telecom

MDPI

Article

Deep Learning Optimisation of Static Malware Detection with Grid Search and Covering Arrays

Fahad T. ALGorain [1,*,†] and Abdulrahman S. Alnaeem [2,*,†]

[1] Department of Computer Science, University of Sheffield, Sheffield S10 2TN, UK
[2] Department of Computer Science, University of Manchester, Manchester M13 9PL, UK
* Correspondence: ftalgorain1@sheffield.ac.uk (F.T.A.); abdulrahman.alnaeem@postgrad.manchester.ac.uk (A.S.A.)
† These authors contributed equally to this work.

Abstract: This paper investigates the impact of several hyperparameters on static malware detection using deep learning, including the number of epochs, batch size, number of layers and neurons, optimisation method, dropout rate, type of activation function, and learning rate. We employed the cAgen tool and grid search optimisation from the scikit-learn Python library to identify the best hyperparameters for our Keras deep learning model. Our experiments reveal that cAgen is more efficient than grid search in finding the optimal parameters, and we find that the selection of hyperparameter values has a significant impact on the model's accuracy. Specifically, our approach leads to significant improvements in the neural network model's accuracy for static malware detection on the Ember dataset (from 81.2% to 95.7%) and the Kaggle dataset (from 94% to 98.6%). These results demonstrate the effectiveness of our proposed approach, and have important implications for the field of static malware detection.

Keywords: hyperparameter optimisation; static malware detection; neural network; deep learning; grid search; cAgen; combinatorial testing; covering arrays

Citation: ALGorain, F.T.; Alnaeem, A.S. Deep Learning Optimisation of Static Malware Detection with Grid Search and Covering Arrays. *Telecom* **2023**, *4*, 249–264. https://doi.org/10.3390/telecom4020015

Academic Editor: Philip Branch

Received: 8 March 2023
Revised: 23 April 2023
Accepted: 26 April 2023
Published: 4 May 2023

1. Introduction

Malware poses a persistent threat to software systems, and its detection is a critical issue in modern cybersecurity. Machine learning (ML) has shown promise in detecting malware, particularly static malware detection, which examines harmful binary files without executing them. However, the performance of ML algorithms is highly dependent on the choice of hyperparameters, which can be challenging to determine. This study investigates the use of hyperparameter-optimisation (HPO) techniques to improve the performance of a deep learning-based model for static malware detection. Specifically, we aim to enhance the performance of static malware detection using deep learning models by fine-tuning hyperparameters. The selection of hyperparameters, such as the number of epochs, batch size, number of layers, number of neurons, optimisation method, dropout relay, type of activation function, and learning rate (LR), was motivated by their significant impact on the efficacy of deep learning models in previous studies [1,2]. Our goal is to optimise these hyperparameters to achieve better accuracy and generalisation performance of deep learning models for static malware detection.

1.1. Malware and Its Detection

Malware is any code added, changed, or removed from a software system in order to intentionally cause harm or subvert the intended function of the system [1]. For some time now, malware detection research and development has been a consistent focus of academic and commercial efforts. Malware continues to be one of the most critical issues in modern cybersecurity [2]. Malware detection can be performed in three ways: statically,

dynamically, and hybrid. This study focuses on static malware detection, which examines harmful binary files without actually running them. Dynamic malware detection identifies malware by utilising characteristics of run-time execution behaviour. The two methods are combined in hybrid detection. Researchers have investigated the prospect of utilising machine learning (ML) to detect malware, and numerous corporations and universities have made considerable investments in the development of novel techniques for recognising it [3]. The choice of static analysis for this research was motivated by the fact that static malware detection can be performed without running the binary files, making it a safer approach.

1.2. Brief about HPO

When dealing with ML-based approaches, it can be challenging to determine their true detection capabilities. For instance, the performance of ML algorithms often relies on the parameters used, yet the literature often lacks a persuasive justification for the specific choices made. In this study, we investigate a technique called hyperparameter optimisation (HPO) for improving the performance of such algorithms. To address the curse of dimensionality that arises from using the most popular systematic approach, grid search, we explore the use of covering arrays (CAs).

1.3. Generating Covering Arrays

While this study focuses on using hyperparameter optimisation to improve the performance of machine learning algorithms, it is important to provide a brief overview of covering arrays, which will be used in our optimisation approach. Covering arrays have been widely used in software testing to reduce the required number of tests compared to comprehensive combinatorial testing [4]. This approach is particularly useful when dealing with a system with multiple input parameters. The In-Parameter-Order-General (IPOG) technique has gained traction as a way to generate covering arrays of variable strengths t. Covering arrays can be generated with different methods, such as the automatic efficient test generator (AETG) system [5], the deterministic density algorithm (DDA) [6,7], and the advanced combinatorial testing system (ACTS) [5–9]. The IPO method expands the covering array column by column, adding rows as necessary to achieve full t-way coverage. Finding the ideal covering array is an NP-complete problem [10]. To decrease the size of covering arrays, IPOG employs a greedy construction method. In our study, we used a tool called cAgen [11] to generate covering arrays efficiently. The cAgen tool combines IPOG with a genetic algorithm to generate high-quality covering arrays in a reasonable amount of time. By using cAgen to explore the hyperparameter space, we aimed to improve the performance of our deep learning model for static malware detection.

1.4. Brief about Grid Search and Covering Arrays

Grid search is a widely used tool for exploring parameter spaces to find optimal parameter values for a given model. However, conducting a full grid search can be computationally expensive, making it necessary to have an efficient method of combinatorial space exploration. Covering arrays are a well-established method for this purpose, and they can be implemented at different "strengths" to customize the depth of the search space. In this method, each test is represented as a row in the array, with different variables represented in the columns. The Cartesian product of all parameter sets achieves full combinatorial coverage. For any subset of t parameters, each conceivable t-tuple of values appears in exactly one row in a covering array of strength t, also known as a CA_t^D. An orthogonal array (OA) is a covering array where each tuple occurs exactly once, making it the most effective covering array. In practice, pairwise tests, where $t = 2$, are commonly used, but low values of t can also yield excellent performance in fault-finding. The use of covering arrays can significantly reduce the size of test sets compared to a full combinatorial grid search, making it an efficient method for hyperparameter optimisation.

