electronics

MDPI

Article

Ensemble-Based Classification Using Neural Networks and Machine Learning Models for Windows PE Malware Detection

Robertas Damaševičius [1,*], Algimantas Venčkauskas [2], Jevgenijus Toldinas [2] and Šarūnas Grigaliūnas [2]

[1] Department of Software Engineering, Kaunas University of Technology, 44249 Kaunas, Lithuania
[2] Department of Computer Science, Kaunas University of Technology, 44249 Kaunas, Lithuania; algimantas.venckauskas@ktu.lt (A.V.); eugenijus.toldinas@ktu.lt (J.T.); sarunas.grigaliunas@ktu.edu (Š.G.)
* Correspondence: robertas.damasevicius@ktu.lt

Abstract: The security of information is among the greatest challenges facing organizations and institutions. Cybercrime has risen in frequency and magnitude in recent years, with new ways to steal, change and destroy information or disable information systems appearing every day. Among the types of penetration into the information systems where confidential information is processed is malware. An attacker injects malware into a computer system, after which he has full or partial access to critical information in the information system. This paper proposes an ensemble classification-based methodology for malware detection. The first-stage classification is performed by a stacked ensemble of dense (fully connected) and convolutional neural networks (CNN), while the final stage classification is performed by a meta-learner. For a meta-learner, we explore and compare 14 classifiers. For a baseline comparison, 13 machine learning methods are used: K-Nearest Neighbors, Linear Support Vector Machine (SVM), Radial basis function (RBF) SVM, Random Forest, AdaBoost, Decision Tree, ExtraTrees, Linear Discriminant Analysis, Logistic, Neural Net, Passive Classifier, Ridge Classifier and Stochastic Gradient Descent classifier. We present the results of experiments performed on the Classification of Malware with PE headers (ClaMP) dataset. The best performance is achieved by an ensemble of five dense and CNN neural networks, and the ExtraTrees classifier as a meta-learner.

Keywords: malware analysis and detection; applied machine learning; mobile security; neural network; ensemble classification

Citation: Damaševičius, R.; Venčkauskas, A.; Toldinas, J.; Grigaliūnas, Š. Ensemble-Based Classification Using Neural Networks and Machine Learning Models for Windows PE Malware Detection. *Electronics* **2021**, *10*, 485. https://doi.org/10.3390/electronics10040485

Academic Editor: Suleiman Yerima
Received: 11 January 2021
Accepted: 16 February 2021
Published: 18 February 2021

1. Introduction

Many aspects of society have shifted online with the broad adoption of digital technology, from entertainment and social interactions to business, entertainment, industry and, unfortunately, crime as well. Cybercrime is rising in frequency and magnitude in recent years, with a projection of reaching USD 6 trillion by 2021 (up from USD 3 trillion in 2015) [1] and also taking on conventional crime both in number and revenues [2]. Additionally, these new cyber-attacks have become more complex [3], generating elaborate multi-stage attacks. By the end of 2018, about 9599 malicious packages appeared per day [4]. Such attacks also resulted in significant damage and major financial losses. Up to USD 1 billion was stolen from financial institutions around the world in two years due to malware [5]. In addition, Kingsoft estimated that between 2 and 5 million computers were attacked each day [6]. With cybercrime revenues reaching USD 1.5 trillion in 2018 [7] and cybercrime's global cost predicted to reach USD 6 trillion by 2021 [8], addressing cyber threats has become an urgent issue.

Moreover, the COVID-19 pandemic has delivered an extraordinary array of cybersecurity challenges, as most services have moved to online and remote mode, raising the danger of cyberattacks and malware [9,10]. Especially, in the healthcare sector, cyber-attacks can lead to compromised sensitive personal patient data, while data tampering can lead to incorrect treatment, with irreparable damage to patients [11].

Today, computer programs and applications are developed at high speed. Malicious software (malware) has appeared and is growing in many formats and is becoming increasingly sophisticated. Computer criminals use them as a tool to infiltrate, steal or falsify information, causing huge damage to individuals, businesses and even threatening national security. A generic term generally used to describe all various types of unauthorized software programs is malware (malicious software), which includes viruses, worms, Trojans, spyware [12], Android malicious apps [13], bots, rootkits [14] and ransomware [15]. In achieving its objectives, malware has been used by cybercriminals as weapons. Malware has been used to conduct a wide variety of security threats, such as stealing confidential data, stealing cryptocurrency, sending spam, crippling servers, penetrating networks and overloading critical infrastructures. While large numbers of malware samples have been identified and blocked by cybersecurity service providers and antivirus software manufacturers, a significant number of malware samples have been created or mutated (e.g., "zero-day" malware [16]) and appear to evade conventional anti-virus scanning tools based on signatures. As these techniques are primarily based on modifications of signature-based models, this has caused the information security industry to reconsider their malware recognition techniques.

Malware detection methods can be classified into methods based on signatures and behavior. Currently, signature-based malware detectors can work effectively with previously known malware that has already been detected by some anti-malware vendors. However, it cannot detect polymorphic malware that can change its signatures, as well as new malware whose signatures have not yet been created. One solution to this problem is to use heuristic analysis in combination with machine learning techniques that provide higher detection efficiency. As practice has shown, the traditional approach to the field of malware detection, which is based on signature analysis [17], is not acceptable for detecting unknown computer viruses. To maintain the proper level of protection, users are forced to constantly and timely update anti-virus databases. However, the delay in the response from the anti-virus companies for the emergence of new malware (its detection and signature creation) can vary from several hours to several days. During this time, malicious new programs can cause irreparable damage.

To address this problem, in addition to the signature approach, heuristic analysis is used. At the same time, the file can be considered "potentially dangerous" with some probability based on its behavior (dynamic approach) or the analysis of its structure (static approach). Static analysis generally consists of two main stages: the training stage and the stage of using the results (detection of virus programs). At the training stage, a sample of infected (virus) and "clean" (legitimate) files is formed. In the structure of the files, some signs characterize each of them as viral or legitimate. As a result, a list of feature characteristics is compiled for each file. Next, the most significant (informative) features are selected, and redundant and irrelevant features are discarded. At the detection stage, feature characteristics are extracted from the scanned file. Heuristic algorithms developed specifically to detect unknown malware are characterized by a high error rate. Heuristic-based detection uses rules formulated by experts to distinguish between malicious and benign files. Additionally, behavior-based, model checking-based and cloud-based methods have performed effectively in malware detection [18].

Modern research in the area of information security aimed at creating such protection methods and algorithms that would be able to detect and neutralize unknown malware, and thus not only increase the computer security but also save the user from constant updates of antivirus software. The size of gray lists is constantly growing with the advancement of malware writing and production techniques. Intelligent methods for automatically detecting malware are, therefore, urgently required. As a result, several studies have been published on the development of smart malware recognition systems using artificial intelligence methods [19–22].

A prerequisite for creating effective anti-virus systems is the development of artificial neural network (ANN)-based technologies. The ability of such systems to learn and

generalize results makes it possible to create smart information security systems. Artificial intelligence (AI) has several advantages when it comes to cybersecurity: AI can discover new previously unknown attacks; AI can handle a high volume of data; AI-based cybersecurity systems can learn over time to respond better to threats [23].

This study aims to implement an ensemble of neural networks for the detection of malware. The novel contributions of this paper are the following:

(1) The detailed experimental analysis and verification of machine learning and deep learning methods for malware recognition performed on the Classification of Malware with PE headers (ClaMP) dataset;
(2) A novel ensemble learning-based hybrid classification framework for malware detection with a heterogeneous batch of convolutional neural networks (CNNs) as base classifiers and a machine learning algorithm as a final-stage classifier, which allows us to achieve the improvement of malware detection accuracy;
(3) An extensive ablation study to select CNN model architectures and a machine learning algorithm for the best overall malware detection performance.

The other parts of this study are structured as follows. In Section 2, related works are discussed including the presentation of adequate criticism of existing methods and approaches. Section 3 describes the methodology used in this paper. Section 4 discusses the implementation and results obtained. Section 5 presents the conclusion of the study.

