Deep Learning Defenders: Harnessing Convolutional Networks for Malware Detection

Ahmed Abdelmonem* , Shimaa S. Mohamed

Faculty of Computers and Informatics, Zagazig University, Zagazig 44519, Sharqiyah, Egypt

Emails: aabdelmounem@zu.edu.eg; shimaa_said@zu.edu.eg;

Abstract

Malware attacks continue to pose a significant threat to computer systems and networks worldwide. Traditional signature-based malware detection methods have proven to be insufficient in detecting the increasing number of sophisticated malware variants. This has led to the exploration of new approaches, including machine learning-based techniques. In this paper, we propose a novel approach to malware detection using residually connect convolutional networks. We demonstrate the effectiveness of our approach by training CNN on a large dataset of malware samples and benign files and evaluating its performance on a separate test set. Extensive experiments on a public dataset of malware images demonstrated that our model could achieve high accuracy in detecting both known and unknown malware samples. The findings suggest that our residual convolution has great potential for improving malware detection and enhancing the security of computer systems and networks.

Keywords: Computational Intelligence; Deep Learning; Convolutional Neural Networks; Malware Detection; Machine Learning.

1. Introduction

Malware, short for malicious software, refers to any software designed with malicious intent to harm computer systems, steal sensitive information, or disrupt normal operations. The criticality of malware lies in its ability to cause significant damage to individuals, businesses, and even entire networks. Malware can take various forms, including viruses, worms, Trojans, ransomware, and spyware. These malicious programs can spread rapidly, exploit vulnerabilities, and compromise the security and privacy of affected systems [1-3].

Traditional methods of malware detection typically rely on signature-based scanning and heuristic analysis. The former type of method involves comparing files or code snippets against a database of known malware signatures, which represent specific patterns or characteristics that are unique to each malware variant. When a file matches a signature, it is flagged as malware, and appropriate action is taken. This method is effective in detecting known malware and is relatively fast. However, it struggles to detect zero-day threats because they have no known signatures. It requires frequent updates to the signature database to keep up with new malware variants, which can result in a lag between the emergence of a new threat and its detection [4-5]. On the other hand, heuristic analysis methods involve examining the behavior and characteristics of files or programs to identify potential malware. They look for suspicious activities, such as attempts to modify system files or unusual network communication. Heuristic analysis can detect unknown malware to some extent but often generates false positives as legitimate files or programs with similar

Doi: https://doi.org/10.54216/IJAACI.010203
Received: January 15, 2022 Accepted: May 23, 2022

behavior may also trigger alerts. Moreover, sophisticated malware can employ techniques to evade heuristic analysis, making it less effective against advanced threats [6-7].

The above methods suffer from several limitations. Firstly, they heavily rely on known signatures or predefined rules, which means they struggle to detect new or evolving malware variants. Zero-day threats, which are previously unknown malware, can easily bypass signature-based scanning. Secondly, the reliance on file signatures makes these methods vulnerable to polymorphic or metamorphic malware, which can change their signatures or appearance to evade detection [8-12]. Thirdly, they cannot often analyze the runtime behavior of programs, making it difficult to detect malware that hides or activates malicious activities after installation [13-15]. In response to these challenges, modern security solutions are incorporating machine learning (ML) approaches to enable the detection of malware based on its behavior rather than relying solely on known signatures. By analyzing the actions and patterns exhibited by programs, these methods can identify suspicious activities and potentially malicious behavior [16-20].

In this paper, we propose an applied approach to malware detection using convolutional neural networks (CNNs). CNNs are a type of deep learning model that has achieved great success in various image and signal processing tasks. Our approach involves training CNN on a large dataset of malware samples and benign files and using the learned model to classify new files as either malware or benign. The main contribution of this paper is the demonstration of the effectiveness of our proposed approach in detecting both known and unknown malware samples. We show that our CNN-based approach outperforms traditional signature-based methods and other machine learning-based techniques in terms of accuracy and detection rates. Additionally, we provide insights into the learned representations of CNN and the features that are most discriminative for malware detection. This paper serves as a step towards a more effective and efficient defense against the ever-evolving threat of malware attacks.