1.5. Brief about Deep Learning

Deep learning (DL) is a widely used subset of machine learning (ML) that has had particular success in fields such as computer vision, natural language processing, and machine translation. DL is part of the artificial neural network (ANN) theory, and there are several types of DL models, including deep neural networks (DNNs), feed forward neural networks (FFNNs), and deep belief networks (DBNs), among others [12]. Most DL models have similar traits and hyperparameters, and the performance of the trained model significantly depends on the values of the parameters of the model [1–3,13–15].

1.6. Hyperparameter Optimisation for Deep Learning

Hyperparameter optimisation is crucial for improving the performance of machine learning (ML) models [16–22]. The most common approach for hyperparameter optimisation is grid search, which requires the user to define a search space beforehand and perform an exhaustive search over the discretised space to find optimal values. However, grid search has limitations and may suffer from poor performance if the search space does not include the best values [23].

To address this limitation, we explored the feasibility of using a tool called cAgen [11] to implement covering arrays (CAs) in our study. CAs provide an efficient method of combinatorial space exploration by customising the depth to which the search space is probed. We compared the performance of grid search and cAgen in finding optimal parameter choices for our deep learning models, aiming to improve the robustness and accuracy of our models for static malware detection.

Our study shows that the time and efficiency of cAgen [11] in finding the best hyperparameters are superior to those of grid search. Furthermore, we demonstrate that the choice of different hyperparameter values can significantly impact the model's performance. Finally, we show that our hyperparameter-optimisation choices significantly improve the performance of the neural network model for static malware detection on two datasets: Ember [24] and Kaggle [25]. The findings of our study have important implications for the field of static malware detection, providing a framework for improving the performance of deep learning models for malware detection through hyperparameter optimisation using covering arrays.

1.7. Paper Contribution

1. The paper presents a novel approach to improving static malware detection using deep learning models with hyperparameter optimisation through covering arrays. The study demonstrates the feasibility of using cAgen in combination with grid search to find the optimal hyperparameter values. The findings show that this approach can significantly improve the accuracy of the baseline model for both the Ember and Kaggle datasets.
2. The study provides insights into the effects of different hyperparameters and their interactions on the performance of deep learning models for static malware detection.
3. It provides a framework for researchers and practitioners to improve the performance of deep learning models for malware detection through hyperparameter optimisation using covering arrays.
4. The results of this study have implications for the wider field of cybersecurity, where accurate malware detection is critical in protecting computer systems and networks.

The targeted performance metric used in this work is accuracy. This follows many malware detection research papers and is very common practice for many deep learning applications. It is one of the major accepted performance criteria. To the best of our knowledge, no one has attempted to study the effects of different deep learning hyperparameters for malware detection. This is especially with both datasets Ember and Kaggle while utilising covering arrays and grid search approaches.

1.8. The Structure of the Paper

See Section 2 for related literature, Section 3 for our methodology, and Section 4 for experiment setup and a brief of the datasets used. Section 5 presents the grid search versus cAgen for NN hyperparameter-optimisation tasks. In Section 6 we have the results and discussion. Finally, in Section 7 we conclude.

2. Related Literature

2.1. ML-Based Static Malware-Detection-Related Literature

Malware that runs on Windows and uses the portable execution (PE) format is particularly prevalent. To detect PE malware, several studies have investigated the feasibility of using machine learning, such as [26–28].

Recently, the authors of [24] released a dataset known as the Ember dataset, along with Python functions, to facilitate data accessibility. They also provided examples of how various machine learning methods can be applied to their dataset as a baseline. The authors of [29] developed a static detection technique based on a gradient-boosting decision tree algorithm, which outperformed the baseline model with less time spent on training.

Another study [30] examined various machine learning models using a subset of the Ember dataset, focusing on efficiency and scalability in identifying malware families. Their work demonstrated an improvement in performance between the suggested random forest model and the baseline model, with the random forest method achieving the highest accuracy of 99.9%.

In [31], researchers presented an ensemble learning-based strategy for identifying malicious software using a stacked ensemble of fully connected, one-dimensional convolutional neural networks (CNNs) and compared 15 existing machine learning classifiers to develop a meta-learner. The highest accuracy was achieved by an ensemble of seven neural networks, where the ExtraTrees classifier was used in the final stage of classification.

Similarly, in [32], the authors attempted a novel method for detecting malware using four machine learning techniques tuned with cAgen, a tool for generating covering arrays. Their results showed that cAgen is an efficient approach to achieving optimal parameter choices for machine learning techniques with less time and iteration than grid search. In [33], the authors have tried a new approach in which they utilised mixed-level covering arrays to design and tune a Convolutional neural network (CNN) for audio classification tasks.

To compare and contrast the proposed work with state-of-the-art literature, a table is presented below.

From Table 1, it can be seen that although various studies have been conducted on machine-learning-based static malware detection, only a few of them have utilised covering arrays for hyperparameter optimisation. Moreover, most of the studies have been limited to the evaluation of small datasets, and some have not compared their methods with covering arrays for hyperparameter optimisation. It is noteworthy that hyperparameter optimisation was not the focus of the previous literature.

In contrast, the proposed work introduces a novel approach to utilising covering arrays for hyperparameter optimisation, which improves the performance of deep learning models for static malware detection. Additionally, the proposed work comprehensively evaluates the methodology on two datasets, Ember and Kaggle, and compares covering arrays and grid search for hyperparameter optimisation.

Overall, the proposed work stands out from the previous studies by providing a more robust and comprehensive analysis of the impact of covering arrays on hyperparameter optimisation. By utilising covering arrays, the proposed work achieves better performance with less computational resources, making it a promising avenue for future research in the field of machine-learning-based static malware detection.

Table 1. Comparison of ML-based static malware-detection techniques.