2. Related Works

Malware search algorithms are divided into two classes based on the method of collecting information—dynamic and static. In static analysis, suspicious objects are considered without starting them, based on the assembly code and attributes of executable files [24]. Dynamic analysis algorithms work either with already running programs or run them themselves in an isolated environment, exposing the information that has arisen in the course of work: they analyze the behavior of the program, sections of code and data and monitor resource consumption [25]. According to the type of objects detected, malware search algorithms are divided into signature and anomalous ones. Signature programs tend to highlight the signatures of malware. Anomaly detection algorithms seek to describe legitimate programs and learn to look for deviations from the norm.

At the same time, machine learning is also widely used as a powerful tool for security experts to identify malicious programs with high accuracy, when the number of malicious programs is high enough, and their options have become diverse. Among the main methods is the Windows Portable Executable 32-bit (PE32) file header analysis [26]. For example, Nisa et al. [27] transformed malware code into images and applied segmentation-based fractal texture analysis for feature extraction. Deep neural networks (AlexNet and Inception-v3) were used for classification. Previously, the use of ensemble methods, such as random forest and extremely randomized trees, allowed the improvement of the performances of machine learning models in detecting malware in internet of things (IoT) environments [28] and Wireless Sensor Networks (WSN) [29].

Many studies are being performed to analyze malware to curb the increase in malicious software [30]. The existing deep learning-based malware analysis methods include convolutional neural networks (CNN) [31], deep belief network (DBN) [32], graph convolutional network (GCN) [33], LSTM and Gated Recurrent Unit (GRU) [34], VGG16 [35] and generative adversarial networks (GAN) [36]. However, it is not possible to guarantee the generalization potential of artificial neural network-based models [37].

To solve the above-mentioned problems, more general and robust methods are, therefore, required. Researchers are creating numerous ensemble classifiers [38–42] that are less susceptible to malware feature collection. Ensemble methods [43] are a class of techniques that incorporate several learning algorithms to enhance the precision of overall prediction. To minimize the risk of overfitting in the training results, these ensemble classifiers integrate several classification models. In this way, the training dataset can be more effectively used, and generalization efficiency can be increased as a result. While several models of

ensemble classification are already developed, there is still space for researchers to improve the accuracy of sample classification, which would be useful for improving malware detection.

Therefore, this paper proposes an ensemble earning-based approach for using fully connected and convolution neural networks as base learners for malware detection.

3. Materials and Methods

Malware developers are primarily focused on targeting computer networks and infrastructure to steal information, make financial demands or prove their potential. The standard approaches for detecting malware were effective in detecting known malware. Via these approaches, however, new malware can never be blocked. The latest machine learning platform [44] has significantly enhanced the identification capability of models used for malware detection. It is possible to detect malware using machine learning methods in two steps, namely, extracting features from input data and choosing important ones that best represent the data, and classifying/clustering. The technology proposed is focused on machine learning that can learn and discern malicious and benign files, as well as make reliable forecasts of new files that have not been seen before.

The phases involved in achieving the final solution are (1) data processing and feature selection and (2) model engineering, which includes the following steps: data selection and scaling, reduction in dimensionality, ANN model exploration and meta-learner classifier selection, ensemble model development, model testing and performance evaluation. Figure 1 indicates the flow to the model evaluation stage of the stages involved in the system methodology, beginning with data selection, which is described in more depth in the following subsections.

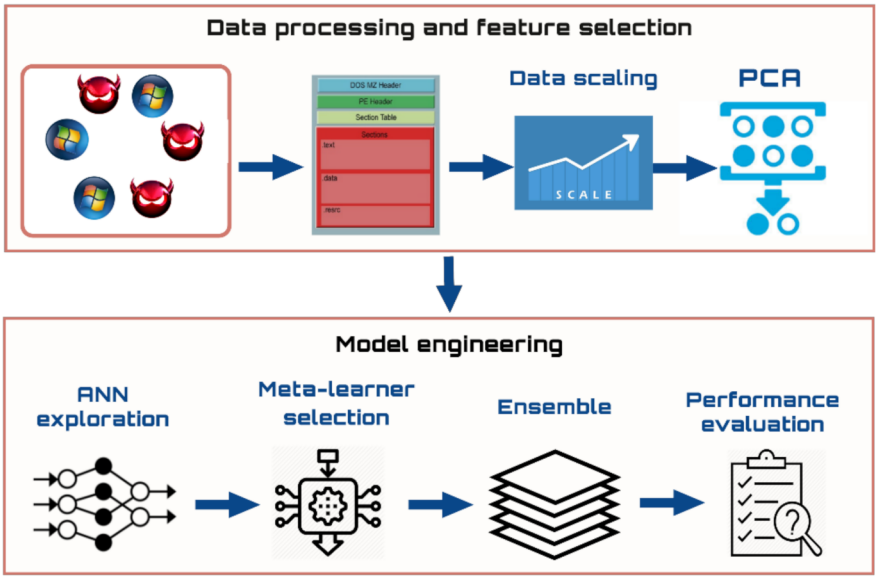

Figure 1. Outline of malware detection methodology.

3.1. Data Collection and Processing

For machine learning to be a success, the selection of a representative dataset is necessary. This is because it is important to train a machine learning algorithm on a dataset that correctly represents the conditions for the model's real-world applications.

For this model, the dataset gathered contains malicious and benign data from the Classification of Malware with PE headers (ClaMP) dataset, obtained from GitHub. We used the ClaMP_Integrated dataset, which has 2722 malware and 2488 Benign instances. The dataset has 69 features, which include, among others, the following features:

- DOS image header: e_cp–pages in file, e_cblp–bytes on the last page, e_cparhdr–size of header, e_crlc–number of relocations, e_cs–initial CS value, e_csum - checksum,

e_ip–initial IP value, e_lfanewe_lfarlc, e_magic–Magic number, e_maxalloc–maximum extra paragraphs, e_minalloc–minimum number of extra paragraphs, e_oemid–OEM ID, e_oeminfo–OEM information, e_ovno–overlay number, e_res and e_res2–reserved words, e_sp–initial SP value, e_ss–initial SS value.

- File header features: CharacteristicsCreationYear, Machine, NumberOfSections, NumberOfSymbols, PointerToSymbolTable, SizeOfOptionalHeader.
- Other raw features: AddressOfEntryPoint, BaseOfCode, BaseOfData, CheckSum, DllCharacteristics, FileAlignment, ImageBase, LoaderFlags, Magic, MajorImageVersion, MajorLinkerVersion, MajorOperatingSystemVersion, MajorSubsystemVersion, MinorImageVersion, MinorLinkerVersion, MinorOperatingSystemVersion, MinorSubsystemVersion, NumberOfRvaAndSizes, SectionAlignment, SizeOfCode, SizeOfHeaders, SizeOfHeapCommit, SizeOfHeapReserve, SizeOfImage, SizeOfInitializedData, SizeOfStackCommit, SizeOfStackReserve, SizeOfUninitializedData, Subsystem.
- Derived features: sus_sections, non_sus_sections, packer, packer_type, E_text, E_data, filesize, E_file, fileinfo.

However, we used only 68 features (all numerical), because one feature "packer_type" is a string, which was not used. The numerical features were scaled using the standard scaling method. These features, along with the class label (0 for benign and 1 for malicious), were used to build the ensemble classification model.

3.2. Dimensionality Reduction

To fix a variety of estimation and classification questions, machine learning methods are commonly used. Bad machine learning output can be triggered by overfitting or underfitting the results. Removing the unimportant characteristics guarantees the algorithms' optimal efficiency and improves pace. Principal Component Analysis (PCA) was introduced to perform attribute dimensionality reduction. Based on previous studies, 40 features were chosen to be passed into the machine learning model (representing 95% of the total variability in the dataset), because these features are critical in neural network learning, whether a file is malicious or benign.

3.3. Deep Learning Models

As deep learning models, we considered fully connected (FC) multilayer perceptron (MLP) and one-dimensional convolutional neural networks (1D-CNN), which are discussed in detail below.