2. Related work

Malware detection has been a subject of extensive research for many years, and a variety of approaches have been proposed to address this problem. In this section, we review the related work on malware detection, with a focus on ML-based techniques. For example, Justin et al [5] addressed the increasing concern of malware threats targeting Android devices, which have become popular due to their widespread usage. They proposed an ML approach to detect Android malware, recognizing the need for advanced techniques to combat the evolving nature of these threats. They experimented with different algorithms, including Naive Bayes (NB), decision trees (DT), and support vector machines (SVM), to train classifiers capable of distinguishing between benign and malicious Android applications. Sethi et al., [6] presented a comprehensive framework to address the challenge of effectively detecting and classifying malware by combining static and dynamic analysis techniques with multiple ML techniques. They recognized that malware is constantly evolving and becoming more sophisticated, requiring a robust and adaptable approach for accurate detection. In [8], Liu et al studied the challenge of malware classification and the detection of new and previously unseen malware variants. They proposed a framework that combines feature extraction, feature selection, and ML algorithms to automatically classify malware samples based on the behavioral characteristics of malware. By analyzing the system call sequences and constructing dynamic behavior graphs, their framework extracted meaningful features that represent the actions and interactions of the malware within a system. Ivan et al [10] addressed the importance of behavior-based detection methods, which focused on analyzing the actions and behaviors of malware samples rather than relying on static signatures. They recognized that behavior-based approaches offer advantages in detecting previously unseen and evolving malware variants. They also presented a comprehensive analysis of different ML techniques applied to behavior-based malware detection including artificial neural networks (ANN), SVM, DT, and NB classifiers. They consider factors such as detection accuracy, false positive rates, and computational efficiency. In [11], Rathore et al explored the need for effective malware detection methods due to the increasing complexity and sophistication of malware attacks. They developed a framework that integrated ML classifiers with deep networks for malware detection. Narudin et al [13] explored the threat of malware targeting mobile devices and the need for effective detection methods. They focused on evaluating different ML classifiers to determine their suitability and performance in detecting mobile malware. In [14], Jin et al studied the challenge of effective detection of Android malware by focusing on the analysis of app permissions, which played a crucial role in identifying potential malicious behavior. They focused on the development of a feature selection method that ranks the importance of Android app permissions based on their relevance to malware detection. They utilized information gain and mutual information to

Doi: https://doi.org/10.54216/IJAACI.010203
Received: January 15, 2022 Accepted: May 23, 2022

assess the significance of permissions in differentiating between benign and malicious applications. In [15], Amos et al developed a framework that applied ML classifiers to analyze the runtime behavior of Android applications and identify potential malware. They empirically evaluated the performance of different ML classifiers for dynamic Android malware detection using a large dataset of Android applications. In [17], the paper addressed the challenge of detecting Android malware by focusing on the analysis of app permissions and API calls, which were crucial indicators of potentially malicious behavior. They developed a framework that combined ML techniques with these features to classify Android applications as either benign or malicious.

3. Methodology

In this section, we describe the methodology used in our proposed computational intelligence approach for malware detection based on intelligent convolutional networks. We describe the preprocessing steps applied to the raw data to prepare it for input to the model. Next, we explore the architecture and hyperparameters of the convolutional model used in our experiments. Then, we discuss the training process, including the optimization algorithm and the stopping criteria for training. Finally, we outline the evaluation metrics used to assess the performance of our models on the test set. By providing a detailed methodology, we aim to ensure the reproducibility of our results and enable other researchers to build upon our work.

A. Case study

The Malevis dataset serves as a valuable case study in our research paper on malware detection. The dataset is a comprehensive collection of imaged malware samples gathered from diverse sources, making it a reliable and representative resource for evaluating the performance of our proposed detection method. The Malevis dataset is composed of malware images belonging to 25 and also contains samples belonging to the legitimate class. The data is built through a bin2png script developed by Sultanik to extract binary images from malware files in 3 channels. The Malevis dataset contains a total of 9100 training samples and 5126 testing samples. Each class in the training set contains 350 images, which reflects that the data is balanced. Conversely, the validation set contains a different number of images per class, in which the legitimate class was the majority class with 1482 samples.

The Malevis dataset comprises a wide range of malware samples, including various types such as trojans, worms, ransomware, and spyware. This diversity enables us to assess the effectiveness of our malware detection approach across different malware families and understand its robustness in detecting evolving threats. In this paper, we utilize the Malevis dataset to evaluate the performance of our machine learning-based malware detection algorithm. By training our algorithm on a subset of the dataset and testing it on the remaining samples, we can assess its accuracy, precision, recall, and other relevant metrics. This allows us to measure the efficacy of our approach in identifying and classifying malware accurately.