Reference	Methodology	Limitations	Comparison with Proposed Work
[26]	Machine learning applied to the PE format	Limited dataset size, performance varies with dataset size and complexity	Early work in the field, smaller dataset used, does not utilise covering arrays for hyperparameter optimisation.
[27]	Machine-learning-based classification using n-gram frequency counts	Reliance on feature engineering, performance not evaluated on large-scale dataset	Older approach, less efficient feature engineering.
[29]	Gradient-boosting decision tree algorithm for static detection	Limited evaluation on only one dataset	Methodology not compared with covering arrays for hyperparameter optimisation.
[30]	Analysis of machine learning models using a subset of the Ember dataset	Only used a subset of the Ember dataset, no comparative analysis of covering arrays	Evaluates only a subset of the Ember dataset, no comparison with covering arrays for hyperparameter optimisation.
[31]	Stacked ensemble of fully connected, one-dimensional CNNs	Large amount of computational resources required, the selection of hyperparameters still crucial	No comparison with covering arrays for hyperparameter optimisation.
[32]	Novel method for detecting malware using four machine learning techniques tuned with cAgen	Limited evaluation of only one dataset, less efficient approach for covering array generation (compatible with classical ML techniques only)	Similar approach to the proposed work, but less efficient method for covering array generation (classical ML techniques only).
[33]	Utilisation of mixed-level covering arrays to design and tune a CNN for audio classification	Limited to audio classification task, no comparison with grid search	Different problem domain, no comparison with grid search.

2.2. Covering-Array-Related Literature

The generation of optimal values is the most challenging component of many problems for covering arrays [5]. Several approaches have been presented to generate covering arrays, each with its own set of pros and cons, but they all share similarities with various systems, including the automatic efficient test generator (AETG) system [5], the deterministic density algorithm (DDA) [6,7], the in-parameter-order system [8], and the advanced combinatorial testing system (ACTS) [5–9].

The IPO method expands the covering array column by column, adding rows as necessary to achieve full t-way coverage. Different studies have been conducted to better understand how the in-parameter-order technique can be used to generate covering arrays. The in-parameter-order-general (IPOG) technique was developed to make it possible to use the strategy to generate covering arrays of variable strength. A change to IPOG in [34] allowed for shorter generating times and smaller covering arrays in some situations. The number of possible permutations was reduced thanks to the recursive construction method presented in [35]. The size of covering arrays can be decreased by employing graph-colouring approaches, as suggested in [36].

It should be noted that while prior literature has explored the use of various methods to optimise hyperparameters for deep learning models in the context of static malware detection, the present study is the first to utilise cAgen, a covering-array tool, for this purpose. Therefore, our work provides novel insights into the potential benefits of using cAgen in improving the performance of deep learning models for static malware detection. cAgen was chosen for its ability to efficiently explore the hyperparameter search space and identify the best hyperparameter configurations.

3. Methodology

Overall Approach

This section provides an overview of the methodology used in this study. The overall approach involved utilising deep learning models for static malware detection and optimising their hyperparameters through covering arrays. The study aimed to compare the performance of CAs and grid search in terms of efficiency and accuracy. The first step was to preprocess the dataset and split it into training and testing sets with an 80:20 ratio. The training set was used to train the deep learning models, and k-fold cross-validation with a 3-fold configuration was used to ensure the robustness of the results. Next, the hyperparameters of the deep learning models were optimised through both CAs and grid search. CAs were found to be more efficient in terms of computational resources and time complexity, and they outperformed grid search in terms of model accuracy.

In summary, this study demonstrates the potential of utilising deep learning models and CAs to improve static malware detection. The use of CAs for hyperparameter optimisation provides significant advantages over the conventional grid-search approach.

4. Experiment Details and Dataset Brief

4.1. Datasets

We utilised two datasets for our experiments. The first dataset, EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models [37], was used to train our baseline model. Version 2018 of the dataset was used. The authors stressed that the 2018 version of the dataset would present a significant challenge to ML-based algorithms. This dataset contains a total of 1 million samples, of which 200 k are unknown (not used in our experiments), 300 k are benign, 300 k are malware samples for training, and 200 k are reserved for testing. The second dataset, obtained from Kaggle.com, was produced by [25] and uses PE files from [38]. This dataset contains 19,611 malicious and benign samples, and 75 features represent each sample. The dataset was partitioned into 80% training and 20% testing sets. All experiments were performed using a Jupyter Notebook version 6.1.0 and analysed using Python version 3.6.0.

4.2. Experiment Setup

DL models were based on scikit-learn [39]. The experiments were performed using iMac (Retina 5K, 27-inch, 2017, 4.2 GHz Quad-Core Intel Core i7, 16 GB 2400 MHz DDR4 RAM, and a Radeon Pro 580 8 GB graphics card) and Windows OS 11, with 11 Gen Intel Core i7-11800H, with 2.30 GHz processor, and 16 GB RAM. Inputs were subject to normalisation via the scikit-learn library's StandardScaler function [40]. A 3-fold validation was used throughout. Our work concerns supervised learning only.

4.3. Baseline Model

The flowchart in Figure 1 shows the sequential steps taken in the study, starting with the baseline model for Ember, which had a 3-layer neural network with 12, 8, and 1 neurons and achieved an accuracy of 81.2%. The next step was to train another baseline model for Kaggle with the same architecture, which achieved an accuracy of 94%. These results in Table 2 provided the foundation for exploring hyperparameter optimisation, which was the next step in the study.

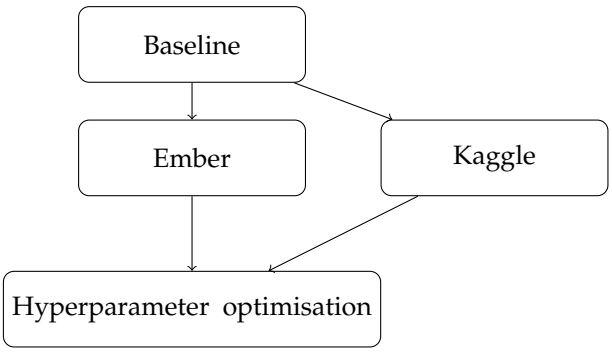

Figure 1. Workflow for neural network experiments.

Table 2. Baseline model results.

Dataset	Architecture	Accuracy
Ember	3-layer neural network	81.2%
Kaggle	3-layer neural network	94%

4.4. Performance Evaluation

To assess the effectiveness of our approach, we evaluated the accuracy and efficiency of the deep learning models using two popular hyperparameter-optimisation techniques, covering arrays and grid search. We compared the performance of the baseline models against the optimised models, considering both accuracy and efficiency metrics. Specifically, we focused on optimising the selected hyperparameters. To ensure that the models are not overfitted during the tuning process, we employed several techniques, such as early stopping, which stops the training process when the model performance on the validation set starts to deteriorate, and regularisation techniques, such as dropout relay. Additionally, the model's performance was evaluated on a separate test set to ensure that it generalises well to unseen data.

