Malware Detection Using Deep Learning Approaches

Ahmad Shah * and Liqaa Nawaf

Posted Date: 15 July 2024

doi: 10.20944/preprints202407.1214.v1

Keywords: Malware Detection; IoT; Deep Learning; Neural Networks; Long Short-term Memory; Hybrid Model; Central Processing Unit; Graphics Processing Unit; Tensor Processing Unit; Portable Executable

Article

Malware Detection using Deep Learning Approaches

Ahmad Shah * and Liqa Nawaf

Cardiff School of Technology, Cardiff Metropolitan University, Cardiff CF5 2YB, UK; lllnawaf@cardiffmet.ac.uk
* Correspondence: st20255002@outlook.cardiffmet.ac.uk

Abstract: Malware detection is a crucial pillar of cybersecurity and aims to identify and neutralize malware threats. This study investigates how three deep learning algorithms - Convolutional Neural Networks with One-Dimensional Architecture (CNN1D), Neural Networks (NN), and a hybrid model of one dimensional convolutional neural networks (CNN1D) and Long Short-Term Memory (LSTM) - help detect malware in Portable Executable working (.exe) files. Using a large dataset of tagged malware and benign samples, the effectiveness of these algorithms in distinguishing dangerous from harmless executables is extensively tested. In addition, the study evaluates the computational efficiency of the proposed methods for a variety of hardware configurations, including Central Processing Units (CPU), Graphics Processing Units (GPU), Raspberry Pi (IoT device), and Tensor Processing Units (TPU). Performance metrics provide insight into the scalability and effectiveness of the model across different hardware configurations. The results of this study have significant impact on the development of malware detection, particularly in the context of IoT security. By demonstrating the capabilities and limitations of CNN1D, Neural Networks, and CNN1D with LSTM models, the study provides practitioners with efficient and scalable solutions to real-world malware detection difficulties. In addition, insights from performance tests on various hardware platforms provide important assistance in developing robust security measures in IoT contexts, strengthening cybersecurity defenses in an ever-changing digital landscape.

Keywords: malware detection; IoT; deep learning; neural networks; long short-term memory; hybrid model; central processing unit; graphics processing unit; tensor processing unit; portable executable

1. Introduction

Malware is software designed to compromise or breach a protected system without the consent of the legitimate owner. [1] Malware that includes computer applications or packages that pose a system risk can be classified as record infectors or standalone software. Identifying malicious software represents a major cybersecurity challenge, particularly given society's increasing reliance on computers [4]. Today, a single malware incident can cause millions of dollars in damages. Infection-resistant elements provide some protection against malware but are no longer enough to solve the problem. Modern antivirus software uses a flag-based approach, where a flag is a collection of physically created rules designed to identify a limited subset of malware. These criteria are often obvious and cannot reliably identify newly discovered malware, even if it uses a similar capability [6]. This method is not sufficient because millions of new malware tests are discovered every day, and most conditions have unique pairs that have never been observed before. Traditional malware detection methods such as signature-based scanning and heuristic analysis have proven unsuccessful in dealing with the ever-changing environment of malware variants [31]. A study was conducted by R. Perdisci et al [3] in which they introduced a method for classification of packed executables to enhance the accuracy of computer virus detection. The primary drawback of this study was this study focused primarily on packed executables, which are only one aspect of malware obfuscation techniques. Other techniques, such as polymorphism and metamorphism, were not addressed. Advanced detection algorithms that can identify both known and unexpected dangers are urgently needed. Deep learning approaches such as Convolutional Neural Networks (CNN), Neural Networks (NN), and hybrid models have emerged as promising answers. The purpose of this project is to investigate malware detection utilizing three deep learning algorithms, improve system

 doi:10.20944/preprints202407.1214.v1

performance, and assess effectiveness across various computing environments. The goal of the research is to create and apply malware detection algorithms to safeguard digital assets and infrastructure from emerging cyber threats by combining deep learning and detailed hardware inspection. Effective and efficient virus mitigation has long been a goal in the field of information security networks. Enhancing an anti-malware framework, which can examine complex malware, is a widespread initiative that could benefit multiple IT industries [9]. Signature-based tactics, often known as behavioral detection techniques, and anomaly-based strategies can both detect malware [6]. The impetus behind this study stems from the critical need to confront the escalating threat posed by malware in today's digital world. Malicious software, which includes ransomware, spyware, and trojans, is becoming more sophisticated and widespread, posing significant risks to individuals, businesses, and society [6].

Traditional malware detection systems, most of which rely on signature and rule-based techniques, have proven ineffective in identifying and mitigating new malware variants. As malware authors deploy more sophisticated evasion techniques and polymorphic malware strains, the limitations of traditional detection methods become clearer.

Against this backdrop, there are two motivations for this research. The study is based on the idea of the work of Edward Raff [1] with changes for more recent and advanced contributions. This study aims to contribute to the advancement of malware detection technologies through deep learning approaches. Deep learning, with its ability to generate hierarchical representations from data autonomously, has great promise for improving malware detection system accuracy and flexibility. The study also aims to compare the efficacy of CNN1D, NN, and CNN1D and LSTM hybrid models on different architectures for malware detection, identifying strengths and limits. This research not only provides insight into the appropriateness of deep learning models for cybersecurity tasks, but it also lays the path for the development of more durable and effective detection systems for the sustainability of future architectures in IoT. Furthermore, the dataset known as Ember 2018, which contains a large collection of labeled malwares and benign samples, provides a unique opportunity to evaluate the proposed detection system in a real-world environment. The study intends to use this dataset to train and validate deep learning models and analyze their performance in different environments, to bridge the gap between theoretical research and applied implementation in cybersecurity and support the hybrid development of AI and promote cybersecurity, which will also open doors for new developments in the field of IoT and its security. By addressing these motivations, researchers aim to contribute to the development of unique and effective strategies to combat malware attacks, ultimately increasing the security and resilience of digital systems and networks.

The objective of the phases would be to check the accuracy and precision of three Deep Learning based classification algorithms. In addition, these models will also be created to test whether it outweighs the conventional algorithms and detects the nature of PE File (exe file) [1,2]. Based on the results, make recommendations and insights to guide the deployment of the malware detection system in real-world scenarios, and shape future cybersecurity research initiatives and IoT security development. By achieving these research goals, the study aims to contribute to the advancement of malware detection approaches and provide practical insights into the deployment of deep learning-based detection systems in various hardware environments.

2. Background

A file or executable's maliciousness can be determined using a number of different hypotheses. Here are a few of them, along with their shortcomings and areas for development. The following section discusses several completed investigations, along with a critical assessment and a list of drawbacks.

2.1. Literature Review

Zarni Aung and Win Zaw's work [15] is also based on a machine learning method, with the collection of permissions required by executables and APIs being considered as the subject's feature vector. Based on these characteristics, the ML-powered framework identified the file as benign or

 doi:10.20944/preprints202407.1214.v1

malicious. The disadvantage of this approach is that it ignores the majority of critical parameters, such as the quantity of initialised and uninitialized data, which varies with obfuscated code [11], the min-max resource entropy, the count of invalid strings, and the version information. Because a valid executable usually always contains numerous legitimate strings, if this is not the case, there is a significant possibility that the designer has obfuscated the code and is thus attempting to perform some backdoor activity. Rieck et al. [30] proposed a method for automatic analysis of malware behaviour using machine learning techniques, significantly enhancing the ability to detect and categorize malicious software. Another study [16] extracted 189 features from the PE headers and reduced the feature set using various feature selection methods such as Principal Component Analysis, Haar wavelet transformation, and RFR removal before training a J48 decision tree to achieve a maximum accuracy of 97% and a false positive rate of 0.5%. The experiment used a huge dataset of over 10,000 samples, and it took an average of 0.245 seconds to scan an unseen file. Wang et al. [34] proposed an SVM-based discovery technique for detecting inconspicuous PE malware. PE header [5,17] passages were split via static analysis, and the SVM classifier was created based on selected highlights. The grouping approach accurately recognises viruses and worms, however, trojans and indirect accesses have a poorer location accuracy.

The use of several deep learning architectures for malware detection has been the subject of more research in recent years. Convolutional neural networks (CNN), neural networks (NN), and a CNN and Long Short-Term Memory (LSTM) network combination are among these architectures that have been the subject of extensive research. This is a brief evaluation of related recent studies. CNNs have been effectively used in malware detection because of their capacity to automatically learn hierarchical characteristics from raw data. Researchers used 1D convolutional layers to detect temporal trends inside data sequences, such as API requests or malware opcode sequences [19]. These models excel at capturing local patterns, making them ideal for detecting specific features of malware code. Neural networks (NN), including traditional feedforward designs, have been used for malware detection. Studies have focused on employing neural networks to learn complex correlations between various features retrieved from malware samples [1]. The feature learning and representation capabilities of NNs help to effectively classify both known and unknown viruses such as one study by Carlini et al. [24] proposed methods for evaluating the robustness of neural networks against adversarial attacks. In order to classify IoT malware, one conducted research [27] presents a hybrid static classifier that uses bidirectional LSTM and CNN. The method allows for efficient feature extraction without the need for specialised knowledge by analysing Byte and Assembly files as one-dimensional pictures. CNN chooses and extracts features automatically and feeds them into a bidirectional LSTM for classification. Trials using the Microsoft Malware and IoT Malware datasets show better results than with conventional techniques, with average accuracy rates on Hybrid Classifier model were 99.91% and 99.83%, respectively. The study also identifies distinct characteristics that are particular to certain files within the dataset, demonstrating both experimentally and graphically the effectiveness of the suggested method. CNN-LSTM models have been used to assess time-series data such as system call traces or malware samples' dynamic behaviour logs. The combination enables for effective feature extraction and dependency capture over time, which improves the overall performance of malware detection models.