B. Data Preprocessing

Before using the Malevis data to train our classifier, a set of data preprocessing steps is applied as a critical step in image classification tasks as it helps to improve the quality of the data, improve the performance of the models, and ensure meaningful and accurate predictions. First, we have resized all samples to a uniform size (i.e., $128x2128$ pixels) to ensure consistency and help models process the data efficiently. Then, we rescaled all pixel values to a standardized range (e.g., 0 to 1), and improved the convergence during the training. After that, we normalize the pixel values to reduce the impact of variations in lighting conditions or color channels. This involves subtracting the mean and dividing it by the standard deviation of the pixel values across the dataset or per image. This helps the model focus on the patterns and features within the images rather than being influenced by variations in pixel intensity.

C. Model Building

The main building of the architecture of our model is inspired by the residual convolutional network presented in [21]. In our model, we employed residual convolutions as a key architectural component to improve performance and training efficiency. Residual convolutions, also known as residual blocks or skip connections, were introduced to alleviate the vanishing gradient problem and enable the effective flow of gradients during backpropagation. Residual convolutions make use of skip connections that directly connect the input of a convolutional layer to its output. This

Doi: https://doi.org/10.54216/IJAACI.010203
Received: January 15, 2022 Accepted: May 23, 2022

allows the model to learn residual mappings, capturing the difference between the input and output of the convolutional layer. By adding the original input to the output, the model can effectively propagate gradients through the skip connection, facilitating the optimization process. With this design, we provide a direct path for gradient flow, residual convolutions enable easier backpropagation, and our model is allowed to effectively learn deep representations.

Batch normalization is incorporated into our residual blocks to improve their training efficiency and overall performance. By normalizing the feature maps within each mini-batch during the training process, batch normalization reduces the internal covariate shift problem and accelerates convergence. In our model, batch normalization is applied after each convolutional. It operates by normalizing the features based on the mean and variance calculated across the current mini-batch.

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}} \tag{1}$$

To attain nonlinear information, BatchNormalization includes two trainable parameters (scale and shift), the actual operation process is as follows:

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)} \tag{2}$$

batch normalization acts as a regularizer, mitigating the impact of overfitting. By reducing the interdependence of the activations within each mini-batch, batch normalization introduces some noise to the model, making it more robust and less prone to overfitting. This regularization effect helps to improve the model's generalization performance on unseen data.

The residual blocks are stacked in our model to learn the complex patterns of malware in the input images. Then, the output of the last block is globally averaged across channel dimensions and passed to the SoftMax layer, at which the final decision is computed as class probabilities. The general implementation of the architecture of our model is given as follows:

```
1.  from tensorflow.keras.layers import *
2.  class ResidualBlock(Model):
3.      def __init__(self, channels, stride=1):
4.          super(ResidualBlock, self).__init__(name="ResidualBlock")
5.          self.flag = stride != 1
6.          self.conv1 = Conv2D(channels, 3, stride, padding="same")
7.          self.bn1 = BatchNormalization()
8.          self.conv2 = Conv2D(channels, 3, padding="same")
9.          self.bn2 = BatchNormalization()
10.         self.relu = ReLU()
11.         if self.flag:
12.             self.bn3 = BatchNormalization()
13.             self.conv3 = Conv2D(channels, 1, stride)
14.
15.     def call(self, x):
16.         x1 = self.conv1(x)
17.         x1 = self.bn1(x1)
18.         x1 = self.relu(x1)
19.         x1 = self.conv2(x1)
20.         x1 = self.bn2(x1)
21.         if self.flag:
22.             x = self.conv3(x)
```