3.3.1. Multilayer Perceptron

As a baseline approach, we adopted a simple multilayer perceptron (MLP). Let the output of the MLP be known $y(t)$ at the input $X(t)$, where $X(t)$ is a vector with components $(x_1, x_2, \ldots, x_n)$, t is the number of the sequence value and $t = \overline{1, T}$ (T is predetermined).

To find model parameters $w = (w_0, w_1, \ldots, w_m)$ and $V_k = (V_{1k}, V_{2k}, \ldots, V_{nk})$, h_k, $k = \overline{1, m}$ such that the model output $F(X, V, w)$ and the real output of the MLP $y(t)$ would be as close as possible. The relationship between the input and output of a two-layer perceptron is established by the following relationships:

$$Z_k = \sigma(V_{1k}x_1 + V_{2k}x_2 + \ldots V_{nk}x_n - h_k), k = \overline{1, m} \quad (1)$$

$$y = \sigma(w_1Z_1 + w_2Z_2 + \ldots w_mZ_m + w_0) \quad (2)$$

The following expression describes a perceptron with one hidden layer, which is able to approximate any continuous function defined on a bounded set.

$$\sum_{k=1}^{m} w_k \cdot \sigma(V_{1k}x_1 + V_{2k}x_2 + \ldots V_{nk}x_n - h_k) \; + \; w_0 \quad (3)$$

Training of MLP occurs by applying a gradient descent algorithm (such as error backpropagation) similar to a single-layer perceptron.

3.3.2. One-Dimensional Convolutional Neural Network (1D-CNN)

While CNN models have been developed for image processing, where an internal representation of a two-dimensional input (2D) is learned by the model, the same mechanism can be used in a process known as feature learning on one-dimensional (1D) data sequences, such as in the case of malware detection. The model understands how to extract features from observational sequences and how to map hidden layers to different types of software (malware or benign).

$$\hat{y} = \Phi([x_1, \ldots, x_N]), \tag{4}$$

where $X : x_1, \ldots, x_N$ indicates the input of the network, and $Y : \hat{y}$ is the output. Therefore, the network learns a mapping from the input space X to the output space Y.

The key block of the convolutional network is the convolutional layer. A group of trainable filters are the parameters of this layer (scan windows). Each filter operates in size through a tiny window. The scanning window sequentially traverses the whole picture during the forward propagation of the signal (from the first layer to the last layer) according to the tiling principle and measures the dot products of two vectors: the filter values and the outputs of the chosen neurons. Thus, a two-dimensional activation map is generated after passing all the shifts in the width and height of the input field, which gives the effect of applying a particular filter in each spatial area. The network uses filters that are enabled when there is an input signal of some kind. A series of filters are used for each convolutional layer, and each generates a different activation map.

$$x_j^l = f\left(\sum_{i=1}^{M} x_i^{l-1} \cdot k_{ij}^l + b_j^l\right), \tag{5}$$

where k is the convolution kernel, j is the size of kernels, M is the number of inputs x_i^{l-1}, b, is kernel bias, $f(\ )$ is the neuron activation function and $(\cdot)$ represents the convolution operator.

The sub-sampling layer is another feature of a convolutional neural network. It is usually positioned between successive convolution layers, so it may occur periodically. Its purpose is to reduce the spatial size of the vector gradually to reduce the number of network parameters and calculations, as well as to balance overfitting. The convolution layer resizes the feature map, using the max operation most frequently. If the output from the previous layer is to be fed to the fully connected layer, the flattening layer is used, and then it needs to be flattened. The layer of the Parametric Rectified Linear Unit (PReLU) is an activation function that complements the rectified unit with a negative value slope.

The dropout layer is used to regularize the network. It also makes it possible to be thinner for the network size. The neurons that are less likely to raise the weight of learning are randomly removed. The practical importance of dropout unit is to prevent overfitting [45]. This dropout layer, as we have two classes, is succeeded by a fully linked (dense) layer that reduces the final output vector to two classes, and we expect the program's behavior to be either malicious or benevolent. The final activation function is SoftMax, which shrinks the two outputs to one.

The output of each convolutional layer in 1D-CNN is also the input of the subsequent layer. It also represents the weights learned by the convolution kernel from the training samples.

A unique and essential part of CNNs is the fully connected (FC) layer, which outputs a final output. The output of the network's previous layers is reshaped into a single vector (flattened). Any of them reflects the probability that a class label is a special function. The final probabilities for each label are supplied by the output of the FC layer.

3.4. Network Model Optimization

Optimization of neural network hyper-parameters, which rule how the network operates and governs its accuracy and validity, is still an unsolved problem. Optimizers adjust the parameters of neural networks, such as weight and learning rate, to minimize loss. Known examples of neural network optimization algorithms are Stochastic Gradient Descent (SGD) [46], AdaGrad [47], RMSProp [48] and Adam [49], which usually show a tradeoff of optimization vs. generalization. This means that higher training speed and higher accuracy in the training may result in poorer accuracy on the testing dataset. Here, we adopted the Exponential Adaptive Gradients (EAG) optimization [50], which combines Adam and AdaBound [51]. During training, it exponentially sums the gradient in the past and adaptively adjusts the learning rate to address poor generalization of the Adam optimizer.

3.5. Ensemble Classification

The basic principle of ensemble methods is that training datasets are rearranged in several ways (either by resampling or reweighting) and by adding a base classifier to each rearranged training dataset, an ensemble of base classifiers is built. After that, a new ensemble classifier is developed using the stacked ensemble method by combining the prediction effects of all those base classifiers, where a new model learns how to better integrate predictions from multiple base models. We used the two-stage stacking technique [52]. First, several models are trained based on a dataset. Then, the output of each of the models is processed to create a new dataset. The actual value it is supposed to approximate is related to each instance in the current dataset. Second, the dataset with the meta-learning algorithm is used to provide the final output.

In the design of a stacking model (Figure 2), base models are often referred to as level-0 models, and a meta-learner (or generalizer) that integrates base model projections, referred to as a level-1 model, is involved. Models that fit into the training data and are compiled with forecasts are the base models. The meta-learner (level-1 model) is a classification model trained to combine the predictions of the base model. The meta-learner is informed by simple models on the choices made. To train the base models, a new batch of previously unused data is used and predictions are made, and the input and output value pairs of the training dataset are used to fit the meta-learner, along with projected outputs given by these predictions.

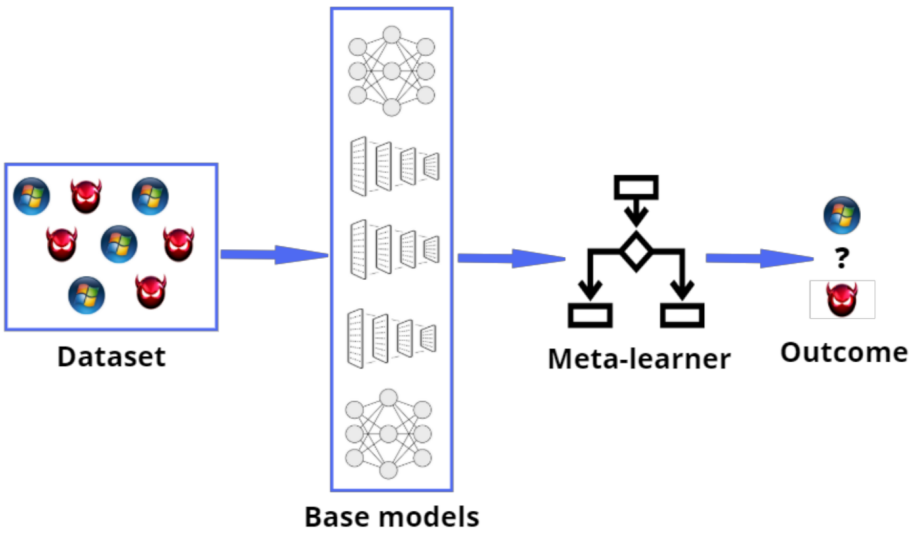

Figure 2. Schematics of ensemble classification approach.