A different study proposes a novel approach for malware detection that employs neural networks to ingest entire executables, displaying strong performance [1]. One op-code study investigates the use of 1D convolutional layers to catch patterns in opcode sequences, demonstrating the effectiveness of CNNs in malware detection [19]. LSTM has previously been used to depict cyber-attack detection in an industrial control system. The study uses a combination of CNN and LSTM layers to detect cyber-attacks, demonstrating the advantages of temporal analysis in malware detection [20]. An additional study investigates deep learning models and their resilience, particularly those for malware detection, which are vulnerable to adversarial assaults in which small changes to input data can result in misclassifications, providing a considerable security risk [23]. One research [21] introduces DeepWAF, a prototype implementation that uses deep learning techniques for online attack detection, in response to the growing concern of cybercrimes attacking web

 doi:10.20944/preprints202407.1214.v1

applications. The efficiency of CNN, LSTM, and their combinational models, CNN-LSTM and LSTM-CNN is methodically investigated in this work. With an average detection rate of about 95% and a false alarm rate of about 2%, experimental findings on the HTTP DATASET CSIC 2010 dataset show good performance across all four detection models. Case studies on false positives and false negatives demonstrate the promising use of machine learning in online attack detection and offer suggestions for future improvements. Another study [22] concluded that models may struggle to generalise to new and previously undetected malware variants (also known as zero-day attacks), particularly when attackers' strategies improve. Further investigation by Yuan et al. [25] proposes a DroidDetector, a deep learning framework for Android malware detection which results in giving an insight into the forms of malware used in a different environment and detection based on deep learning framework.

Traditional security methods face considerable hurdles as malware becomes more sophisticated and evolves at a rapid pace. In recent years, researchers have used deep learning approaches to improve malware detection capabilities. The Ember dataset, which contains a broad collection of labelled malware samples, has become a well-known standard for assessing the performance of deep learning models in this domain. A thorough study [29] utilising two-dimensional binary programme characteristics. This paper investigates the use of deep neural networks for malware detection, emphasising the significance of two-dimensional binary programme features. The study emphasises the power of deep learning in detecting complicated patterns indicative of harmful behaviour [26].

Recent advances in Malware Detection using Deep Learning are briefly discussed providing an insight to different model’s performances. One study [28] investigates the use of recurrent neural networks (RNN) and Long Short-Term Memory (LSTM) networks for malware detection. According to the trial, RNN with Long-Short Term Memory (LSTM) outperformed SVM and Random Forest, two traditional machine learning algorithms, with accuracy rates of 99.70%, 98.55%, and 99.42%, respectively. This is made feasible by RNNs' built-in memory, which can recall several previous states and implicitly extract important characteristics, complicated hidden structures, and complex sequential relationships from data to improve accuracy. These results are also useful for further explorations and developments in Recurrent Neural Networks. Another study by Oyama et al. [32] on Ember Dataset, identified useful features for malware detection in the dataset, but their study also had several limitations because of dataset specific findings such as not covering all new type of potential features for new malware and model dependency.

There are some studies done that emphasis use and significance in the development of Artificial Intelligence or Deep Learning use in Cyber-Security specifically in IoT Environments or Architectures in the recent years. Some are discussed here in this review. A Study done [35] presents a deep learning-based Intrusion Detection system (IDS) solution to as a promising approach for protection. It uses convolutional neural networks (CNNs) and long short-term memory (LSTM) networks in an IDS model, which resulted in high accuracy rate of 98.42%, low loss (0.0275) and a moderate false positive rate of 9.17%, demonstrating its effectiveness in enhancing IoT security. A study [33] was done in which the authors proposed a deep learning approach known as BEFSO-Net, a specialized BERT-based Feed Forward Neural Network Framework designed for IoT security. This model is optimized using the Spotted Hyena Optimizer and achieves outstanding performance metrics, including 97.99% accuracy, 97.96 Matthews Correlation Coefficient, and 97% F1-Score. The research demonstrates BEFSO-Net's effectiveness in addressing the security challenges posed by AI-enabled IoT devices, offering a robust defense mechanism in dynamic decision-making environments. Another study [36] conducted a comparative study on machine learning and deep learning approaches for detecting IoT network intrusion using a dataset with normal and malicious traffic. The suggested approach efficiently detects fraudulent activity in IoT networks by extracting information from network traffic data such as packet size and frequency, the study underscores the importance of feature selection and suggests future research on hybrid models and real-world implementation. In order to identify malware in Internet of Things (IoT) devices, one study [37] presents a hybrid deep learning system that combines long short-term memory networks (LSTMs) with convolutional neural networks (CNNs). To efficiently identify harmful patterns, the system

 doi:10.20944/preprints202407.1214.v1

makes use of the advantages that CNNs have in capturing local spatial correlations and LSTMs' ability to learn long-term relationships. The suggested technique showed excellent detection accuracy, making it a reliable IoT malware detection solution. In one research [38], the efficacy of several machine learning algorithms such as random forests, decision trees, and support vector machines (SVM) in identifying Internet of Things malware was assessed. Real-world IoT datasets were used for the trials, and the algorithms' respective performances were compared. The findings showed that SVM had the lowest detection efficiency and that the random forest method had the greatest accuracy in detection. This study offers insightful information about how to choose the best machine learning algorithms to improve IoT security. Another work [39] investigates deep learning-based techniques that use an ensemble classification strategy that combines CNNs and LSTMs to identify malware in Internet of Things devices. The article addresses current advancements and upcoming research challenges while highlighting how well these algorithms identify complex malware. They also reviewed recent developments in the field, such as recurrent neural networks (RNNs) and generative adversarial networks (GANs).

These recent efforts highlight the importance of deep learning techniques, such as CNNs, NNs, and hybrid architectures, in advancing the field of malware detection. The use of temporal analysis and the capacity to automatically learn features from raw data have proven critical in improving the accuracy and durability of malware detection algorithms. As a result, using three distinct deep learning algorithms for static analysis of malware and benign will seek to provide insight into the effectiveness of these approaches as well as the differences in executional analysis of a set of technologies (CPU and GPU, for example), to add a large quantity of labour progress in order to contribute and create something valuable for the development of Secure IoT Infrastructures as well as advancements within the Cyber-Security Domain. Malware evaluation techniques use static and dynamic methods to detect malware signatures [7]. Static methods inspect code and header statistics without executing the malware, while dynamic methods run the suspicious executable in a controlled environment [10]. However, static methods cannot detect zero-day attacks [8], and dynamic methods can outwit sandboxed environments [10]. To improve reliability, in this study that has been conducted, three Deep Learning methodologies (CNN1D, NN, and a hybrid model approach) can be used for ideal decision-making and comprehensive understanding of performance in different environments.

3. Technical Approaches

The work would be split into two phases as per the proposed design [12,18]. Ember Dataset 2018 Features version 2 is obtained in the first step, and three models are trained on the dataset. Cross-validation is typically done at this step as well. The second phase will involve converting the raw bytes to features data by extracting all of the typical PE file structure data [5,17] from the input PE file (exe file). executing the prediction for a PE file by calling a saved model in accordance with the selection to handle real-world validation and determine if a file is harmful or not. Following this phase, many configurations will be analyzed to address the issue of performance on various configurations, including CPU, GPU, TPU, and Raspberry Pi as an IoT based edge-computing device.

3.1. Data Collection

Dataset: A dataset is essentially a grouping of data points that a computer may process and analyze as a single entity for forecasting and expectation-related reasons. This suggests that since a machine cannot see information in the same way as a human, the information collected must be made consistent and justified. In this study, we used an open publicly available dataset known as Ember Dataset [12].

Ember Dataset: The EMBER dataset, a benchmark for scientists, comprises items from Portable Executables (PE) [5] records. It includes elements from 1.1 million PE records filtered before 2017, and highlights from 1 million PE documents analyzed before 2018. This archive makes it simple to reproducibly prepare the benchmark models, expand that gave highlight set, or characterize new PE files with the benchmark models [12]. The EMBER dataset is composed of JSON line files with one

 doi:10.20944/preprints202407.1214.v1

object per line, each containing a unique sha256 hash, coarse time information, a label (0 for benign, 1 for malicious, or -1 for unlabeled), and eight groups of raw features (parsed values and format-agnostic histograms). These data types provide a unique identification of the file. For convenience our dataset is comprised of raw features that are human readable. Ember provided code that produces from raw features a numeric feature vector required for model building. This allows researchers to decouple raw features from the vectorizing strategies. The LIEF project is a tool that extracts features from PE files within the EMBER dataset, converts them to JSON format, and includes them in the dataset. Vectorized features can be created from raw features and saved in binary format, which can be translated to CSV, data frame, or other formats.