5. Grid Search versus cAgen for NN Hyperparameter-Optimisation Tasks

We explored two methods for hyperparameter optimisation, grid search and cAgen. There are structural and behavioural representations for each method described below.

5.1. Grid Search

The four primary phases of grid search include searching through the defined values in the search space, utilising the training set, performing cross-validation (in this study, we utilised three folds), and using the validation set to determine the optimal values that will improve model performance. Structurally, grid search involves defining a set of hyperparameters and their respective values to search through. This creates a search space for possible combinations of hyperparameters, which is explored systematically. The steps involved in the grid search process are:

1. Define the search space for hyperparameters and their respective values;
2. Split the data into a training set and a validation set;
   For each combination of hyperparameters in the search space:
3. Train a model on the training set using the current hyperparameters;
4. Evaluate the model's performance on the validation set;
5. Select the hyperparameters that result in the best performance on the validation set.

Behaviourally, grid search can be described as a brute-force method of hyperparameter optimisation, as it exhaustively searches through all possible combinations of hyperparameters. While this method can be effective, it can also be computationally expensive and time consuming, especially for large search spaces. Figure 2 below depicts the flow of the grid search experiment.

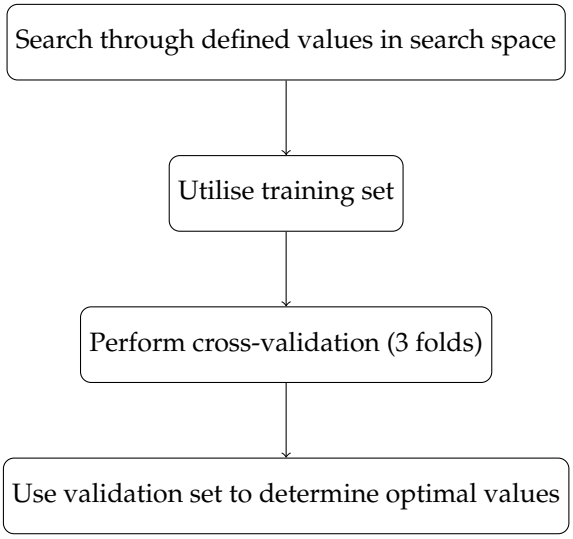

Figure 2. Flowchart for grid search hyperparameter-optimisation process.

5.2. cAgen

To use cAgen for hyperparameter optimisation, certain parameters need to be defined beforehand. Firstly, a workspace needs to be created to prepare the model. Next, the input parameter (IPM) is selected to modify the model's parameters. The final step is to set the value of t, which determines the strength of the covering array. Due to the large size of the Ember dataset, we only investigated covering arrays of strengths 1 and 2. However, for the Kaggle dataset, we explored covering arrays of strengths 1, 2, and 3.

Structurally, cAgen involves creating a covering array, a matrix representing a set of parameter configurations. Each row in the matrix represents a unique configuration of hyperparameters, and the columns represent the individual hyperparameters. The steps involved in the cAgen process are:

1. Create a workspace to prepare the model;
2. Select the input parameter (IPM) to modify the model's parameters;
3. Set the value of t, which determines the strength of the covering array;
4. Generate a covering array of hyperparameter configurations using the IPM and t-value;
   For each configuration in the covering array:
5. Train a model on the training set using the current hyperparameters;
6. Evaluate the model's performance on the validation set;
7. Select the hyperparameters that result in the best performance on the validation set.

Behaviourally, cAgen can be described as an optimisation method that uses a covering array to explore a subset of the search space rather than exhaustively searching through all possible combinations of hyperparameters. This approach can be more efficient than grid search for large search spaces, as it reduces the number of configurations needing exploration. However, the quality of the covering array depends on the strength value and the choice of input parameters, which can affect the effectiveness of the optimisation process. The implementation details for using cAgen with neural networks are shown below in Figures 3 and 4.

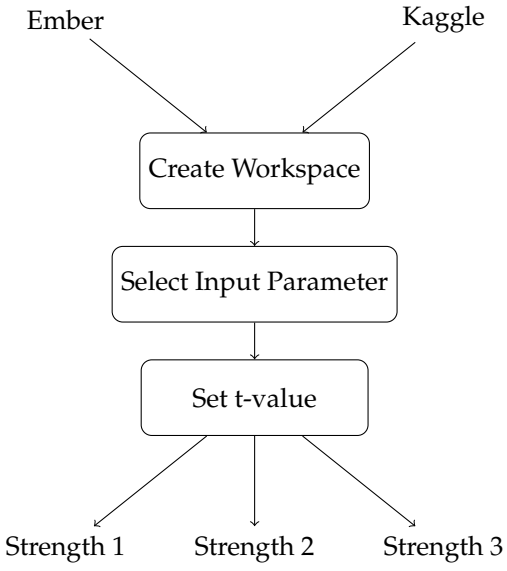

Figure 3. Flowchart for cAgen implementation details.

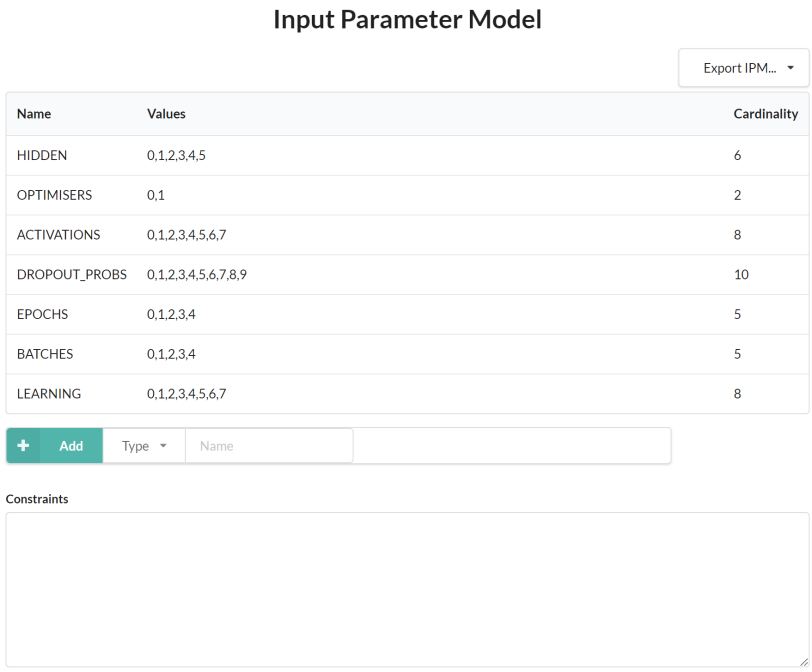

Figure 4. cAgen implementation detail for both the Ember and the Kaggle datasets.