Doi: https://doi.org/10.54216/IJAACI.010203
Received: January 15, 2022 Accepted: May 23, 2022

```
23.          x = self.bn3(x)
24.        x1 = add([x, x1])
25.        x1 = self.relu(x1)
26.        return x1
27.
28.
29. class ResidualNet(Model):
30.     def __init__(self, classes=10):
31.        super(ResidualNet, self).__init__(name="ResidualNet")
32.        self.conv1 = Conv2D(64, 7, 2, padding="same")
33.        self.bn = BatchNormalization()
34.        self.relu = ReLU()
35.        self.mp1 = MaxPooling2D(3, 2)
36.
37.        self.conv2_1 = ResidualBlock(64)
38.        self.conv2_2 = ResidualBlock(64)
39.
40.        self.conv3_1 = ResidualBlock(128, 2)
41.        self.conv3_2 = ResidualBlock(128)
42.
43.        self.conv4_1 = ResidualBlock(256, 2)
44.        self.conv4_2 = ResidualBlock(256)
45.
46.        self.conv5_1 = ResidualBlock(512, 2)
47.        self.conv5_2 = ResidualBlock(512)
48.
49.        self.pool = GlobalAveragePooling2D()
50.        self.fc = Dense(classes, activation="softmax")
51.
52.     def call(self, x):
53.        x = self.conv1(x)
54.        x = self.bn(x)
55.        x = self.relu(x)
56.        x = self.mp1(x)
57.
58.        x = self.conv2_1(x)
59.        x = self.conv2_2(x)
60.
61.        x = self.conv3_1(x)
62.        x = self.conv3_2(x)
63.
64.        x = self.conv4_1(x)
65.        x = self.conv4_2(x)
66.
```


Doi: https://doi.org/10.54216/IJAACI.010203
Received: January 15, 2022 Accepted: May 23, 2022

```
67.        x = self.conv5_1(x)
68.        x = self.conv5_2(x)
69.        x = self.pool(x)
70.        x = self.fc(x)
71.        return x
```

D. Evaluation Indicators

To evaluate the detection performance of our approach, the following evaluation indicators are presented are used:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (3)$$

$$Precision = \frac{TP}{TP + FP} \quad (4)$$

$$Recall = \frac{TP}{TP + FN} \quad (5)$$

$$F1 - measure = 2 * \frac{Recall \times Precision}{Recall + Precision} \quad (6)$$

Doi: https://doi.org/10.54216/IJAACI.010203
Received: January 15, 2022 Accepted: May 23, 2022