The EMBER dataset consists of eight sets of raw features, including parsed features, histograms, and string counts. It is distinguished from model features, which are vectorized features derived from the dataset size used to prepare a model. The raw highlights are captured using feature hashing technique, including strings, imported names, and sent out names [12].

This dataset consists of 1.1 million samples labelled and mixed as malwares and benign. 2351 features have been extracted from the dataset for feature processing. Byte Histogram, Byte Entropy Histogram, Section Information, Imports Information, Exports Information, General Information, Header Information, Raw Byte Strings, and other pertinent aspects of sample files in the dataset are the parameters utilized to extract features.

3.2. Environmental Setup:

It is crucial to utilize specific procedures and protocols inside the environmental framework of this investigation. In order to conduct the study effectively and obtain the intended outcomes, a suitable setting with appropriate circumstances must be established. The setup parameters ought to be standard for any deep learning environment; however, the Python version ought to exceed Python version 3.7, TensorFlow version should be 2 or above, and Lief version should be 0.10.1 or higher.

While there are differences in system requirements for adopting a hybrid model, main memory segments supported by nmaps typically demand a minimum of 16 GB of RAM or greater allocation. Download the latest version of the feature extractor and the Ember Dataset from ember's git repository [12] in order to utilize it. To experiment with GPUs and TPUs, Google Colab's Service was used. Utilizing a Raspberry Pi 4 Model B, this study examined how an edge computing device can be used to help grow the security in the domain of Internet of things.

3.3. Model Architecture:

It is necessary to construct the model architectures for training, testing, and prediction. The architectures are described in brief below:

CNN1D Model Architecture:

The original Malware Detection model suggested in this paper is based on [18] a one-dimensional convolutional neural network (CNN1D) architecture. This model is composed of sequential layers, beginning with a convolutional layer with 128 filters, a kernel size of 64, a stride of 64, and a rectified linear unit (ReLU) activation function. Batch normalization is then used to standardize the output values, followed by data aggregation using another convolutional layer with a kernel size of 3 and a stride of 2. Batch normalization is used again to normalize the batch values, and then the data is flattened with a flattening layer. The model then incorporates dense neural networks [13] with 256 activations, which use the ReLU activation function. The architecture of this model is depicted in Figure 1.

This model architecture is intended to successfully analyze malware data in a one-dimensional context, with convolutional layers extracting features and dense layers determining categorization. The sequential structure of layers improves information flow via the network, allowing the model to learn and distinguish between harmful and benign data. The use of ReLU activation functions adds nonlinearity to the model, increasing its ability to capture complicated patterns in the data. Overall,

 doi:10.20944/preprints202407.1214.v1

this model setup provides a strong foundation for malware detection tasks by combining 1D CNN architecture and deep learning approaches.

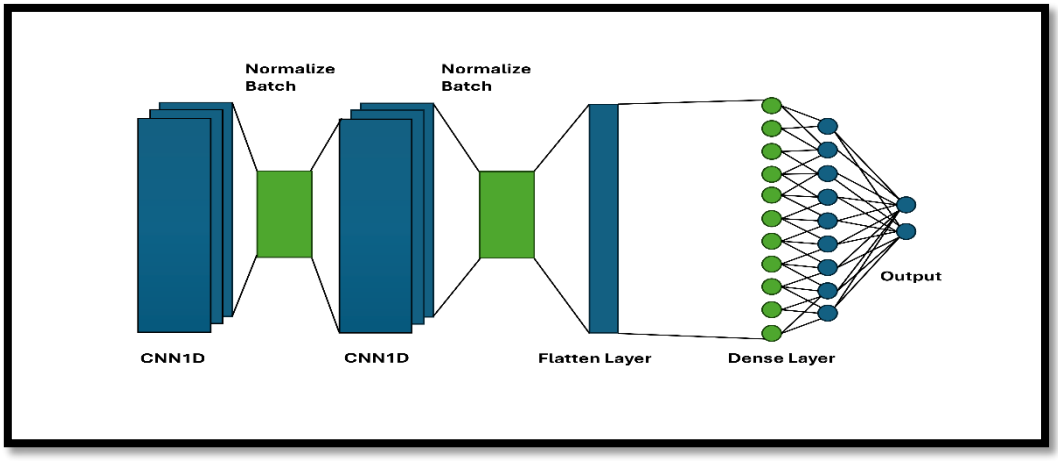

Figure 1. 1D CNN Architecture.

CNN1D with LSTM Model Architecture:

In terms of Malware Detection, the statement describes the design and architecture of a hybrid model for detecting malicious software. This model is based on previously used One Dimensional Convolutional Neural Networks (CNN) with Long Short-Term Memory (LSTM) architecture, a type of Recurrent Neural Network (RNN), to exploit both spatial and sequential information in the data.

The suggested model below in Figure 2 starts with a CNN1D layer made up of 128 filters, each with a kernel size of 64 and a stride of 64, and employs the Rectified Linear Unit (ReLU) activation function. This layer is followed by batch normalisation, which standardises the input data. The output is then further processed by a dense neural network layer with 256 units. The model then uses LSTM units to capture sequential patterns in the data. This is performed with a layer of 64 LSTM units and a two-dimensional input. Again, batch normalisation is used to ensure that the data is scaled consistently. Following that, more dense neural network layers with sizes of 128, 256, and 128 units are implemented, each of which uses the softsign activation function. The model closes with an output layer that consists of two outputs classified using the categorical crossentropy loss function and a sigmoid activation function. This setup enables the model to categorise input data, making it easier to identify malware.

In essence, the hybrid model combines CNN1D and LSTM architectures to efficiently analyse both spatial and sequential data, improving its capacity to detect and classify malware.

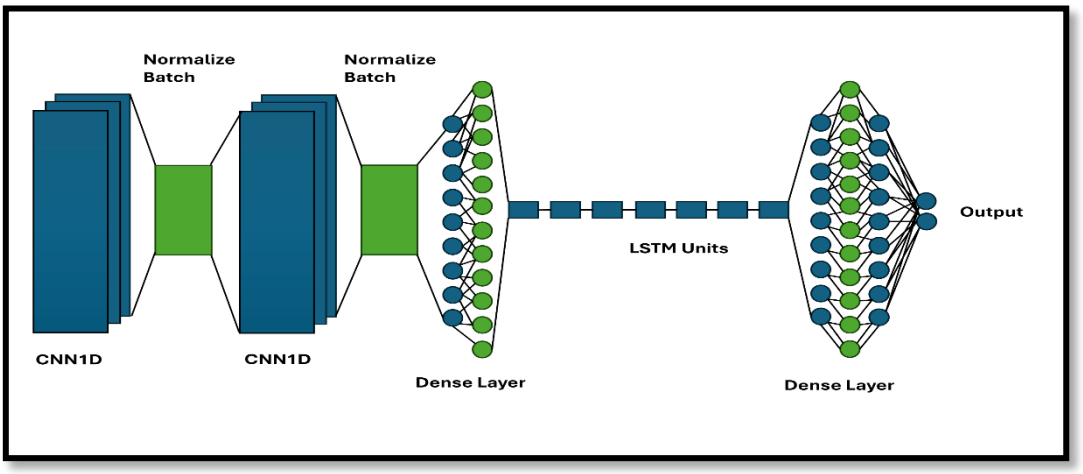

Figure 2. CNN1D-LSTM Model.

 doi:10.20944/preprints202407.1214.v1

Neural Networks Model Architecture:

In the field of Malware Detection, the statement describes the design and implementation of a simple Neural Network architecture as a model for detecting malicious software. This model as described below in the Figure 3, is built as a sequential layer, with numerous layers of interconnected neurons dedicated to processing and analyzing input data. The initial layer, known as the input layer, is located at the top of the sequential architecture and consists of 2351 vectorized inputs. These inputs represent the attributes or characteristics of the malware samples under investigation. The Rectified Linear Unit (ReLU) activation function is used in this layer to introduce nonlinearity and enhance signal propagation through the network.

Following the input layer, the model has two hidden layers, each with a set number of units. The first concealed layer has 64 units, while the second layer is decreased to 32. These hidden layers are used to extract and alter characteristics learned from input data, thereby reducing the dimensionality of the information processed by the model. The model then incorporates dense neural network layers of decreasing sizes (16, 8, and 4 units, respectively). These layers enhance the extracted characteristics, improving the model's capacity to detect malware-related patterns. Similar to the input layer, the ReLU activation function is applied within these layers to introduce non-linearity and facilitate feature extraction.

