay help modern neural networks?[J]. arXiv preprint arXiv:1908.01878, 201