4. Experimentations and Results

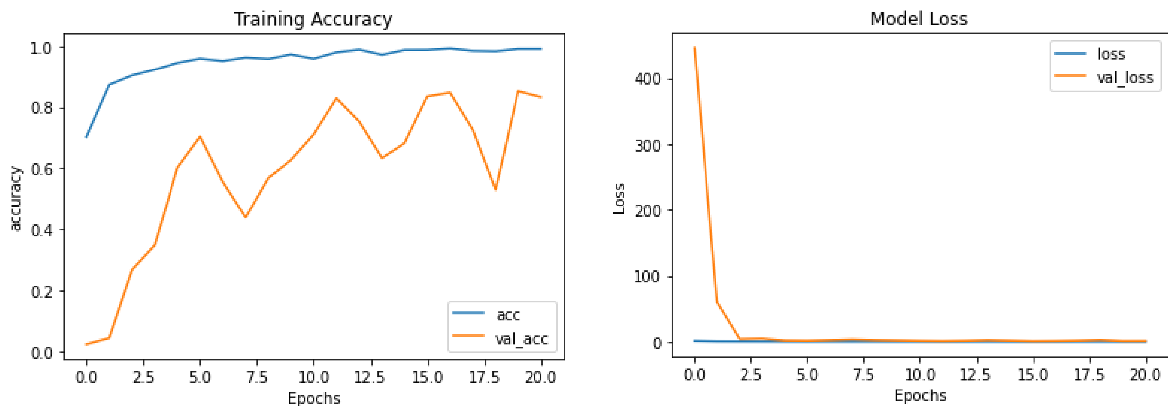


Figure 1: Learning curve analysis for the proposed model

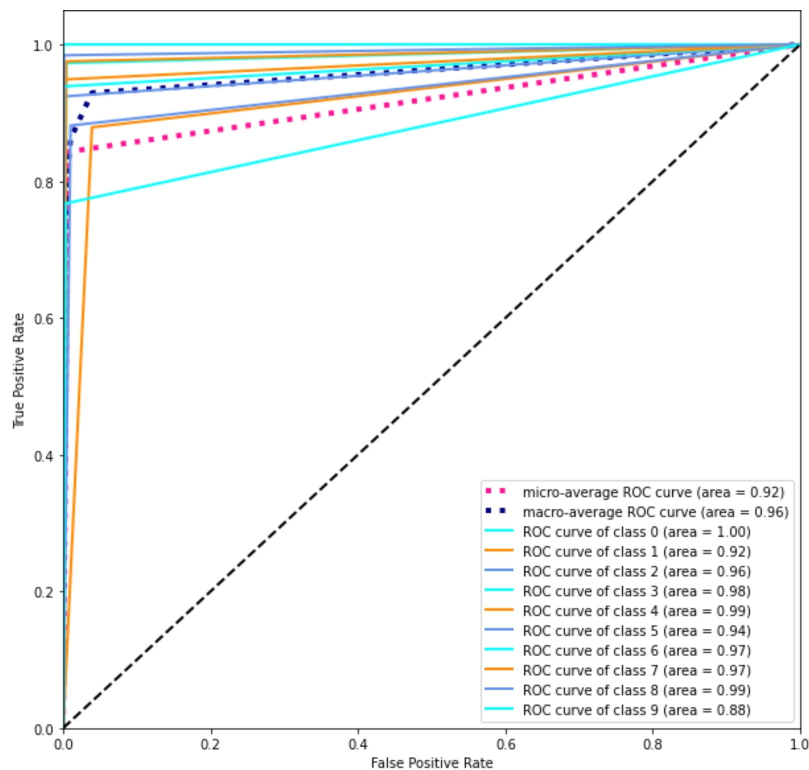


Figure 2: Illustration of the ROC curves for our model.

In this section, we present the experimentation and results of our proposed approach for malware detection using residually connected convolutions. First, we describe the experimental setup, including the hardware and software configurations used for training and testing our models. We then present the results of our experiments, including the performance of our CNN-based approach compared to state-of-the-art techniques.

In our experimental setup, we utilized a Dell workstation configured with a high-performance computing system with a powerful multicore processor (e.g., Intel Core i7 or AMD Ryzen), ample RAM (e.g., 32GB), and a dedicated GPU (e.g., NVIDIA GeForce RTX 1080) with CUDA support. The GPU played a crucial role in accelerating the

Doi: https://doi.org/10.54216/IJAACI.010203
Received: January 15, 2022 Accepted: May 23, 2022