The ensemble learner algorithm consists of three stages:

1 Set up the ensemble:
   (a) Select N base learners;
   (b) Select a meta-learning algorithm.

2 Train the ensemble:
   (a) Train each of the N base learners on the training dataset $\{X_1, X_2, \ldots, X_M\}$, where M is the number of samples;
   (b) Perform the k-fold cross-validation on each of the base learners and record the cross-validated predictions $\{y_1, y_2, \ldots, y_N\}$;
   (c) Combine cross-validated predictions from base learners to form a new feature matrix as follows. Train the meta-learner on the new data (features x predictions from base-level classifiers) $\{(X_1, X_2, \ldots, X_M, y_1), (X_1, X_2, \ldots, X_M, y_2), \ldots, (X_1, X_2, \ldots, X_M, y_N)\}$. Combine base learning models and the meta-learner to generate more accurate predictions on unknown data.

3 Test on new data:
   (a) Record output decisions from the base learners;
   (b) Send base-level decisions to the meta-learner to make ensemble decision.

On the training dataset, stacking capitalizes over every single best learner. Usually, the greatest gains are made when base classifiers used for stacking have high variability and uncorrelated outputs predicted values. As base models, we used the following neural networks: fully connected MLP with one hidden layer (Dense-1), fully connected MLP with two hidden layers (Dense-2) and one-dimensional CNN (1D-CNN). The configurations of neural networks are summarized in Table 1.

Table 1. Model configuration of neural networks with their parameters. FC—fully connected. Conv1D—one-dimensional convolution. PReLU—Parametric Rectified Linear Unit.

Dense-1 Network	Dense-2 Network	1D-CNN Network
	Parameters	
X—number of neurons in 1st hidden layer	X—number of neurons in 1st hidden layer Y—number of neurons in 2nd hidden layer	F—number of filters in convolutional layers N—number of neurons in dense layer
	Input layer of 40 × 1 features	
1 FC layer (X neurons)	1 FC layer (X neurons)	2 Conv1D layers (F 2 × 2 filters)
PReLU	PReLU	Max-pooling layer
Dropout layer (p = 0.3)	Dropout layer (p = 0.3)	2 Conv1D layers (F 2 × 2 filters)
1 FC layer (2 neurons)	1 FC layer (Y neurons)	Max-pooling layer
	PReLU	1 FC layer (N neurons)
Softmax output layer	Dropout layer (p = 0.3)	Dropout layer (p = 0.5)
	1 FC layer (2 neurons)	1 FC layer (2 neurons)
	Softmax output layer	Softmax output layer

The examples of neural network architectures are presented in Figure 3.

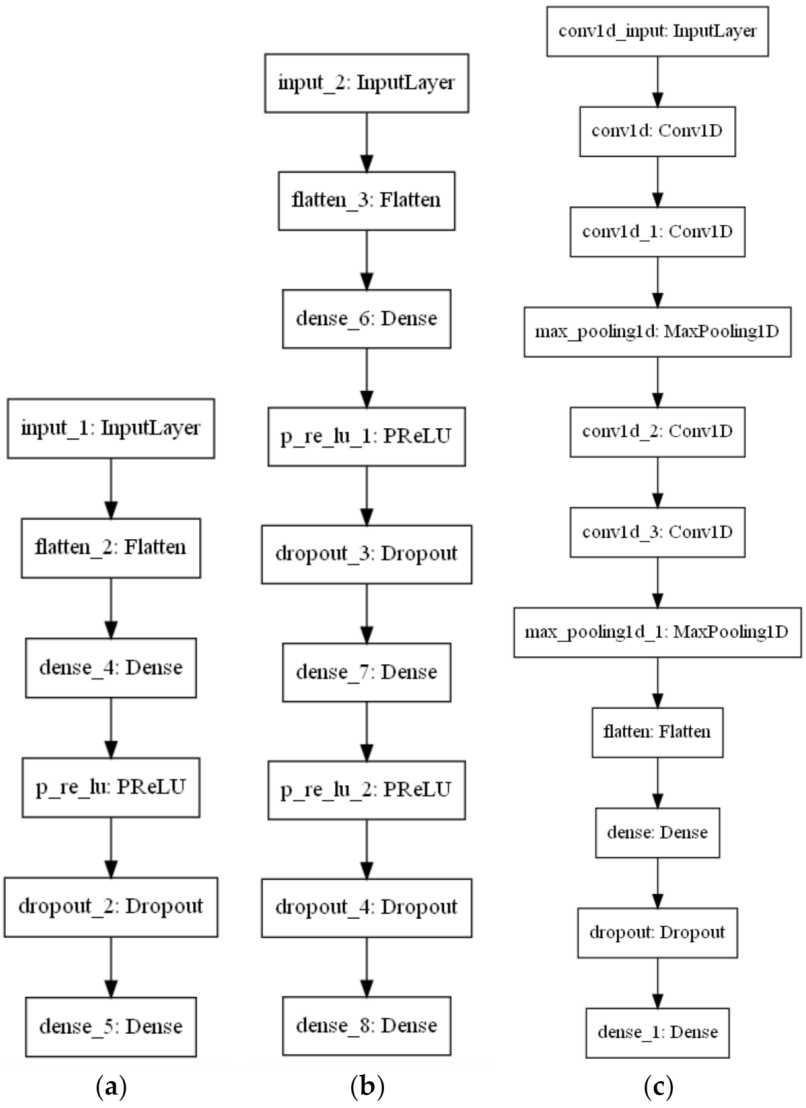

Figure 3. Example of architectures used as base learners: (**a**) Dense-1 network architecture, (**b**) Dense-2 network architecture and (**c**) 1D-CNN network architecture.

The role of the meta-learner is to find how best to aggregate the decisions of the base classifiers. As meta-learners, we explored K-Nearest Neighbors (KNN), Support Vector Machine (SVM) with linear kernel, SVM with radial basis function (RBF) kernel, Decision Tree (DT), Random Forest (RF), Multi-Layer Perceptron (MLP), AdaBoost Classifier, Extra-Trees (ET) classifier, Isolation Forest, Gaussian Naïve Bayes (GNB), Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), Logistic Regression (LR), Ridge Classifier (RC) and stochastic gradient descent classifier (SGDC). Here, KNN is a model that classifies unknown input data based on having the most similarity (least distance) to known input data. SVM is a supervised learning method that constructs a higher dimensional hyperplane to separate input data belonging to various classes while maximizing the data input distance to the hyperplane. The DT classifier creates a decision tree by splitting according to the feature which has the highest information gain. RF fits many DT classifiers on different sub-samples of the dataset and uses averaging to improve the prediction accuracy. AdaBoost fits a classifier on the dataset and performs weighting of incorrectly classified instances to improve accuracy. Isolation Forest performs classification based on identified anomalies in data. GNB performs classification based on the probability distributions of features and classes. The ET Classifier [53] creates a meta-estimator that fits multiple decision trees on the training dataset sub-samples and uses averaging to improve

precision and over-fitting management. The goal of LDA is to find a linear combination of input characteristics that distinguishes two or more input data groups. A quadratic decision surface is used by QDA to distinguish two or more groups of input data. LR is a linear regression-like statistical approach that predicts a result for a binary output variable from an input variable. RC converts the label data to (−1,1) and fixes the regression method problem. As a target class, the greatest value of prediction is admitted. SGDC is a learning algorithm for stochastic gradient descent that finds the decision boundary with a linear hinge loss.

3.6. Evaluation of Malware Detection Results

To measure the classification potential of the proposed ensemble learning model, the performance of the proposed model was evaluated using the Leave-One-Out Cross-Validation (LOOCV) with a 10-fold cross-validation method.