Finally, the model closes with an output layer that employs the sigmoid activation function. This layer generates the model's output, which is a binary classification of the presence or absence of malware. The sigmoid activation function ensures that the result is between 0 and 1, indicating the likelihood of the input sample being classed as malware.

Overall, the Neural Network architecture in Figure 3 described in the statement provides an organized approach to malware detection, with progressive layers of neurons processing incoming data and making informed classifications based on learnt properties.

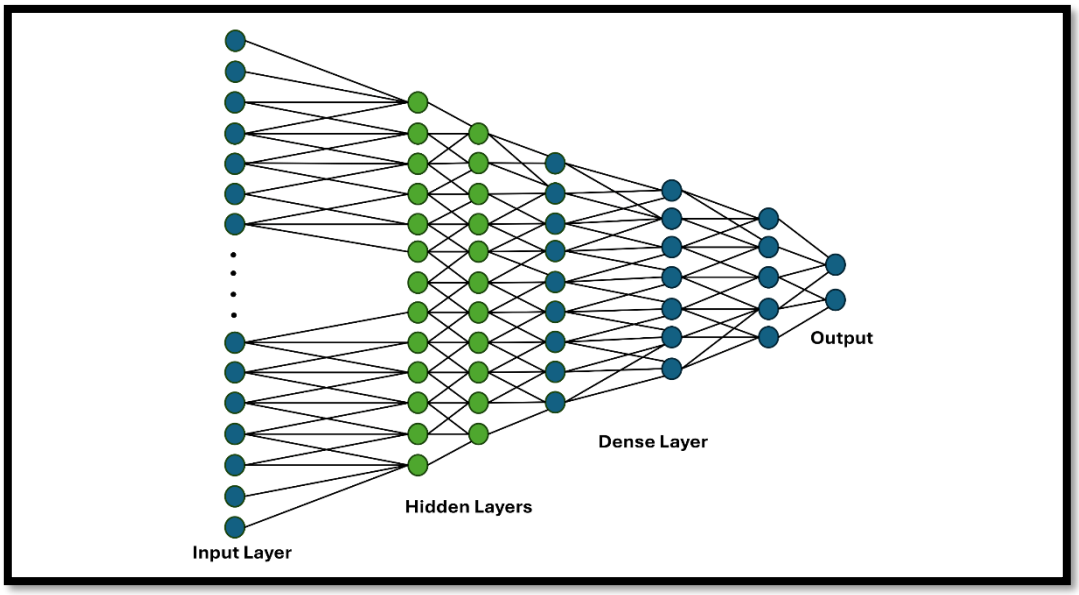

Figure 3. Neural Network Architecture.

Summarizing the discussions above, the data gathering phase consisted of extracting and preparing dataset details from JSON object files using Ember's native functionalities. The dataset was rigorously organized into different segments for training, validation, and testing to enable accurate model performance assessment. Following that, the model architectures were fully built, three separate approaches: a 1-Dimensional Convolutional Neural Network (CNN1D), a hybrid CNN1D with Long Short-Term Memory (LSTM), and a standard Neural Network (NN) architecture. Each model was methodically set up with specified parameters and layers to improve its effectiveness in virus detection activities. Moving forward, the experimental stage will include rigorous testing and

 doi:10.20944/preprints202407.1214.v1

assessment of these models against the segmented dataset. Performance criteria such as accuracy, precision, recall, and F1-score will be used to evaluate each model's ability to distinguish between benign and malicious executables. In addition, the computational efficiency of the models will be assessed across various hardware configurations, such as CPUs, GPUs, TPUs, and Raspberry Pi. The insights gained from these studies will help us refine the models and advance malware detection using deep learning methodologies.

4. Experimentation Results

This section introduces the implantation and results of the experiments.

4.1. Implementation

Ember's proprietary functions were utilized to retrieve the dataset details or the features from executables stored in JSON object files throughout the training phase. The dataset was then divided into various subsets for training, validation, and testing purposes. After dataset segmentation, the model data and weights were retained for further review and testing. Following the completion of the data extraction and preparation operations, the following steps were carried out in sequence. To begin, significant features were picked by partitioning the data into X and Y coordinates, resulting in separate sets for training, validation, and testing to reduce the danger of overfitting. The chosen model was then trained on this curated dataset to assess a variety of performance indicators, including accuracy and other relevant outcomes. Validation testing was done iteratively during the training phase, with data fitting across each epoch. The model with the highest validation accuracy was found, and its associated data and weights were preserved according to preset criteria. (See Appendix B: Figure A1, Figure A2)

Ultimately, the estimated accuracy of the data on the selected model architecture was the outcome of the training procedure. The preprocessing and training methods remained the same for all environments, with the exception of GPU and TPU, which were utilized on Google Colab Servers.

In order to gather pertinent data for testing detection, Ember's proprietary feature extractor was used to extract an executable file's header details. The code appropriately processed the scaled data, which complied with the scaling criteria of the model. Many tools, such as PE readers and Ember's own Feature Extraction Library, helped with this process. Following the extraction of raw bytes and other pertinent data, the following steps were carried out in order:

Loading the scaler data and saved model data to prevent the extracted raw bytes from being overfitted.

- Making predictions with the model using the raw properties extracted from the PE file. (See Appendix B: Figure A3).

With all factors considered, these implementation stages made sure that the malware detection models were appropriately trained and tested, enabling precise estimations and assessments of their effectiveness in identifying malicious software. For all models and devices, the same process as previously mentioned was followed.

4.2. Results

Training, validation, and testing were all part of the deep learning-based malware detection models' implementation phase. After that, the experimental phase gave way to a critical analysis and evaluation phase when their performance was evaluated on a range of hardware platforms. In order to evaluate the models' precision in identifying and categorizing harmful executable files, data was gathered that included accuracy metrics, validation outcomes, and execution times on different computing machines.

After the experimental phase, the collected data is examined to draw appropriate deductions regarding the effectiveness and practicality of malware detection models based on deep learning across a range of hardware platforms and configurations. The initial analysis pertains to performance-based testing, as detailed in Table 1.

 doi:10.20944/preprints202407.1214.v1

Table 1. Performance Results.

Model	CPU	GPU	TPU	Raspberry Pi	Epochs	Batch Size
NN	3 mins 24 seconds	2 mins 39 Seconds	**1 min 17 seconds**	4 mins 36 seconds	10	64
CNN1D	35 min 27 Seconds	4 mins 26 Seconds	**1 min 14 seconds**	42 mins 2 Seconds	10	64
CNN1D + LSTM	21 mins 1 seconds	2 mins 27 seconds	**1 min 25 seconds**	29 mins 20 seconds	10	1000

As illustrated in Table 1, the performance of the NN model was tested on a variety of hardware combinations, including CPU, GPU, TPU, and Raspberry pi. The experiment had 10 epochs and a batch size of 64. On CPU, the Neural Networks model trained in 3 minutes and 24 seconds. With GPU utilization, training time was lowered to 2 minutes and 39 seconds. TPU displayed outstanding efficiency by completing training in 1 minute and 17 seconds. Raspberry Pi had the slowest performance, with training taking 4 minutes and 36 seconds. Overall, the results reveal that the Neural Networks model performs differently across hardware platforms, with TPU providing the most efficient training time. The CNN1D model's training time differed dramatically among hardware platforms. Using a CPU, training took 35 minutes and 27 seconds. With GPU acceleration, training time was reduced to 4 minutes and 26 seconds. The TPU had the fastest training time of 1 minute and 14 seconds. Training on a Raspberry Pi took the longest, 42 minutes and 2 seconds. These findings highlight the importance of hardware acceleration, with TPUs offering the most efficient performance when training the CNN1D model. The CNN1D + LSTM model's training times varied depending on the hardware platform. Training with a CPU took 21 minutes and 1 second. With GPU acceleration, training time was reduced to 2 minutes and 27 seconds. The TPU had the fastest training time of one minute and 25 seconds. Training using a Raspberry Pi device takes 29 minutes and 20 seconds. These findings underscore the role of hardware acceleration in training efficiency, with TPUs outperforming the CNN1D + LSTM model.

Across all models, TPUs consistently outperformed GPUs and CPUs. The Raspberry Pi consistently performed the slowest of the evaluated combinations. The CNN1D + LSTM model outperformed the other models in terms of training time on all hardware platforms.

Furthermore, the analysis of accuracy and detection is also a part of this study ensuring the model performance in system-specific architectures as well as the detection rate. The results discussed are demonstrated in Table 2.

Table 2. Analysis of Accuracy based on Different Devices.