5.3. Hyperparameter Grid Search Configurations

As the hyperparameters of a deep learning model significantly impact its performance, our study aims to explore the effects of various hyperparameter choices on the accuracy of the model in static malware detection. Therefore, we selected a range of hyperparameters, including the number of epochs, batch size, number of neurons, optimisation method, dropout relay, type of activation function, and learning rate, to evaluate their impact on the performance of the model. By examining the effects of different parameter choices, we aimed to identify the optimal configuration of hyperparameters that leads to improved model accuracy. This will provide valuable insights into the impact of hyperparameter choices in the domain of static malware detection. The following Tables 3 and 4 show the hyperparameter configurations for each DL model parameter.

Table 3. DL model hyperparameter configurations (Ember dataset).

Hyperparameter	Grid Search Space
Number of neurons (per layer)	[1200, 1400, 1800, 2000, 2200, 2400]
Number of epochs	[20, 40,60, 80, 100]
Batch size	[16, 32, 48, 64, 80]
Optimiser	Adam or SGD
Drop-out relay	[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
Learning rate	[0.001, 0.01, 0.1, 0.11, 0.12, 0.113, 0.114, 0.2]
Activation function (per layer)	softmax, softplus, softsign, relu, tanh, sigmoid, hard_sigmoid, linear

Table 4. DL model grid search configuration space (Kaggle dataset).

Hyperparameter	Grid Search Space
Number of neurons (per layer)	[55, 60, 65, 70, 75, 80]
Batch size	[48, 64, 80, 100, 128]
Number of epochs	[20, 40, 60, 80, 100]
Optimiser	Adam or SGD
Drop-out relay	[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
Learning rate	[0.001, 0.01, 0.1, 0.11, 0.12, 0.113, 0.114, 0.2]
Activation functions (per layer)	softmax, softplus, softsign, relu, tanh, sigmoid, hard_sigmoid, linear

6. Results and Discussion

6.1. cAgen Tool Hyperparameter Versus Grid Search Results

Below are the results from cAgen tool for both the Kaggle Table 5 and Ember datasets Table 6.

Tables 5 and 6 illustrate the results of our study, and below are the key findings:

1. Both the grid search and cAgen tool were effective in finding optimal hyperparameters for the deep learning models.
2. The cAgen tool was particularly useful for finding optimal hyperparameters for the Ember dataset, which had a larger number of features and samples.
3. The optimised deep learning models achieved significantly higher accuracy than the baseline models for both datasets.
4. The optimised deep learning models achieved higher accuracy than the previous benchmark model for the Ember dataset (it was at 92%).
5. The choice of hyperparameters significantly impacted the accuracy of the models, particularly for the learning rate and number of neurons.
6. The results highlight the potential of using hyperparameter-optimisation techniques for improving the performance of deep learning models in static malware detection

Table 5. DL Model cAgen results comparison (Kaggle dataset).

ML Algorithm	Optimal Values Found	*t*-Strength Values/Grid Search	Time to Complete	Number of Combinations Searched	Score (Accuracy)
DL	75, 150, 1 SGD tanh, sigmoid 0.0 20 128 0.0112	T2	7 min 36 s	60	0.9814
	75, 150, 1 Adam tanh, sigmoid 0.0 20 80 0.0001	T3	49 m 1 s	361	0.9840
	75, 80, 1 SGD tanh, relu 0.1 20 80 0.0112	T4	3 h 17 min 14 s	1452	0.9842
	75, 80, 80 Adam softsign, tanh 0.2 60 80 0.0112	Full grid search	~4 days 3 h 6 min	all	**0.9862**

Table 6. DL Model cAgen results comparison (Ember dataset).

ML Algorithm	Optimal Values Found	*t*-Strength Values/Grid Search	Time to Complete	Number of Combinations Searched	Score (Accuracy)
DL	2400, 1200, 1200 Adam softplus, relu, sigmoid 0.0 20 64 0.00001	T1	15 h 31 min	10	0.9562
	1200, 1200, 1200 Adam sofplus, relu, sigmoid 0.5 40 128 0.0001	T2	4 d 23 h 5 min 47 s	60	**0.9575**
	2400, 1200, 1200 Adam hard_sigmoid, relu, sigmoid 0.1 40 64 0.0	Full grid search	∼29 days 15 h	all	0.9542

It is important to carefully choose and tune all relevant hyperparameters when training deep learning models for malware detection.

Table 7 shows the results of the optimised models for the Ember and Kaggle datasets. The table lists the dataset name, the neural network architecture, and the accuracy achieved by the model. The optimised model for Ember is a 6-layer neural network, which achieved an accuracy of 95.7%. On the other hand, the optimised model for Kaggle is a 5-layer neural network, which achieved an accuracy of 98.6%. It is important to note that in the new optimised model, two dropout layers were added for regularisation to avoid overfitting. Dropout layers randomly ignore some of the neurons in the neural network during training, which helps to prevent the model from relying too heavily on any single feature. This ensures the model generalises well and does not overfit the training data. Below, Figure 5 displays the final structure for both models side by side.

Table 7. Optimised model results.

Dataset	Architecture	Accuracy
Ember	6-layer neural network	95.7%
Kaggle	5-layer neural network	98.6%

6.2. Discussion

The discussion presents the findings of the study, which aims to optimise deep learning models using covering arrays (CAs) for static malware detection. We analysed the effects of different hyperparameters on the performance of the deep learning models.