The true labels were compared against the predicted labels and the true positive (TP), true negative (TN), false positive (FP) and false-negative (FN) values were calculated. The recall, precision, accuracy, error rate and F-score values were calculated (we assumed the binary classification problem, where a positive class is labeled by +1 and a negative class is labeled by -1):

False positive rate (FPR) (also specificity):

$$FPR = \frac{\sum_{i=1}^{m}[a(x_i) = +1][y_i = -1]}{\sum_{i=1}^{m}[y_i = -1]} \tag{6}$$

here $[\cdot]$ is the Iverson bracket operator.

True positive rate (TPR) (also sensitivity and recall):

$$TPR = \frac{\sum_{i=1}^{m}[a(x_i) = +1][y_i = +1]}{\sum_{i=1}^{m}[y_i = +1]} \tag{7}$$

False negative rate (FNR):

$$FNR = \frac{\sum_{i=1}^{m}[a(x_i) = -1][y_i = +1]}{\sum_{i=1}^{m}[y_i = +1]} \tag{8}$$

Here, $a(x)$ is the classifier with inputs $X^m = (\ x_1,\ \ldots,\ x_m\ )$, and $(\ y_1,\ \ldots,\ y_m\ )$ are outputs.

Precision is calculated as:

$$Precision = \frac{TPR}{TPR + FPR} \tag{9}$$

To compute F-score, the following equation is used:

$$F - score = 2\frac{Precision \times Recall}{Precision + Recall} \tag{10}$$

The Matthews Correlation Coefficient (MCC) is calculated as:

$$MCC = \frac{TP{\cdot}TN - FP{\cdot}FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{11}$$

The Cohen's Kappa statistic (shortly, kappa) is

$$k = 1 - \frac{1 - p_0}{1 - p_e} \tag{12}$$

where p_0 represents the ratio of correct agreement in the test dataset, and p_e is the ratio of agreement that is expected by random selection.

In this study, performance was calculated using 10-fold cross-validation. According to F1-score, instead of checking the performance of the model with accuracy alone, we selected the best model. The accuracy can be a confusing metric in datasets where a major class imbalance occurs. For a highly imbalanced sample, a model would correctly guess the value of the majority class for all predicted outcomes, and achieve a high performance in classification but making erroneous predictions in the minority and main classes. The F1-score discourages this type of action by computing the metrics for each mark and finding its unweighted average. We also consider area under curve (AUC) as a measure of binary classification consistency, which is known as a balanced metric that can be used even though there are classes of very different sizes in the dataset. Furthermore, the performance of the proposed model on a binary dataset is represented using the confusion matrix.

We used the performance outcomes achieved from the results from each fold of the 10-fold cross-validation for statistical analysis. We adopted the non-parametric Friedman test followed by the post-hoc Nemenyi test to compare the findings and measure their statistical value. Second, both strategies were ranked based on the selected performance measures (we used accuracy, AUC and F1-score). Then, each method's mean ranks were determined. If the difference between the mean ranks of the methods was less than the critical difference obtained from the Nemenyi test, the difference between method outputs was assumed not to be significant.

4. Implementation and Results

4.1. Experimental Settings

The machine learning models were trained on the features acquired from the dataset using Python's Scikit-learn libraries. All experiments were performed on a laptop computer with 64-bit Windows 10 OS with Intel® Core™ i5-8265U CPU @ 1.60 GHz 1.80 GHz with 8 GB RAM (Intel, Santa Clara, CA, USA).

4.2. Results of Machine Learning Methods

The results from using classical machine learning models are summarized in Table 2, while their confusion matrices are summarized in Figure 4. The best results were obtained by the ExtraTrees (ET) model, achieving an accuracy of 98.8%. As can be seen from Table 2 and Figure 3, the ET model generated very good results for the precision, recall, F1 and accuracy of the two classes. This agrees with the low FPR and FNR of 0.8% and 1.4% obtained by the ET model.

Table 2. Summary of results of machine learning models. Acc–Accuracy. Prec–Precision. Rec–Recall. Spec–Specificity. FPR–False Positive Rate. FNR–False Negative Rate. AUC–Area Under Curve. MCC–Matthews Correlation Coefficient. SVM–Support Vector Machine. RBF–Radial Basis Function. LDA–Linear Discriminant Analysis. SGDC–Stochastic Gradient Descent Classifier.

Meta-Learner	Acc	Prec	Rec	Spec	FPR	FNR	F1	AUC	MCC	Kappa
Nearest Neighbors	0.973	0.973	0.973	0.973	0.029	0.025	0.973	0.973	0.946	0.946
Linear SVM	0.954	0.954	0.954	0.954	0.045	0.047	0.954	0.954	0.908	0.908
RBF SVM	0.924	0.924	0.924	0.924	0.012	0.132	0.924	0.928	0.856	0.849
Decision Tree	0.933	0.933	0.933	0.933	0.107	0.032	0.933	0.93	0.866	0.864
Random Forest	0.931	0.931	0.931	0.931	0.08	0.059	0.931	0.93	0.861	0.861
Neural Net	0.977	0.977	0.977	0.977	0.021	0.025	0.977	0.977	0.954	0.954
AdaBoost	0.962	0.962	0.962	0.962	0.041	0.036	0.962	0.961	0.923	0.923
ExtraTrees	0.988	0.988	0.988	0.988	0.008	0.014	0.988	0.989	0.977	0.977
LDA	0.936	0.936	0.936	0.936	0.08	0.05	0.936	0.935	0.871	0.871
Logistic	0.959	0.959	0.959	0.959	0.039	0.043	0.959	0.959	0.917	0.917
Passive	0.933	0.933	0.933	0.933	0.037	0.094	0.933	0.935	0.867	0.866
Ridge	0.936	0.936	0.936	0.936	0.08	0.05	0.936	0.935	0.871	0.871
SGDC	0.958	0.958	0.958	0.958	0.035	0.049	0.958	0.958	0.915	0.915

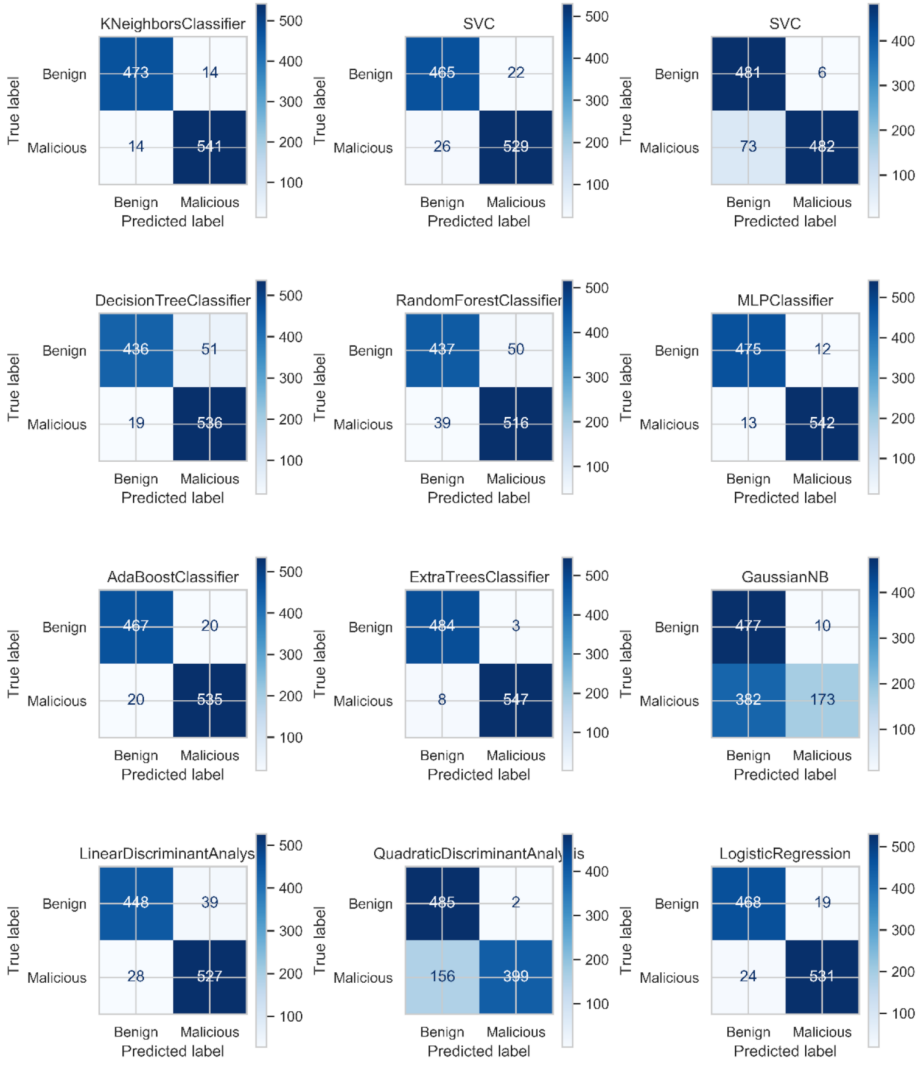

Figure 4. Confusion matrices of machine learning models.