Device	Models	Accuracy	Validation Accuracy	Testing Accuracy (exe Files)
CPU	NN	**99.25%**	**98.82%**	**0.9%**
	CNN1D	**98.89%**	98.52%	0.8%
	CNN1D + LSTM	98.77%	**98.56%**	0.8%
GPU	NN	**99.87%**	**98.79%**	**0.9%**
	CNN-1D	95.94%	95.54%	0.7%
	CNN1D + LSTM	**98.86%**	**98.61%**	0.8%
TPU	NN	**99.91%**	**98.12%**	**0.9%**
	CNN1D	**98.89%**	**98.59%**	0.8%
	CNN1D + LSTM	98.81%	98.56%	0.7%
Raspberry Pi	NN	**97.29%**	**97.02%**	**0.6%**
	CNN1D	**96.95%**	95.21%	0.5%
	CNN1D + LSTM	96.14%	**95.23%**	0.6%

 doi:10.20944/preprints202407.1214.v1

As indicated by the results presented in Table 2,, The analysis is based on different setup environments to see the performances of different deep learning approaches. The performance of several models on a CPU device was assessed using accuracy measures for neural networks, CNN1D, and CNN1D plus LSTM models. The Neural Networks model obtained 99.25% accuracy, with a little lower validation accuracy of 98.82%. The testing accuracy for exe files was rated as 90%. The CNN1D model attained an accuracy of 98.89%, while validation accuracy was 98.52%. However, the testing accuracy for exe files was slightly lower (80%). Similarly, the CNN1D + LSTM model achieved an accuracy of 98.77%, with a validation accuracy of 98.56%. Testing accuracy for exe files was 80%, which was consistent with the CNN1D model.

The previously constructed models were then tested on a GPU device. The Neural Networks model had the highest accuracy (99.87%), with a validation accuracy of 98.79%. The testing accuracy for exe files was rated as 90%. The CNN1D model obtained 95.94% accuracy, with a slightly lower validation accuracy of 95.54%. Test accuracy for exe files were significantly lower, at 70%. The CNN1D + LSTM model achieved an accuracy of 98.86%, while validation accuracy was 98.61%. Testing accuracy for exe files was 80%, which was consistent with the CNN1D model. In conclusion, the Neural Networks model surpassed the others in terms of accuracy on the GPU device, followed by the CNN1D + LSTM model. The CNN1D model demonstrated decreased accuracy compared to the other two models.

The performance of the three models was assessed on a TPU device. The Neural Networks model had the highest accuracy (99.91%), with a validation accuracy of 98.12%. The testing accuracy for exe files was rated as 90%. The CNN1D model obtained 98.89% accuracy, with a slightly better validation accuracy of 98.59%. The test accuracy for exe files was 80%. The CNN1D + LSTM model achieved an accuracy of 98.81%, while validation accuracy was 98.56%. Test accuracy for exe files were slightly lower, at 70%. In conclusion, the Neural Networks model achieved the highest accuracy on the TPU device, followed by the CNN1D and CNN1D + LSTM models. However, all three models obtained great accuracy rates, demonstrating the efficacy of TPUs for deep learning.

All three models were also tested on a Raspberry Pi. The Neural Networks model had an accuracy of 97.29%, with a validation accuracy of 97.02%. The testing accuracy for exe files was rated as 60%. The CNN1D model obtained 96.95% accuracy, with a little lower validation accuracy of 95.21%. The test accuracy for exe files was 50%. The CNN1D + LSTM model achieved an accuracy of 96.14%, while validation accuracy was 95.23%. The testing accuracy for exe files was 60%, which is consistent with the Neural Networks model. In conclusion, all three models have relatively high accuracy rates on the Raspberry Pi device. The Neural Networks model outperformed the CNN1D and CNN1D + LSTM models in terms of accuracy and validation accuracy. However, the testing accuracy for exe files was consistent across all models.

Training times differed dramatically between hardware setups. CPUs often have lengthier training times than GPUs, TPUs, and Raspberry Pi devices. TPUs consistently had the fastest training times, followed by GPUs, with Raspberry Pi devices having the longest training times. Neural networks consistently had the highest accuracy rates across all hardware platforms, followed by CNN1D and CNN1D + LSTM models. However, accuracy rates varied among device combinations. The testing accuracy of exe files was consistently good across all models and hardware platforms. However, there were disparities in accuracy rates for validation datasets, with neural networks frequently outperforming CNN1D and CNN1D + LSTM models.

4.3. Improvisation Techniques

To enhance training efficiency and accuracy, numerous strategies can be used:

- Hardware Optimization: Using hardware accelerators like GPUs and TPUs can dramatically cut training times.
- Algorithm Optimization: Fine-tuning hyperparameters, optimizing network designs, and using advanced optimization techniques can all increase model performance.
- Data Augmentation: Increasing the diversity and volume of training data using strategies like data augmentation can improve model generalization and accuracy.

 doi:10.20944/preprints202407.1214.v1

- Ensemble Learning: Bringing together predictions from several models or model modifications might increase overall performance and robustness.
- Transfer Learning: Using pre-trained models or features from big datasets can speed up training and increase performance, particularly in settings with limited training data.
- Device specific Model Development: Specific Models for raspberry pi, CPU, GPU, and TPU should be developed which must be consistent on all platforms based on collective performance.

Implementing these improvisation strategies can improve the models' training efficiency and accuracy, resulting in more effective malware detection systems also ensuring safeguarding of IoT Infrastructures.

4.4. Discussion

The study compared three malware detection models: Neural Networks (NN), 1-Dimensional Convolutional Neural Networks (CNN1D), and a hybrid model that combines CNN1D with Long Short-Term Memory (CNN1D+LSTM). These models were tested on a variety of devices, including CPUs, GPUs, TPUs, and Raspberry Pi. The findings show differences in accuracy, validation accuracy, and testing accuracy among models and devices. The findings are consistent with prior research demonstrating the efficiency of deep learning models in malware detection. Neural Networks have demonstrated promising outcomes due to their capacity to detect complicated patterns in data. CNN1D models have demonstrated success in identifying spatial dependencies in sequential data, making them useful for analyzing malware executables. Furthermore, the inclusion of LSTM in the hybrid model tries to capture temporal dependencies, potentially improving detection accuracy.

The results show that malware detection system performance is significantly impacted by the model design and device selection. Additional research could look into optimization strategies to boost the effectiveness of model inference and training across a variety of hardware platforms. Additionally, researching how model performance is affected by hyperparameters like batch size and epochs may provide helpful information for improving detection accuracy. The study's findings are especially pertinent when discussing the security of the Internet of Things. Malware attacks on IoT devices are becoming more likely as the number of connected devices rises. Therefore, it is essential for IoT eco-system security to design effective and efficient malware detection algorithms that are compatible with IoT devices with limited resources, like Raspberry Pi [14]. The potential to enhance model-device compatibility in Internet of Things scenarios is demonstrated by the enhanced performance of certain models on particular devices, as TPU for Neural Networks.

Potential avenues for further exploration and improvement in the field of deep learning algorithms for malware detection could be explored in future study. These potential avenues offer fascinating opportunities to strengthen malware detection systems and cybersecurity.

Researchers could research and construct novel deep learning architectures designed exclusively for malware detection. This could entail experimenting with variants on current models or developing totally new architectures capable of efficiently capturing malware's complicated properties. The efficacy of ensemble learning approaches in integrating different deep learning models for better malware detection could be investigated. Bagging, boosting, and stacking techniques have showed promise for improving classification accuracy by utilizing the variety of individual models. The utility of transfer learning approaches in malware detection might be examined, with pre-trained models fine-tuned on specific malware datasets. Transfer learning also offers the ability to address the issues of limited labelled data and speed up the training process for new detection tasks. Research could focus on adversarial assaults and defenses in the context of deep learning-based malware detection. The robustness of deep learning models against adversarial attacks aimed to elude detection should be investigated, as well as the creation of effective defense mechanisms to prevent such attacks. It is also critical to develop real-time malware detection systems that can rapidly analyze incoming data streams for harmful activities. Techniques for optimizing model inference speed and scalability to suit real-world deployment requirements in high-speed network environments could be investigated. The use of Explainable AI (XAI) approaches may improve the interpretability and transparency of deep learning models for malware detection.

 doi:10.20944/preprints202407.1214.v1

Methods for creating human-readable explanations of model predictions could help cybersecurity analysts better comprehend and validate detection results. It is critical to conduct thorough evaluations of deep learning-based malware detection systems in real-world cybersecurity environments. Collaborations with industry partners could help to deploy and validate created models for detecting emerging malware threats and evolving attack scenarios. Addressing ethical and privacy concerns related with the implementation of deep learning-based malware detection systems is critical. Frameworks for the responsible and ethical use of AI technologies in cybersecurity applications should be investigated. Additional Future Research in IoT Impact Studies: More research on malware detection using deep learning in the context of IoT (Internet of Things) could provide useful insights. Researchers could investigate the particular problems and security implications of installing deep learning models on resource constrained IoT devices. Furthermore, research on the influence of malware detection on the overall security and dependability of IoT ecosystems may provide useful guidance for future IoT development and deployment strategies.

By following these possibilities for future study, scholars and practitioners can help to advance the state-of-the-art in malware detection utilizing deep learning techniques. Finally, these initiatives can help to improve the security posture of digital systems and networks against evolving cyber threats, secure key infrastructure, and protect user privacy and data integrity.

Finally summarizing the discussion, comparing different malware detection methods on different devices reveals important information about their performance. The study emphasizes the need of selecting appropriate model architectures and hardware platforms to achieve the highest detection accuracy. Moving forward, continued study in this field is critical for improving cybersecurity measures, especially in the context of IoT deployments.