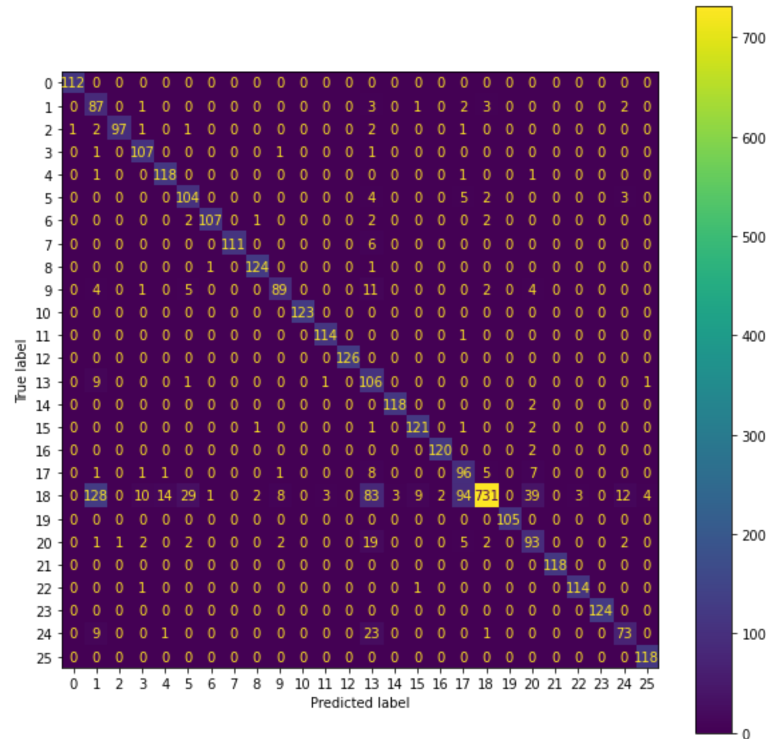


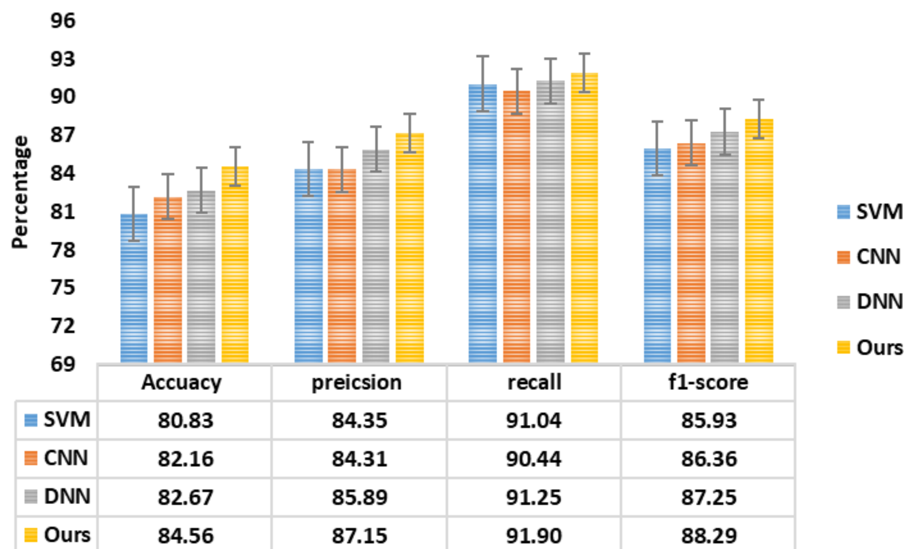
Figure 3. Illustration of the confusion matrix for our model.

training process, especially for deep learning models. For the software configuration, we employed an Ubuntu operating system that provided compatibility with the chosen ML libraries. We used TensorFlow along with its GPU-enabled versions, to take advantage of hardware acceleration.

Learning curve analysis is displayed in Figure 1 to assess the performance and progress of our model during the training process. This can enable us to observe how the model's accuracy and loss evolve as the training data size increases. It could be noted that our model is free from underfitting or overfitting the data. By examining the learning curve, we can make informed decisions that our model is not so complex, but the training data may be insufficient, which implies that additional data is needed to adjust and optimize our model's performance.

In Figure 2, we display the Receiver Operating Characteristic (ROC) to evaluate and visualize the ability of our model to discriminate between different classes of malware. This is achieved by plotting the ROC curve, then examining the trade-off between the true positive rate (sensitivity) and the false positive rate (1-specificity) at various classification thresholds. As shown, we calculate the area under the ROC curve (AUC) for each class to act as a summary measure of the model's performance, with a higher AUC indicating better discriminative power. According to the AUC values, we provide and assess the effectiveness of our model for detecting each class of attacks. Further, the confusion matrix is presented in Figure 3 to evaluate the performance of our model in a classification task. It provides a comprehensive summary of the model's predictions by displaying the counts of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). By examining the confusion matrix, we can find that the results of our model conform to the findings from the ROC analysis.

Doi: https://doi.org/10.54216/IJAACI.010203
Received: January 15, 2022 Accepted: May 23, 2022



	Accuacy	preicsion	recall	f1-score
SVM	80.83	84.35	91.04	85.93
CNN	82.16	84.31	90.44	86.36
DNN	82.67	85.89	91.25	87.25
Ours	84.56	87.15	91.90	88.29

Figure 4: comparison of the performance of different malware detection methods.

In our experiments, we compare the performance of our model against different ML methods for malware detection, where several indicators are considered, as shown in Figure 4. It is notable that the comparison analysis includes three popular ML methods namely standard deep networks, SVM, and standard CNN, upon which, our model can achieve significant improvement across all performance measures. This in turn implies the effectiveness of the proposed model to be used as a general tool to discriminate between different types of malware.

5. Conclusions

This paper presents an applied computational intelligence approach that integrates an intelligent convolutional model for detecting malware encoded into the imaged format. By leveraging the representational power of CNNs, we have demonstrated their ability to capture intricate patterns and representations within malware samples, leading to improved accuracy and robustness in detecting malicious software. Through our experiments and evaluations, we have shown that our model outperforms traditional ML approaches in terms of detection performance. The ability of our model to automatically learn and extract meaningful features from raw data, coupled with their inherent capability to handle complex and high-dimensional input makes them well-suited for the challenging task of malware detection.

Our research contributes to the growing body of knowledge in the field of cybersecurity and showcases the potential of deep learning for combating the ever-evolving landscape of malware threats. The application of computational intelligence in malware detection opens up new avenues for developing more robust and efficient defense mechanisms to protect systems and users from malicious attacks. As future work, we recommend exploring additional aspects, such as the integration of temporal information or the combination of multiple modalities (e.g., API calls, network traffic) with image-based malware detection, to further enhance the capabilities of our model.

References

[1] Liu, Kaijun, et al. "A review of android malware detection approaches based on machine learning." *IEEE Access* 8 (2020): 124579-124607.