6.3. Tuning the Number of Neurons

The number of neurons is a critical factor in determining the effectiveness of a neural network, as demonstrated in previous studies such as [2]. To determine the optimal number of neurons for our deep learning model, we performed a comprehensive grid search process, varying the number of neurons from 1200 to 2400. Our results, as shown in Tables 3 and 4, revealed a significant correlation between the number of neurons and the performance of our model. We observed a gradual increase in accuracy, with the peak of 84.5% (Kaggle 92%) achieved with 2400 neurons (Kaggle 80 neurons). Interestingly, this number is slightly higher than the number of features in both the Ember and Kaggle datasets, highlighting the importance of tuning this hyperparameter. Our findings emphasise the crucial role that diligent hyperparameter tuning plays in improving the efficacy of deep learning models for static malware detection.

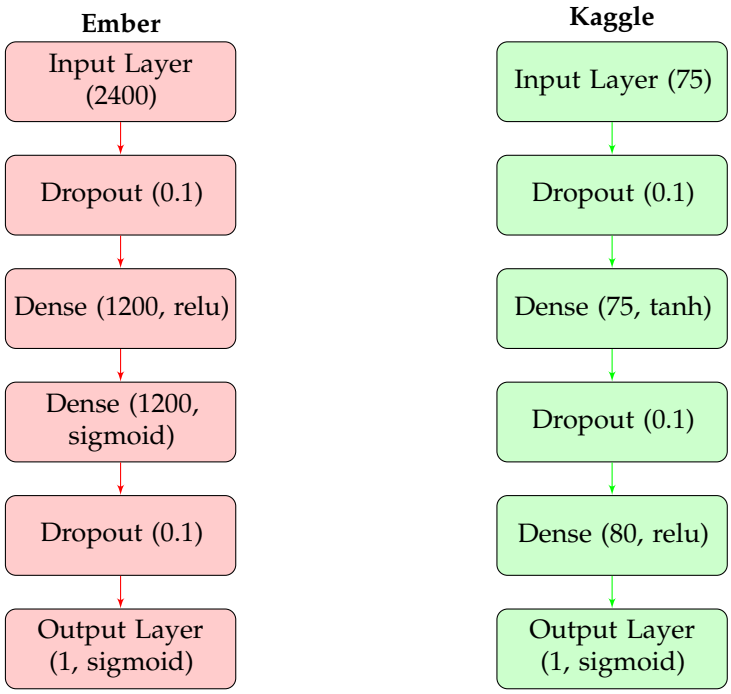

Figure 5. Optimised deep learning models for the Ember and Kaggle datasets.

6.4. Tuning the Number of Layers

The process of determining the optimal number of layers in our model was a journey of experimentation and discovery. Utilizing cAgen and grid search, we explored the potential of our search space, starting with a simple foundation of one dense layer and linear activation function. At first, our primary focus was not on the accuracy, but rather on ensuring that the model produced a plausible result. Through the process of iteration, we experimented with different configurations, observing varying accuracy results from 35% to 45%. Finally, through trial and error, we arrived at a good architecture—one dense layer with a vector size of 2381, a second dense layer of 1200, a third dense layer of 1200, and a final dense layer that provided the binary output.

Our results suggest that the optimal number of layers for our deep learning model is four, with three intermediate dense layers. This finding is consistent with previous studies on deep learning models for static malware detection, which have typically used three or four dense layers [41–43]. However, the exact architecture and number of neurons in each layer can vary significantly depending on the specific dataset and model configuration. By carefully tuning the number of layers and neurons, we were able to improve the accuracy of our model, demonstrating the importance of diligent hyperparameter tuning.

6.5. Tuning of Activation Functions

The activation functions in neural network design are of paramount importance as they greatly impact the model's performance [44]. Our experimentation with various activation functions including Softmax, Softplus, Softsign, Relu, Tanh, Sigmoid, Hard_sigmoid and Linear, through a grid search using scikit-learn grid search CV, revealed differential results across the selected functions for both datasets. The results were analysed and four activation functions were chosen based on their accuracy for both Ember and Kaggle. Hard_sigmoid, Relu, Sigmoid, and Sigmoid were selected for Ember, while Sigmoid, Tanh, Relu, and Sigmoid were selected for Kaggle. These diverse activation functions, when combined, played a crucial role in elevating the accuracy of our model. The results of our experiment demonstrate the significant impact that the choice of appropriate activation functions has on the success of the neural network model.

6.6. Tuning the Dropout Relay

The dropout relay is a crucial mechanism in preventing the over-complexity of neural networks by regularising it through approximation. It allows for parallel exploration of diverse network architectures during the training process, thereby enhancing the generalisability of the model. To the best of our knowledge, previous studies on static malware detection using deep learning did not explicitly consider the impact of dropout-relay tuning on model performance. Our empirical experimentation found that a range of values between 0.1 and 0.5 for the dropout relay resulted in an optimal increase of 1 to 3% in model accuracy. However, exceeding this range could decrease the accuracy, as evidenced by a drop from 93% accuracy to 88% when the value was set to 0.6. The placement of the dropout relay within the hidden layers also plays a significant role in determining the performance of the network, as noted by Dahl and Srivastava in their studies [3,13]. Thus, our study sheds light on the importance of dropout-relay tuning for improving the performance of deep learning models for static malware detection.

6.7. Choice of Optimisers

The significance of choosing the appropriate optimiser cannot be overstated. After thorough consideration, we opted to compare the performance of two optimisation techniques: SGD and Adam. Each optimiser has its unique set of dependencies and challenges. For instance, with SGD, determining the optimal learning rate requires experimental trials, as determining its value beforehand can be challenging, as demonstrated in prior studies [1]. Our experiments indicated that both SGD and Adam optimisers can yield high accuracy scores for our model. Our experimentation with SGD showed that the learning rate (LR) hyperparameter had a significant impact on the model's accuracy. Thus, we conducted a grid search with a learning rate range from 0.0001 to 0.5. Our findings revealed that a learning rate above 0.2 negatively impacted the model's accuracy score, with a value of 0.4 yielding an accuracy of 65%. Based on these findings, we refined our search space for the learning rate, as reflected in Table 3. On the other hand, the accuracy of the Adam optimiser was found to be dependent on the number of epochs and batch size.

It is worth noting that while previous studies on deep learning for static malware detection have explored the use of optimisers such as SGD and Adam [45,46], few have investigated the impact of hyperparameters on their performance. Our study provides valuable insights into the effects of hyperparameter tuning on the efficacy of these optimisers, thereby contributing to the development of more robust deep learning models for static malware detection. Furthermore, our comparison of the performance of SGD and Adam in our study highlights the importance of careful consideration of optimiser selection in deep learning models, particularly when it comes to static malware detection.