4.3. Results of Neural Network Classifiers

To select the base classifiers, first, we performed an ablation study to find the best representatives of Dense-1, Dense-2 and 1D-CNN models in terms of their performance with respect with different values of hyperparameters. The results are presented in Tables 3–5. Note that in all cases, we used sparse categorical cross-entropy loss function and an Adam optimizer. For the training of Dense-1 and Dense-2 models, we used 100 epochs, while for the training of 1D-CNN models, we used 20 epochs. In all cases, 80% of data were used for training and 20% for testing.

Table 3. Malware detection performance with different number of neurons in hidden layer of Dense-1 model. Best models are shown in bold.

No. of Neurons in 1st Layer	Acc	Prec	Rec	Spec	FPR	FNR	F1	AUC	MCC	Kappa
5	0.956	0.956	0.956	0.956	0.035	0.052	0.956	0.956	0.912	0.912
10	0.962	0.962	0.962	0.962	0.043	0.033	0.962	0.962	0.924	0.924
15	0.965	0.965	0.965	0.965	0.034	0.036	0.965	0.965	0.929	0.929
20	0.971	0.971	0.971	0.971	0.026	0.033	0.971	0.971	0.941	0.941
25	0.977	0.977	0.977	0.977	0.023	0.023	0.977	0.977	0.954	0.954
30	0.977	0.977	0.977	0.977	0.022	0.024	0.977	0.977	0.954	0.954
35	**0.980**	**0.980**	**0.980**	**0.980**	**0.022**	**0.019**	**0.980**	**0.979**	**0.959**	**0.959**
40	0.979	0.979	0.979	0.979	0.023	0.019	0.979	0.979	0.958	0.958

Table 4. Malware detection performance with different number of neurons in hidden layers of Dense-2 model. Best models are shown in bold.

No. of Neurons in 1st Layer	No. of Neurons in 2nd Layer	Acc	Prec	Rec	Spec	FPR	FNR	F1	AUC	MCC	Kappa
5	5	0.954	0.954	0.954	0.954	0.057	0.036	0.954	0.953	0.908	0.908
5	10	0.958	0.958	0.958	0.958	0.046	0.039	0.958	0.958	0.915	0.915
5	15	0.960	0.960	0.960	0.960	0.032	0.047	0.960	0.960	0.919	0.919
5	20	0.964	0.964	0.964	0.964	0.039	0.033	0.964	0.964	0.928	0.928
5	25	0.961	0.961	0.961	0.961	0.042	0.036	0.961	0.961	0.922	0.922
5	30	0.965	0.965	0.965	0.965	0.038	0.033	0.965	0.965	0.929	0.929
5	35	0.963	0.963	0.963	0.963	0.040	0.034	0.963	0.963	0.926	0.926
10	5	0.963	0.963	0.963	0.963	0.042	0.033	0.963	0.963	0.926	0.926
10	10	0.969	0.969	0.969	0.969	0.031	0.032	0.969	0.969	0.937	0.937
10	15	0.965	0.965	0.965	0.965	0.036	0.033	0.965	0.965	0.931	0.931
10	20	0.968	0.968	0.968	0.968	0.030	0.034	0.968	0.968	0.936	0.936
10	25	0.971	0.971	0.971	0.971	0.028	0.030	0.971	0.971	0.941	0.941
10	30	0.971	0.971	0.971	0.971	0.028	0.030	0.971	0.971	0.941	0.941
10	35	0.972	0.972	0.972	0.972	0.024	0.030	0.972	0.973	0.945	0.945
15	5	0.958	0.958	0.958	0.958	0.055	0.029	0.958	0.958	0.917	0.917
15	10	0.972	0.972	0.972	0.972	0.026	0.030	0.972	0.972	0.944	0.944
15	15	0.972	0.972	0.972	0.972	0.034	0.023	0.972	0.972	0.944	0.944
15	20	0.969	0.969	0.969	0.969	0.028	0.034	0.969	0.969	0.937	0.937
15	25	0.971	0.971	0.971	0.971	0.024	0.033	0.971	0.971	0.942	0.942
15	30	0.977	0.977	0.977	0.977	0.026	0.021	0.977	0.977	0.954	0.954
15	**35**	**0.980**	**0.980**	**0.980**	**0.980**	**0.022**	**0.019**	**0.980**	**0.979**	**0.959**	**0.959**
20	5	0.967	0.967	0.967	0.967	0.035	0.032	0.967	0.967	0.933	0.933
20	10	0.976	0.976	0.976	0.976	0.016	0.032	0.976	0.976	0.951	0.951
20	15	0.972	0.972	0.972	0.972	0.028	0.027	0.972	0.972	0.945	0.945
20	20	0.973	0.973	0.973	0.973	0.027	0.027	0.973	0.973	0.946	0.946
20	**25**	**0.980**	**0.980**	**0.980**	**0.980**	**0.023**	**0.018**	**0.980**	**0.979**	**0.959**	**0.959**
20	30	0.978	0.978	0.978	0.978	0.023	0.022	0.978	0.978	0.955	0.955
20	35	0.978	0.978	0.978	0.978	0.022	0.022	0.978	0.978	0.956	0.956
25	5	0.970	0.970	0.970	0.970	0.030	0.030	0.970	0.970	0.940	0.940
25	10	0.974	0.974	0.974	0.974	0.024	0.027	0.974	0.974	0.949	0.949
25	15	0.974	0.974	0.974	0.974	0.022	0.030	0.974	0.974	0.947	0.947
25	20	0.975	0.975	0.975	0.975	0.028	0.022	0.975	0.975	0.950	0.950
25	**25**	**0.980**	**0.980**	**0.980**	**0.980**	**0.023**	**0.017**	**0.980**	**0.980**	**0.960**	**0.960**
25	30	0.980	0.980	0.980	0.980	0.023	0.018	0.980	0.979	0.959	0.959
25	35	0.980	0.980	0.980	0.980	0.024	0.017	0.980	0.979	0.959	0.959
30	5	0.976	0.976	0.976	0.976	0.031	0.017	0.976	0.976	0.953	0.952
30	10	0.978	0.978	0.978	0.978	0.015	0.029	0.978	0.978	0.955	0.955
30	15	0.974	0.974	0.974	0.974	0.024	0.027	0.974	0.974	0.949	0.949
30	20	0.979	0.979	0.979	0.979	0.024	0.018	0.979	0.979	0.958	0.958
30	25	0.980	0.980	0.980	0.980	0.024	0.017	0.980	0.979	0.959	0.959
30	**30**	**0.981**	**0.981**	**0.981**	**0.981**	**0.022**	**0.017**	**0.981**	**0.981**	**0.962**	**0.962**
30	**35**	**0.983**	**0.983**	**0.983**	**0.983**	**0.013**	**0.019**	**0.983**	**0.984**	**0.967**	**0.967**

Table 5. Malware detection performance with different number of filters in convolutional layers and neurons in the final fully connected layer of 1D-CNN model. Best models are shown in bold.