[2] Anderson, Hyrum S., et al. "Evading machine learning malware detection." *black Hat* 2017 (2017).

[3] Allix, Kevin, et al. "Are your training datasets yet relevant? an investigation into the importance of timeline in machine learning-based malware detection." *Engineering Secure Software and Systems: 7th International Symposium, ESSoS 2015, Milan, Italy, March 4-6, 2015. Proceedings 7*. Springer International Publishing, 2015.

Doi: https://doi.org/10.54216/IJAACI.010203
Received: January 15, 2022 Accepted: May 23, 2022

[4] Alqahtani, Ebtesam J., Rachid Zagrouba, and Abdullah Almuhaideb. "A survey on android malware detection techniques using machine learning algorithms." *2019 Sixth International Conference on Software Defined Systems (SDS)*. IEEE, 2019.
[5] Sahs, Justin, and Latifur Khan. "A machine learning approach to android malware detection." *2012 European intelligence and security informatics conference*. IEEE, 2012.
[6] Sethi, Kamalakanta, et al. "A novel machine learning based malware detection and classification framework." *2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*. IEEE, 2019.
[7] Demontis, Ambra, et al. "Yes, machine learning can be more secure! a case study on android malware detection." *IEEE transactions on dependable and secure computing* 16.4 (2017): 711-724.
[8] Liu, Liu, et al. "Automatic malware classification and new malware detection using machine learning." *Frontiers of Information Technology & Electronic Engineering* 18.9 (2017): 1336-1347.
[9] Firdausi, Ivan, Alva Erwin, and Anto Satriyo Nugroho. "Analysis of machine learning techniques used in behavior-based malware detection." *2010 second international conference on advances in computing, control, and telecommunication technologies*. IEEE, 2010.
[10] Firdausi, Ivan, Alva Erwin, and Anto Satriyo Nugroho. "Analysis of machine learning techniques used in behavior-based malware detection." *2010 second international conference on advances in computing, control, and telecommunication technologies*. IEEE, 2010.
[11] Rathore, Hemant, et al. "Malware detection using machine learning and deep learning." *Big Data Analytics: 6th International Conference, BDA 2018, Warangal, India, December 18–21, 2018, Proceedings 6*. Springer International Publishing, 2018.
[12] Gavriluţ, Dragoş, et al. "Malware detection using machine learning." *2009 International multiconference on computer science and information technology*. IEEE, 2009.
[13] Narudin, Fairuz Amalina, et al. "Evaluation of machine learning classifiers for mobile malware detection." *Soft Computing* 20 (2016): 343-357.
[14] Li, Jin, et al. "Significant permission identification for machine-learning-based android malware detection." *IEEE Transactions on Industrial Informatics* 14.7 (2018): 3216-3225.
[15] Amos, Brandon, Hamilton Turner, and Jules White. "Applying machine learning classifiers to dynamic android malware detection at scale." *2013 9th international wireless communications and mobile computing conference (IWCMC)*. IEEE, 2013.
[16] Yerima, Suleiman Y., Sakir Sezer, and Igor Muttik. "Android malware detection using parallel machine learning classifiers." *2014 Eighth international conference on next generation mobile apps, services and technologies*. IEEE, 2014.
[17] Peiravian, Naser, and Xingquan Zhu. "Machine learning for android malware detection using permission and api calls." *2013 IEEE 25th international conference on tools with artificial intelligence*. IEEE, 2013.
[18] Kolosnjaji, Bojan, et al. "Adversarial malware binaries: Evading deep learning for malware detection in executables." *2018 26th European signal processing conference (EUSIPCO)*. IEEE, 2018.
[19] Ham, Hyo-Sik, and Mi-Jung Choi. "Analysis of android malware detection performance using machine learning classifiers." *2013 international conference on ICT Convergence (ICTC)*. Ieee, 2013.
[20] Sewak, Mohit, Sanjay K. Sahay, and Hemant Rathore. "Comparison of deep learning and the classical machine learning algorithm for the malware detection." *2018 19th IEEE/ACIS international conference on software engineering, artificial intelligence, networking and parallel/distributed computing (SNPD)*. IEEE, 2018.
[21] Targ, Sasha, Diogo Almeida, and Kevin Lyman. "Resnet in resnet: Generalizing residual architectures." *arXiv preprint arXiv:1603.08029* (2016).

Doi: https://doi.org/10.54216/IJAACI.010203
Received: January 15, 2022   Accepted: May 23, 2022