6.8. Tuning the Number of Epochs and Batch Size

The number of epochs and batch size are two critical parameters in the training of a machine-learning model. The batch size determines the number of samples processed before the model is updated, while the number of epochs represents the total number of complete cycles through the entire training dataset. These parameters play a vital role in shaping the performance of the model and are essential to be set before the training process begins. Finding the optimal values of these parameters is a matter of experimentation, where different combinations are tested until the best outcome is achieved.

In our case, we conducted a grid search to determine the best values for our specific problem. Our results showed that increasing the number of epochs would lead to overfitting and decrease the model's accuracy, while a low batch size would also impact the model's accuracy. We experimented with different values for both parameters and found that for the Ember dataset, the best number of epochs was 40, and the optimal batch size was 64. Our findings align with previous studies [15], as well as with the Kaggle dataset, which had the best number of epochs as 40 and the best batch size as 60. Our study confirms the importance of carefully tuning the number of epochs and batch size to achieve optimal

performance in the neural network model for static malware detection. The results of our experiments are documented in Tables 3 and 5.

7. Conclusions

After conducting a comprehensive study on the feasibility of utilising deep learning models to improve static malware detection through hyperparameter optimisation using covering arrays, our findings indicate a highly promising approach. The performance of our optimised deep learning models using covering arrays outperformed the baseline model on both the Ember and Kaggle datasets. Specifically, our optimised model achieved an accuracy of 95.7% on the Ember dataset and 98.6% on the Kaggle dataset, highlighting the potential of this approach to significantly enhance the performance of malware detection systems.

Moreover, our study provides evidence that incorporating hyperparameter optimisation through covering arrays can be a highly effective technique for enhancing the performance of deep learning models in the field of malware detection. Our results indicate that fine-tuning deep learning models using covering arrays significantly outperformed the traditional grid search method in terms of time and computational resources required. This finding is particularly significant as it demonstrates that the use of covering arrays is not only more efficient, but also leads to improved accuracy.

It is worth noting that the optimal configuration of hyperparameters, such as the number of neurons, layers, activation functions, dropout relay, number of epochs, and batch size, was carefully determined through a rigorous and comprehensive experimentation process. The use of covering arrays allowed for the exploration of a wide range of hyperparameter configurations, enabling us to fine-tune our models to achieve the highest accuracy possible.

In conclusion, our study provides further evidence that deep learning models, combined with hyperparameter optimisation using covering arrays, can significantly enhance the performance of static malware-detection systems. The results of our study suggest that this approach has the potential to be widely adopted in the field of cybersecurity, leading to more efficient and accurate malware-detection systems. Future work could investigate the application of this approach to other domains, and it has the potential for further optimisation techniques.

Author Contributions: Conceptualization, F.T.A. and A.S.A. Authors Contributed equally. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The Ember Dataset can be found on (https://github.com/elastic/ember, accessed 23 April 2023) and the Kaggle dataset can be found on (https://www.kaggle.com/datasets/amauricio/pe-files-malwares, accessed 23 April 2023) .

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

References

1. Sun, S.; Cao, Z.; Zhu, H.; Zhao, J. A survey of optimization methods from a machine learning perspective. *IEEE Trans. Cybern.* **2019**, *50*, 3668–3681. [CrossRef] [PubMed]
2. Shafi, I.; Ahmad, J.; Shah, S.I.; Kashif, F.M. Impact of varying neurons and hidden layers in neural network architecture for a time frequency application. In Proceedings of the 2006 IEEE International Multitopic Conference, Islāmābād, Pakistan, 23–24 December 2006. [CrossRef]
3. Dahl, G.E.; Sainath, T.N.; Hinton, G.E. Improving deep neural networks for LVCSR using rectified linear units and dropout. In Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 26–31 May 2013; pp. 8609–8613.
4. Lei, Y.; Kacker, R.; Kuhn, D.R.; Okun, V.; Lawrence, J. IPOG: A general strategy for t-way software testing. In Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07), Tucson, AZ, USA, 26–29 March 2007; pp. 549–556.