5. Conclusions

In this study, investigation of the use of deep learning algorithms for malware detection took place, which is a vital part of cybersecurity in the digital age. By this study, vital insights have been discovered into the usefulness and performance of these techniques in identifying malicious software by conducting a thorough investigation of numerous models and architectures, as well as implementing them on various hardware platforms.

The study began with the creation of three unique models: a neural network (NN), a 1-dimensional convolutional neural network (CNN1D), and a hybrid CNN1D + long short-term memory (LSTM) model. Each model was trained and tested on datasets containing features retrieved from executable files, with the purpose of correctly recognizing malware cases. The results of these experiments indicated significant differences in performance among hardware platforms. When run on CPUs, GPUs, and tensor processing units (TPUs), each model demonstrated varied degrees of accuracy and efficiency. Furthermore, the viability of deploying these models was investigated on resource-constrained devices like Raspberry Pi, emphasizing the significance of optimizing model architectures for such contexts.

First, the models' performance varied greatly depending on the hardware platform chosen for execution. While CPUs showed decent execution rates, GPUs and TPUs performed faster, with TPUs demonstrating significant efficiency increases. In contrast, execution on resource-constrained devices such as the Raspberry Pi resulted in increased processing times, emphasizing the necessity for optimization solutions targeted to such contexts. Furthermore, the accuracy metrics of the models, as tested by validation and testing on executable files set, indicated subtle discrepancies between the designs. The Neural Network model achieved good accuracy across all devices, but the CNN1D model demonstrated lower accuracy but shorter execution times. The hybrid CNN1D + LSTM model produced encouraging results, achieving a balance of accuracy and efficiency.

This study examined the importance of the findings in light of past research and working theories. By combining the findings of the experimentation, gained better knowledge of the capabilities and limitations of deep learning-based malware detection systems. The findings highlight the significance of ongoing research and development in this area in order to remain ahead of evolving cyber threats. In the future, there are numerous areas for research that warrant

 doi:10.20944/preprints202407.1214.v1

exploration. Novel architectures designed expressly for malware detection, ensemble learning techniques, and transfer learning methodologies all offer potential opportunities to improve detection accuracy and robustness. Furthermore, deploying real-time detection systems, adding explainable AI (XAI) approaches, and resolving ethical and privacy concerns are critical areas for future research. Future research should not only advance the state-of-the-art in malware detection, but also examine the implications of deep learning in IoT environments. Understanding the specific problems and security consequences of installing deep learning models on IoT devices is critical for securing linked systems and protecting user privacy.

Finally, the collective findings of this study contribute to advancing the understanding of deep learning-based malware detection and underscore the need for ongoing research to address evolving cyber threats effectively. This is to inspire continuing inquiry and innovation in this critical area of cybersecurity by drawing on insights from our experiments and discussing future research options.

Author Contributions: Conceptualization, A.S. and L.N.; methodology, A.S.; software, A.S.; validation, L.N., and A.S.; formal analysis, L.N. and A.S.; investigation, A.S.; resources, A.S.; data curation, A.S.; writing—original draft preparation, A.S. and L.N.; writing—review and editing, L.N. and A.S.; visualization, A.S.; supervision, L.N.; project administration, L.N.; funding acquisition, L.N. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by CMet, Cybersecurity and Information Networks Centre (CINC).

Data Availability Statement: Dataset link: https://ember.elastic.co/ember_dataset_2018_2.tar.bz2, CNN-1D Architecture Link: https://github.com/tamnguyenvan/malnet/tree/master

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A: ROC Curve of CNN1D

The true positive and false positive rates are depicted in the above figure, which grows gradually in accordance with accuracy. The false positive rate exhibits a slight oscillation towards false positives before becoming straight, with a false positive rate ranging from 0.0 to 0.2 and a true positive rate from 0.8 to 1.0.

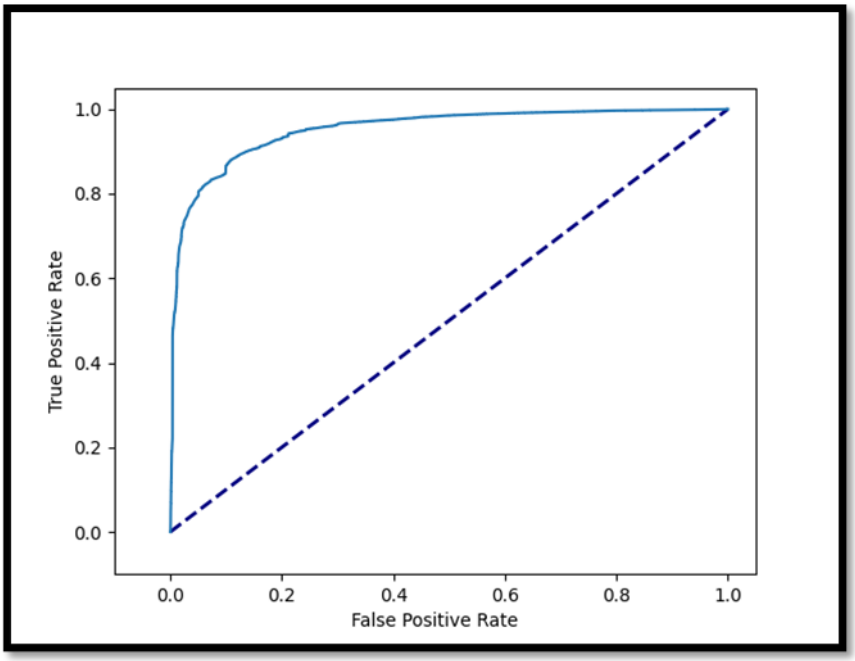

Figure A1. ROC Curve describing True Positives and False Positives.

 doi:10.20944/preprints202407.1214.v1

Accuracy Results of CNN1D:

The training procedure and accuracy results are displayed in the figure below, which demonstrates how the accuracy results improved to 0.965 on the final epoch, reducing the loss.

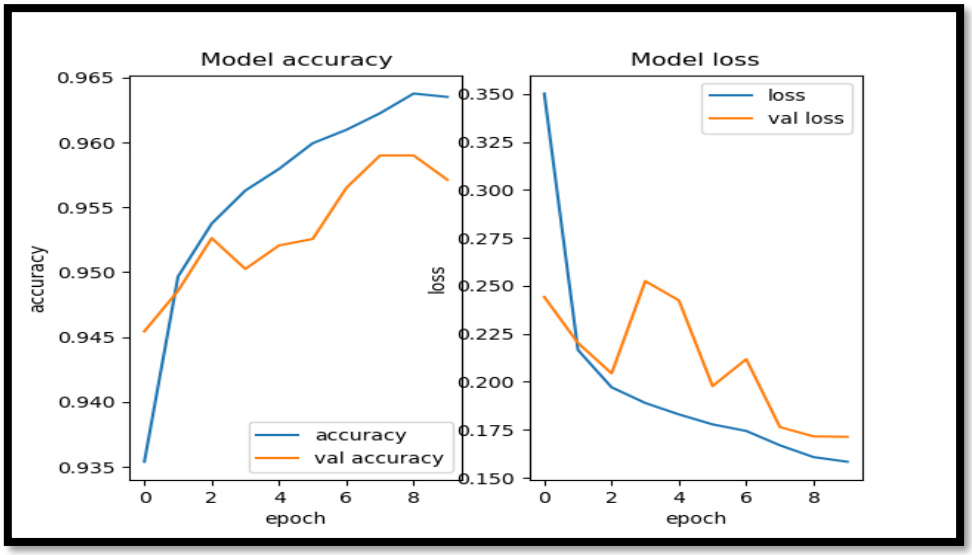

Figure A2. Accuracy results of CNN1D.

ROC Curve of CNN1D-LSTM:

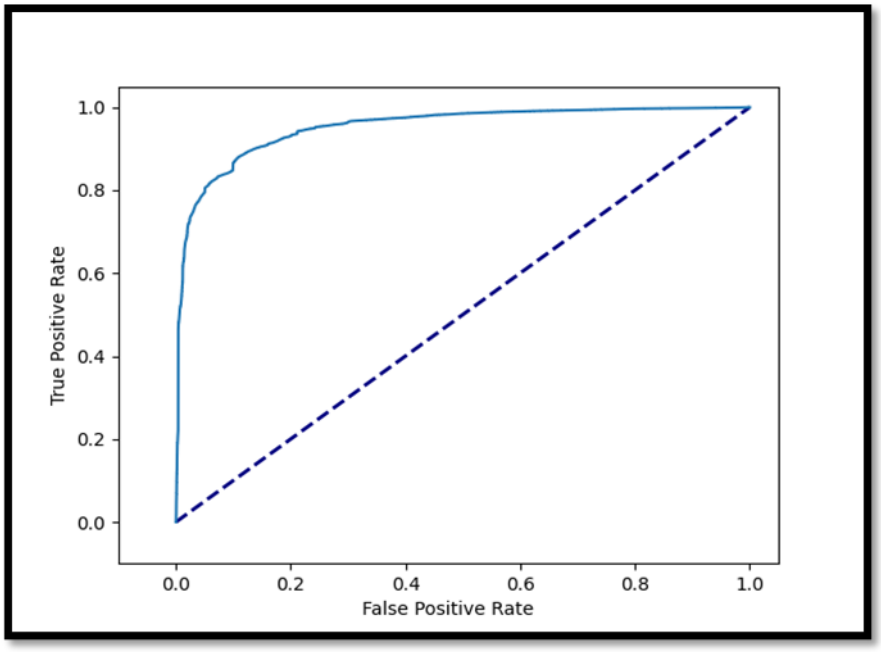

Figure A3. ROC Curve describing True Positives and False Positives.