No. of Filters	No. of Neurons	Acc	Prec	Rec	Spec	FPR	FNR	F1	AUC	MCC	Kappa
32	10	0.957	0.957	0.957	0.957	0.045	0.041	0.957	0.957	0.914	0.914
32	15	0.960	0.960	0.960	0.960	0.055	0.027	0.960	0.959	0.919	0.919
32	20	0.964	0.964	0.964	0.964	0.036	0.036	0.964	0.964	0.927	0.927
32	25	0.960	0.960	0.960	0.960	0.053	0.028	0.960	0.960	0.921	0.920
32	30	0.961	0.961	0.961	0.961	0.058	0.022	0.961	0.960	0.922	0.922
32	35	0.964	0.964	0.964	0.964	0.049	0.026	0.964	0.963	0.927	0.927
32	40	0.966	0.966	0.966	0.966	0.039	0.029	0.966	0.966	0.932	0.932
32	45	0.967	0.967	0.967	0.967	0.042	0.026	0.967	0.966	0.933	0.933

Table 5. *Cont.*

No. of Filters	No. of Neurons	Acc	Prec	Rec	Spec	FPR	FNR	F1	AUC	MCC	Kappa
48	10	0.967	0.967	0.967	0.967	0.032	0.033	0.967	0.967	0.935	0.935
48	15	0.965	0.965	0.965	0.965	0.039	0.032	0.965	0.965	0.929	0.929
48	20	0.972	0.972	0.972	0.972	0.020	0.035	0.972	0.972	0.944	0.944
48	25	0.962	0.962	0.962	0.962	0.032	0.043	0.962	0.963	0.924	0.924
48	30	0.969	0.969	0.969	0.969	0.016	0.045	0.969	0.969	0.938	0.937
48	35	0.970	0.970	0.970	0.970	0.018	0.041	0.970	0.971	0.940	0.940
48	40	0.972	0.972	0.972	0.972	0.019	0.036	0.972	0.972	0.944	0.944
48	45	0.971	0.971	0.971	0.971	0.026	0.033	0.971	0.971	0.941	0.941
64	10	0.961	0.961	0.961	0.961	0.053	0.027	0.961	0.960	0.922	0.922
64	15	0.965	0.965	0.965	0.965	0.020	0.047	0.965	0.966	0.931	0.931
64	**20**	**0.980**	**0.980**	**0.980**	**0.980**	**0.019**	**0.022**	**0.980**	**0.980**	**0.959**	**0.959**
64	25	0.972	0.972	0.972	0.972	0.040	0.017	0.972	0.971	0.944	0.943
64	30	0.974	0.974	0.974	0.974	0.016	0.035	0.974	0.974	0.948	0.947
64	35	0.969	0.969	0.969	0.969	0.046	0.017	0.969	0.969	0.939	0.938
64	40	0.979	0.979	0.979	0.979	0.022	0.021	0.979	0.979	0.958	0.958
64	45	0.974	0.974	0.974	0.974	0.023	0.029	0.974	0.974	0.947	0.947
128	10	0.978	0.978	0.978	0.978	0.013	0.029	0.978	0.979	0.957	0.956
128	15	0.980	0.980	0.980	0.980	0.022	0.018	0.980	0.980	0.960	0.960
128	20	0.975	0.975	0.975	0.975	0.011	0.038	0.975	0.976	0.950	0.950
128	25	0.980	0.980	0.980	0.980	0.022	0.019	0.980	0.979	0.959	0.959
128	30	0.979	0.979	0.979	0.979	0.020	0.022	0.979	0.979	0.958	0.958
128	**35**	**0.985**	**0.985**	**0.985**	**0.985**	**0.018**	**0.013**	**0.985**	**0.985**	**0.969**	**0.969**
128	40	0.983	0.983	0.983	0.983	0.019	0.015	0.983	0.983	0.967	0.967
128	45	0.979	0.979	0.979	0.979	0.013	0.028	0.979	0.979	0.958	0.958

4.4. Results of Ensemble Learning

Based on the ablation study, we selected one Dense-1 (with 35 neurons) model, two Dense-2 (with (40,40) and (40,50) neurons) models and two 1D-CNN (with (25,25) and (30,35) neurons) models as base learners based on their kappa and F1-score performance. We performed classification with several different meta-learner classification algorithms. For KNN, the number of nearest neighbors was set to 3. For linear SVM, C was set to 0.025. For RBF SVM, the C parameter (which performs regularization by applying a penalty to reduce overfitting) was set to 1, and gamma was set to 2. For DT and RF, the max depth was set to 5. In all cases, 10-fold cross-validation was used, where each cross-validation fold was made by randomly selecting 80% of samples, and the remaining 20% were used for testing. The results are presented in Table 6.

Table 6. Ensemble learning results with different meta-learners: mean values from 10-fold cross-validation. Best values are shown in bold.

Meta-Learner	Acc	Prec	Rec	Spec	FPR	FNR	F1	AUC	MCC	Kappa
Nearest Neighbors	0.984	0.984	0.984	0.984	0.014	0.018	0.984	0.984	0.967	0.967
Linear SVM	0.974	0.974	0.974	0.974	0.029	0.023	0.974	0.974	0.948	0.948
RBF SVM	0.979	0.979	0.979	0.979	0.021	0.022	0.979	0.979	0.958	0.958
Decision Tree	0.987	0.987	0.987	0.987	0.002	0.023	0.987	0.987	0.973	0.973
Random Forest	0.991	0.991	0.991	0.991	0.008	0.009	0.991	0.991	0.983	0.983
Neural Net	0.974	0.974	0.974	0.974	0.029	0.023	0.974	0.974	0.948	0.948
AdaBoost	0.997	0.997	0.997	0.997	0.006	0	0.997	0.997	0.994	0.994
ExtraTrees	**0.999**	**0.999**	**0.998**	**1.000**	**0.000**	**0.002**	**0.999**	**0.999**	**0.999**	**0.999**
Naive Bayes	0.968	0.968	0.968	0.968	0.027	0.036	0.968	0.969	0.937	0.936
LDA	0.975	0.975	0.975	0.975	0.027	0.023	0.975	0.975	0.95	0.95
QDA	0.974	0.974	0.974	0.974	0.029	0.023	0.974	0.974	0.948	0.948

Table 6. *Cont.*

Meta-Learner	Acc	Prec	Rec	Spec	FPR	FNR	F1	AUC	MCC	Kappa
Logistic	0.973	0.973	0.973	0.973	0.031	0.023	0.973	0.973	0.946	0.946
Ridge	0.971	0.971	0.971	0.971	0.049	0.011	0.971	0.97	0.943	0.942
SGDC	0.975	0.975	0.975	0.975	0.027	0.023	0.975	0.975	0.95	0.95

The average performance results are visualized in Figures 5–7, whereas the results from the 10-fold cross-validation are shown as boxplots in Figures 8–10. The results demonstrate that the ExtraTrees meta-learner achieved the highest performance in terms of accuracy, AUC and F1-score measures.

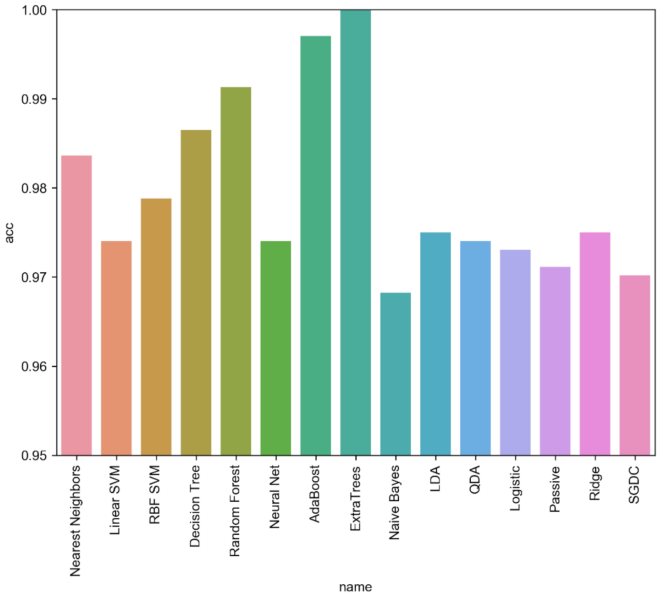

Figure 5. Malware detection performance of deep learning ensemble model by final stage meta-learner classifier: accuracy.