5. Cohen, D.M.; Dalal, S.R.; Fredman, M.L.; Patton, G.C. The AETG system: An approach to testing based on combinatorial design. *IEEE Trans. Softw. Eng.* **1997**, *23*, 437–444. [CrossRef]
6. Bryce, R.C.; Colbourn, C.J. The density algorithm for pairwise interaction testing. *Softw. Test. Verif. Reliab.* **2007**, *17*, 159–182. [CrossRef]
7. Bryce, R.C.; Colbourn, C.J. A density-based greedy algorithm for higher strength covering arrays. *Softw. Test. Verif. Reliab.* **2009**, *19*, 37–53. [CrossRef]
8. Lei, Y.; Tai, K.C. In-parameter-order: A test generation strategy for pairwise testing. In Proceedings of the Third IEEE International High-Assurance Systems Engineering Symposium (Cat. No. 98EX231), Washington, DC, USA, 13–14 November 1998; pp. 254–261.
9. Torres-Jimenez, J.; Izquierdo-Marquez, I. Survey of covering arrays. In Proceedings of the 2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, 23–26 September 2013; pp. 20–27.
10. Seroussi, G.; Bshouty, N.H. Vector sets for exhaustive testing of logic circuits. *IEEE Trans. Inf. Theory* **1988**, *34*, 513–522. [CrossRef]
11. Group, M.R. Covering Array Generation. 2022. Available online: https://matris.sba-research.org/tools/cagen/#/about (accessed on 21 July 2022).
12. Yin, W.; Kann, K.; Yu, M.; Schütze, H. Comparative study of CNN and RNN for natural language processing. *arXiv* **2017**, arXiv:1702.01923.
13. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
14. Jason, B. How to Configure the Number of Layers and Nodes in a Neural Network. Available online: https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/ (accessed on 23 April 2023).
15. Kandel, I.; Castelli, M. The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT Express* **2020**, *6*, 312–315. [CrossRef]
16. Bergstra, J.; Bengio, Y. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* **2012**, *13*, 281–305.
17. Bergstra, J.; Bardenet, R.; Bengio, Y.; Kégl, B. Algorithms for hyper-parameter optimization. *Adv. Neural Inf. Process. Syst.* **2011**, *24*, 1–9.
18. Bengio, Y. Gradient-based optimization of hyperparameters. *Neural Comput.* **2000**, *12*, 1889–1900. [CrossRef]
19. Snoek, J.; Larochelle, H.; Adams, R.P. Practical bayesian optimization of machine learning algorithms. *arXiv* **2012**, arXiv:1206.2944.
20. Hutter, F.; Hoos, H.H.; Leyton-Brown, K. Sequential model-based optimization for general algorithm configuration. In Proceedings of the International Conference on Learning and Intelligent Optimization, Rome, Italy, 17–21 January 2011; pp. 507–523.
21. Karnin, Z.; Koren, T.; Somekh, O. Almost optimal exploration in multi-armed bandits. In Proceedings of the International Conference on Machine Learning, Atlanta, GA, USA, 16–21 June 2013; pp. 1238–1246.
22. Injadat, M.; Moubayed, A.; Nassif, A.B.; Shami, A. Systematic ensemble model selection approach for educational data mining. *Knowl.-Based Syst.* **2020**, *200*, 105992. [CrossRef]
23. Claesen, M.; Simm, J.; Popovic, D.; Moreau, Y.; De Moor, B. Easy hyperparameter search using optunity. *arXiv* **2014**, arXiv:1412.1114.
24. Anderson, H.S.; Roth, P. Elastic/Ember. 2021. Available online: https://github.com/elastic/ember/blob/master/README.md (accessed on 23 April 2023)
25. Mauricio. Benign Malicious. 2021. Available online: https://www.kaggle.com/amauricio/pe-files-malwares (accessed on 10 November 2021).
26. Schultz, M.G.; Eskin, E.; Zadok, F.; Stolfo, S.J. Data mining methods for detection of new malicious executables. In Proceedings of the 2001 IEEE Symposium on Security and Privacy, S&P 2001, Oakland, CA, USA, 14–16 May 2001; pp. 38–49.
27. Kolter, J.Z.; Maloof, M.A. Learning to detect and classify malicious executables in the wild. *J. Mach. Learn. Res.* **2006**, *7*, 2721–2744.
28. Raff, E.; Barker, J.; Sylvester, J.; Brandon, R.; Catanzaro, B.; Nicholas, C.K. Malware detection by eating a whole exe. In Proceedings of the Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018.
29. Pham, H.D.; Le, T.D.; Vu, T.N. Static PE malware detection using gradient boosting decision trees algorithm. In Proceedings of the International Conference on Future Data and Security Engineering, Ho Chi Minh City, Vietnam, 28–30 November 2018; pp. 228–236.
30. Fawcett, C.; Hoos, H.H. Analysing differences between algorithm configurations through ablation. *J. Heuristics* **2016**, *22*, 431–458. [CrossRef]
31. Azeez, N.A.; Odufuwa, O.E.; Misra, S.; Oluranti, J.; Damaševičius, R. Windows PE Malware Detection Using Ensemble Learning. *Informatics* **2021**, *8*, 10. [CrossRef]
32. ALGorain, F.T.; Clark, J.A. Covering Arrays ML HPO for Static Malware Detection. *Eng* **2023**, *4*, 543–554. [CrossRef]
33. Pérez-Espinosa, H.; Avila-George, H.; Rodriguez-Jacobo, J.; Cruz-Mendoza, H.A.; Martínez-Miranda, J.; Espinosa-Curiel, I. Tuning the parameters of a convolutional artificial neural network by using covering arrays. *Res. Comput. Sci.* **2016**, *121*, 69–81. [CrossRef]
34. Forbes, M.; Lawrence, J.; Lei, Y.; Kacker, R.N.; Kuhn, D.R. Refining the in-parameter-order strategy for constructing covering arrays. *J. Res. Natl. Inst. Stand. Technol.* **2008**, *113*, 287. [CrossRef]
35. Lei, Y.; Kacker, R.; Kuhn, D.R.; Okun, V.; Lawrence, J. IPOG/IPOG-D: Efficient test generation for multi-way combinatorial testing. *Softw. Test. Verif. Reliab.* **2008**, *18*, 125–148. [CrossRef]

36. Duan, F.; Lei, Y.; Yu, L.; Kacker, R.N.; Kuhn, D.R. Improving IPOG's vertical growth based on a graph coloring scheme. In Proceedings of the 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Graz, Austria, 13–17 April 2015; pp. 1–8.
37. Anderson, H.S.; Roth, P. Ember: An open dataset for training static pe malware machine learning models. *arXiv* **2018**, arXiv:1804.04637.
38. Carrera, E. pefile. 2022. Available online: https://github.com/erocarrera/pefile (accessed on 15 January 2022).
39. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
40. sklearn. sklearn-StandardScaler. 2022. Available online: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html#sklearn.preprocessing.StandardScaler (accessed on 6 July 2022).
41. Saxe, J.; Berlin, K.; Vishwanathan, S. EXPOSE: A character-level convolutional neural network for predicting malware behavior. *arXiv* **2015**, arXiv:1510.07391.
42. Rajaraman, S.; Huang, Y.H.; Kim, H. Malware detection using deep neural network with multiple learning rates. In Proceedings of the 2018 IEEE International Conference on Electro Information Technology (EIT), Rochester, MI, USA, 3–5 May 2018; pp. 679–683.
43. Seo, H.; Lee, J.; Lee, J. Malware detection using a hybrid convolutional neural network and long short-term memory model. *Inf. Sci.* **2020**, *516*, 423–436.
44. Hayou, S.; Doucet, A.; Rousseau, J. On the impact of the activation function on deep neural networks training. In Proceedings of the International Conference on Machine Learning, PMLR, Long Beach, CA, USA, 10–15 June 2019; pp. 2672–2680.
45. Huang, Y.; Xu, X.; Zhou, X.; Wu, G. Deep learning-based malware detection: A review. *Comput. Secur.* **2020**, *92*, 101716.
46. Shafiq, M.; Mustafa, K.; Yaqoob, I.; Saleem, K.; Makhdoom, I.; Abbas, H. Automated malware classification using ensemble with feature selection. In Proceedings of the 2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET), Sukkur, Pakistan, 3–4 March 2018; pp. 1–6.