The true positive and false positive rates are depicted in the above figure, which grows gradually in accordance with accuracy. The false positive rate exhibits a slight oscillation towards false positives before becoming straight, with a false positive rate ranging from 0.0 to 0.2 and a true positive rate from 0.8 to 1.0.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 July 2024 doi:10.20944/preprints202407.1214.v1

Accuracy Results of CNN1D-LSTM:

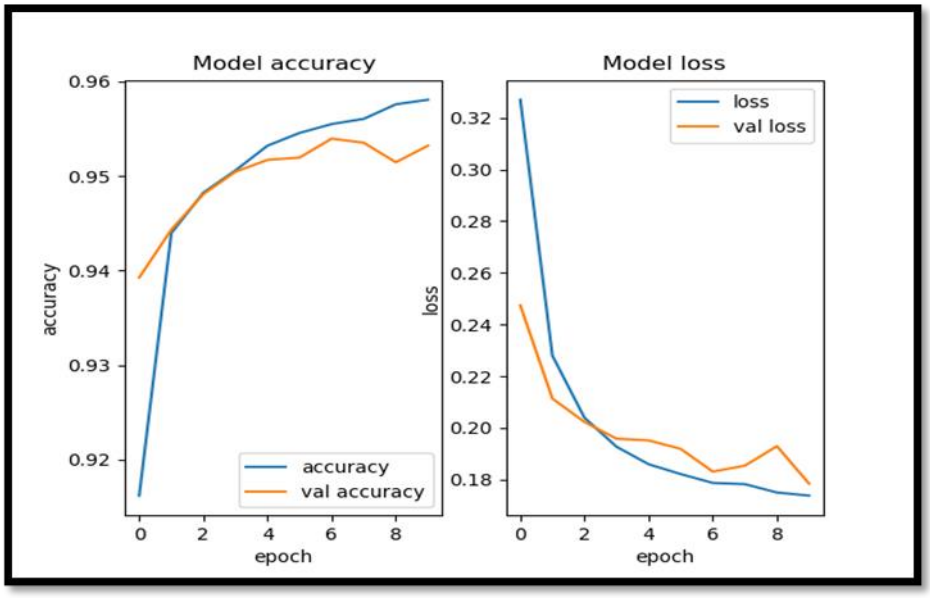

Figure A4. Accuracy results of CNN1D-LSTM.

Figure below shows, the process of training and accuracy results which illustrates on 8th Epoch it improved the accuracy results to 0.96 (96%). The Loss kept reducing to 0.18. The last results were different in all model results. The result of experimentation is listened in the text above (See Experimentation Results).

ROC Curve of NN:

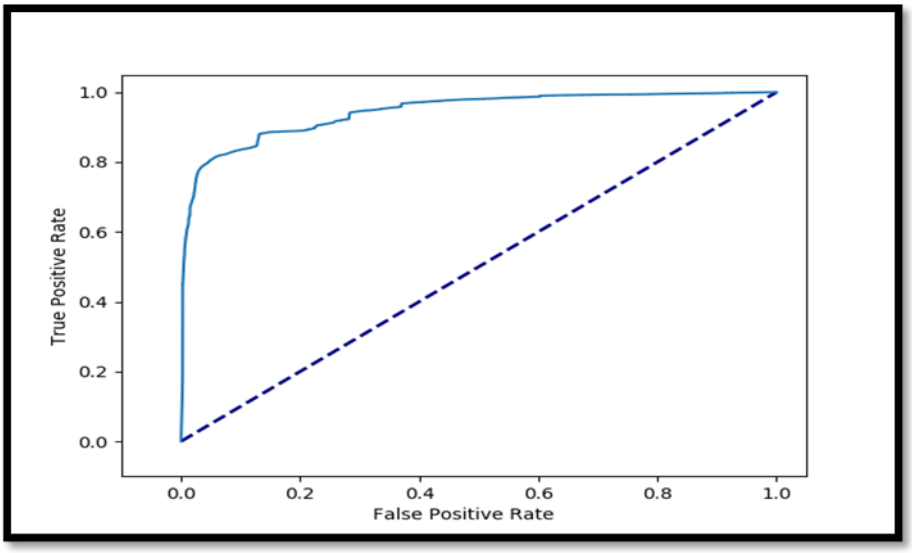

Figure A5. ROC Curve describing True Positives and False Positives.

This figure above illustrates the true positive and false positive rates – as it gradually rises as per the accuracy. False positive rate from 0.0 to 0.2 and true positive rate from 0.8 to 1.0 – it can be seen there is tiny flicker where it inclines towards false positives but then it becomes straight.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 July 2024 doi:10.20944/preprints202407.1214.v1

Accuracy Results of NN:

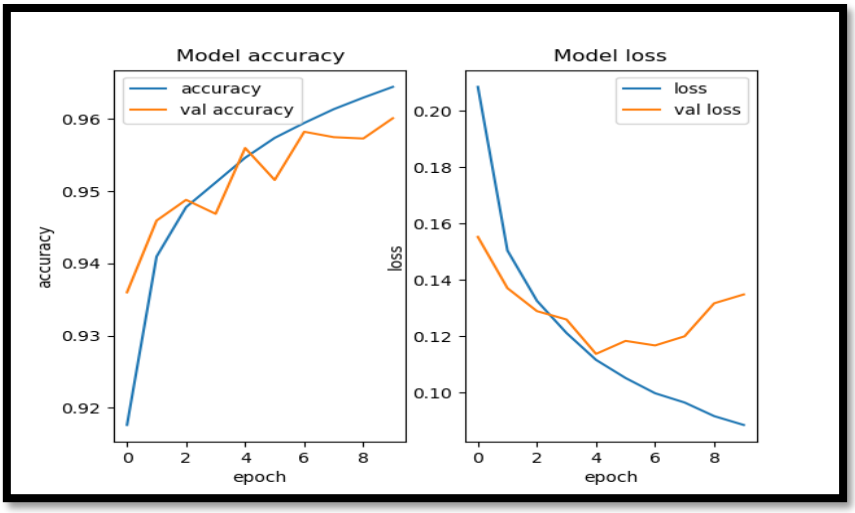

Figure A6. Accuracy results of NN.

The training procedure and accuracy results are displayed in the figure below, which shows how it increased the accuracy results to 0.96 (96%). The Loss continued dropping to 0.10.

Appendix B

Below Figure A7 illustrates the process of training and validation from preprocessing the dataset and loading into the model.

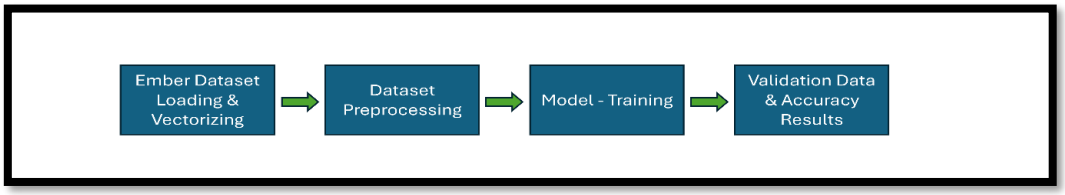

Figure A7. Training & Validation Process.

Figure A8 below shows the process of cross-validation and prediction results.

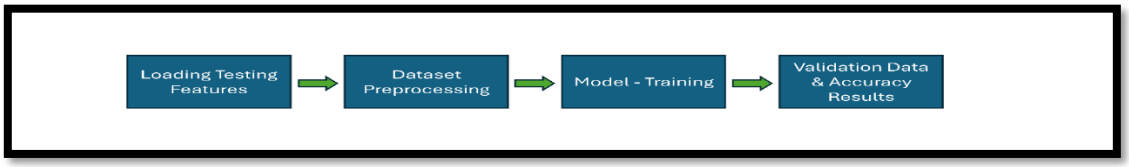

Figure A8. Cross-Validation & Prediction Process.

This figure below (Figure A9) shows the process of Detection of a PE File (i.e. exe file).

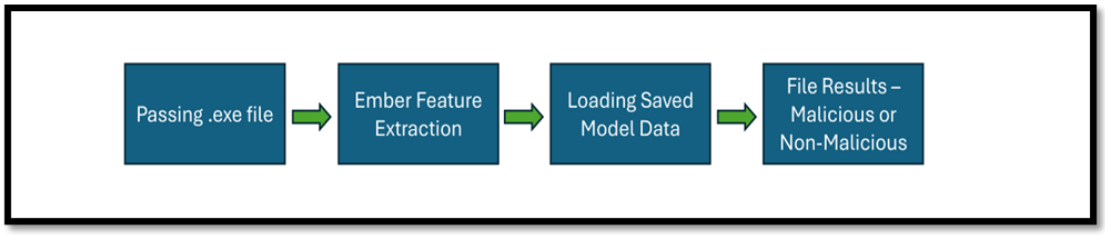

Figure A9. Process of Detection of a PE File.

 doi:10.20944/preprints202407.1214.v1

References

1. Raff, Edward, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K. Nicholas. "Malware detection by eating a whole exe." In Workshops at the thirty-second AAAI conference on artificial intelligence. 2018, doi:1710.094335v1
2. Sikorski, Michael, and Andrew Honig. Practical malware analysis: the hands-on guide to dissecting malicious software. no starch press, 2012.
3. Perdisci, Roberto, Andrea Lanzi, and Wenke Lee. "Classification of packed executables for accurate computer virus detection." *Pattern recognition letters* 29, no. 14: 1941-1946, 2008.
4. S. William, Computer security: Principles and practice. Pearson Education India, 2008.
5. Kim, Samuel. "PE header analysis for malware detection.", 2018.
6. Ken Proska, Corey Hildebrandt, Daniel Kappellmann Zafra, Nathan Brubaker, "Portable Executable File Infecting Malware Is Increasingly Found in OT Networks", Oct 27, 2021. Available Online: https://www.mandiant.com/resources/pe-file-infecting-malware-ot
7. Damodaran, Anusha, Fabio Di Troia, Corrado Aaron Visaggio, Thomas H. Austin, and Mark Stamp. "A comparison of static, dynamic, and hybrid analysis for malware detection." *Journal of Computer Virology and Hacking Techniques* 13: 1-12, 2017.
8. Moser, Andreas, Christopher Kruegel, and Engin Kirda. "Limits of static analysis for malware detection." In *Twenty-third annual computer security applications conference (ACSAC 2007)*, pp. 421-430. IEEE, 2007.
9. Gandotra, Ekta, Divya Bansal, and Sanjeev Sofat. "Malware analysis and classification: A survey." *Journal of Information Security* 2014, 2014.
10. U. Bayer, A. Moser, C. Kruegel, and E. Kirda, "Dynamic analysis of malicious code," Journal in Computer Virology, 2006, vol. 2, no. 1, pp. 67–77.
11. I. You and K. Yim, "Malware obfuscation techniques: A brief survey," in Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on, 2010, pp. 297–300, IEEE.
12. [1804.04637] EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models (arxiv.org) April 16, 2018 https://arxiv.org/abs/1804.04637
13. Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis – https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141, Sept 2021.
14. Li-Ion Batteries Parameter Estimation with Tiny Neural Networks Embedded on Intelligent IoT Microcontrollers, Giulia Crocioni, Danilo Pau, Jean-Michel Delorme, Giambattista Gruosso, July 2020, https://www.researchgate.net/publication/342686148_Li-Ion_Batteries_Parameter_Estimation_With_Tiny_Neural_Networks_Embedded_on_Intelligent_IoT_Microcontrollers.
15. H. Kang, J.-w. Jang, A. Mohaisen, and H. K. Kim, "Detecting and classifying android malware using static analysis along with creator information" International Journal of Distributed Sensor Networks, 2015, vol. 11, no. 6, p. 479174.
16. Z. Aung and W. Zaw, "Permission-based android malware detection," International Journal of Scientific & Technology Research, 2013, vol. 2, no. 3, pp. 228–234.
17. M. Z. Shafiq, S. M. Tabish, F. Mirza, and M. Farooq, "Pe-miner: Mining structural information to detect malicious executables in Realtime," in International Workshop on Recent Advances in Intrusion Detection, 2009, pp. 121–141.
18. GitHub – tamnguyenvan/malnet: Malware Detection using Convolutional Neural Networks, 2020.
19. T.-Y. Wang, C.-H. Wu, and C.-C. Hsieh, "Detecting unknown malicious executables using portable executable headers," in INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on, 2009, pp. 278–284, IEEE.
20. Saxe, A. M., Berlin, K., & Cunningham, A., "Deep neural network-based malware detection using two-dimensional binary program features.", 2015.
21. Kuang, Xiaohui, Ming Zhang, Hu Li, Gang Zhao, Huayang Cao, Zhendong Wu, and Xianmin Wang. "DeepWAF: detecting web attacks based on CNN and LSTM models." In Cyberspace Safety and Security: 11th International Symposium, December 1–3, 2019, Part II 11, pp. 121-136. Springer International Publishing. https://doi.org/10.1007/978-3-030-37352-8_11
22. Kumar, R.; Subbiah, G. Zero-Day Malware Detection and Effective Malware Analysis Using Shapley Ensemble Boosting and Bagging Approach. Sensors, 2022, 22, 2798. https://doi.org/10.3390/s22072798

23. Bilge, L., & Dumitras, T., "Before We Knew It: An Empirical Study of Zero-Day Attacks In The Real World.", 2012.
24. Carlini, N., & Wagner, D., "Towards Evaluating the Robustness of Neural Networks.", 2017.
25. Yuan, Z., Lu, Y., & Xue, Y., DroidDetector: Android malware characterization and detection using deep learning. Tsinghua Science and Technology, 2016, 21(1), 114-123.
26. A. Damodaran, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection," Journal of Computer Virology and Hacking Techniques, 2015, vol. 13, pp. 1–12.
27. Abdullah, Muhammed Amin, Yongbin Yu, Kwabena Adu, Yakubu Imrana, Xiangxiang Wang, and Jingye Cai. "HCL-Classifier: CNN and LSTM based hybrid malware classifier for Internet of Things (IoT)." Future Generation Computer Systems 142, 2023, 41-58. https://doi.org/10.1016/j.future.2022.12.034
28. Omotosho, O.I & Baale, Adebisi & Oladejo, O.A. & Ojiyovwi, Adelodun. "An Exploratory Study of Recurrent Neural Networks for Cybersecurity". Advances in Multidisciplinary and scientific Research Journal Publication. 2021, 197-204. Doi:10.22624/AIMS/ABMIC2021P15.
29. Saxe, J., Berlin, K., & Pentney, W., Deep neural network-based malware detection using two-dimensional binary program features. Journal of Computer Virology and Hacking Techniques, 15(1), 2019, 13-24.
30. Rieck, K., Trinius, P., Willems, C., & Holz, T., Automatic analysis of malware behavior using machine learning. Journal of Computer Security, 19(4), 2011, 639-668.
31. Handaya, Wilfridus Bambang Triadi, Mohd Najwadi Yusoff, and A. Jantan. "Machine learning approach for detection of fileless cryptocurrency mining malware." In Journal of Physics: Conference Series, 2020, vol. 1450, no. 1, p. 012075.
32. Oyama, Yoshihiro et al. "Identifying Useful Features for Malware Detection in the Ember Dataset." 2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW), 2019, 360-366.
33. Almazroi, A.A., Ayub, N. Deep learning hybridization for improved malware detection in smart Internet of Things. 2024, Sci Rep 14, 7838. https://doi.org/10.1038/s41598-024-57864-8
34. T.-Y. Wang, C.-H. Wu, and C.-C. Hsieh, "Detecting unknown malicious executables using portable executable headers," in INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on, 2009, pp. 278–284, IEEE.
35. Gueriani, Afrah, Hamza Kheddar, and Ahmed Cherif Mazari. "Enhancing IoT Security with CNN and LSTM-Based Intrusion Detection Systems." 2024, arXiv:2405.18624.
36. Vitorino, João, Rui Andrade, Isabel Praça, Orlando Sousa, and Eva Maia. "A comparative analysis of machine learning techniques for IoT intrusion detection." In International Symposium on Foundations and Practice of Security, 2021, pp. 191-207. Cham: Springer International Publishing, 2021. arXiv:2111.13149v3
37. Akhtar, M.S.; Feng, T. Detection of Malware by Deep Learning as CNN-LSTM Machine Learning Techniques in Real Time. Symmetry. 2022, 14, 2308. https://doi.org/10.3390/sym14112308
38. Mahadevappa, Poornima, Syeda Mariam Muzammal, and Raja Kumar Murugesan. "A comparative analysis of machine learning algorithms for intrusion detection in edge-enabled IoT networks.", 2021. https://doi.org/10.48550/arXiv.2111.01383
39. Riaz, S.; Latif, S.; Usman, S.M.; Ullah, S.S.; Algarni, A.D.; Yasin, A.; Anwar, A.; Elmannai, H.; Hussain, S. Malware Detection in Internet of Things (IoT) Devices Using Deep Learning. Sensors. 2022, 22, 9305. https://doi.org/10.3390/s22239305