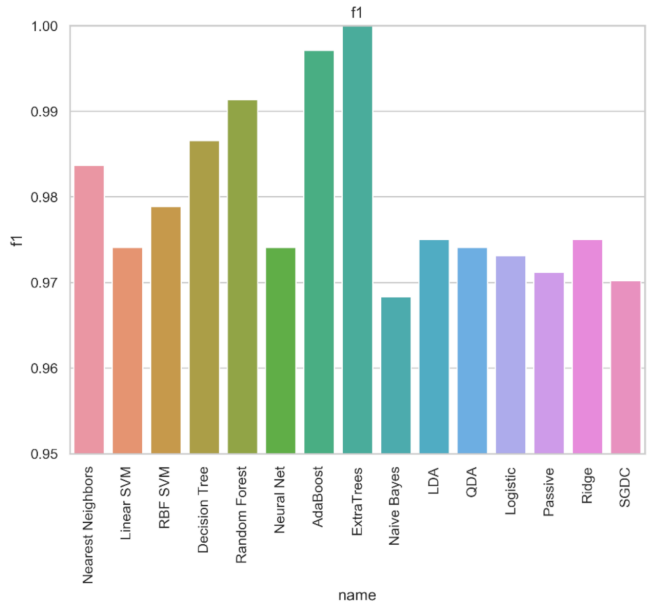

Figure 6. Malware detection performance of deep learning ensemble model by final stage meta-learner classifier: F1-score.

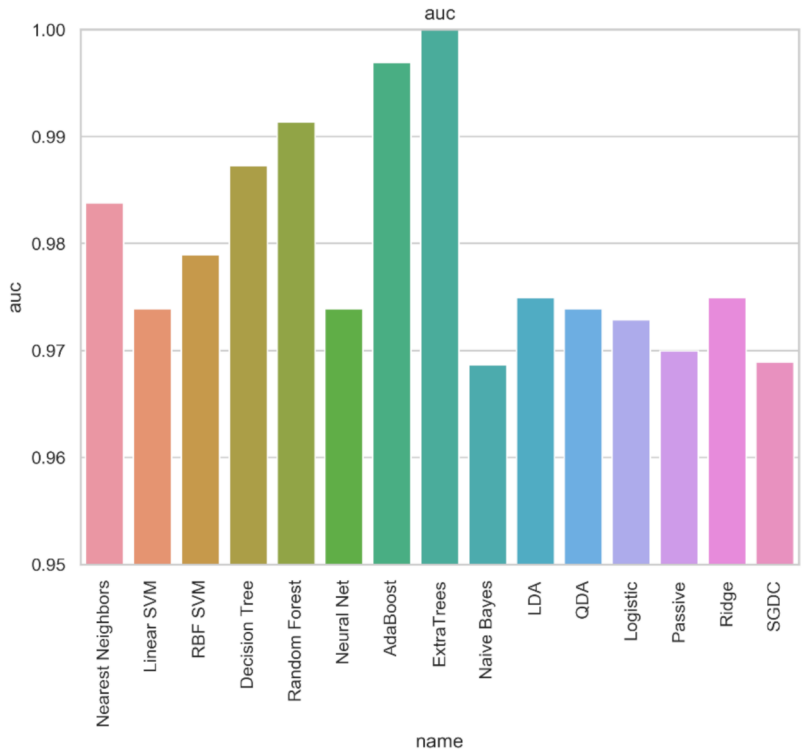

Figure 7. Malware detection performance of deep learning ensemble model by final stage meta-learner classifier: AUC.

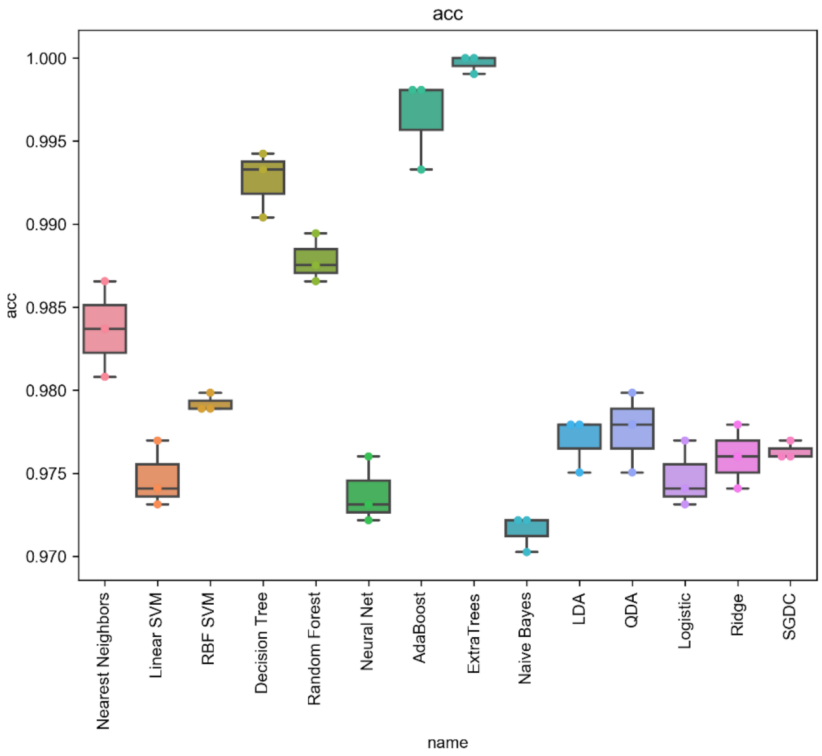

Figure 8. Malware detection performance of deep learning ensemble model by final stage meta-learner classifier: accuracy.

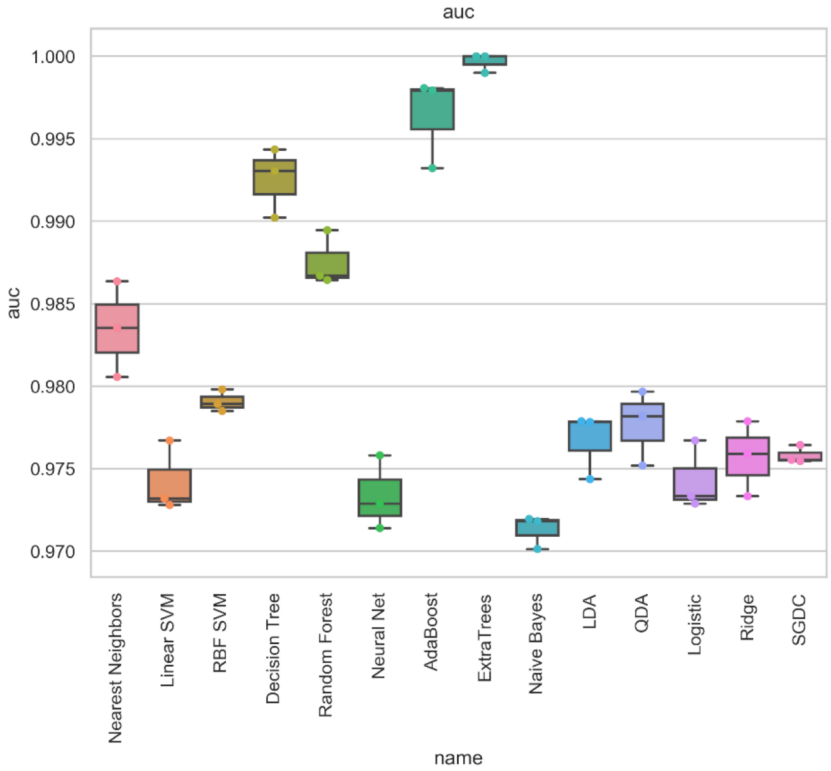

Figure 9. Malware detection performance of deep learning ensemble model by final stage meta-learner classifier: area under curve.

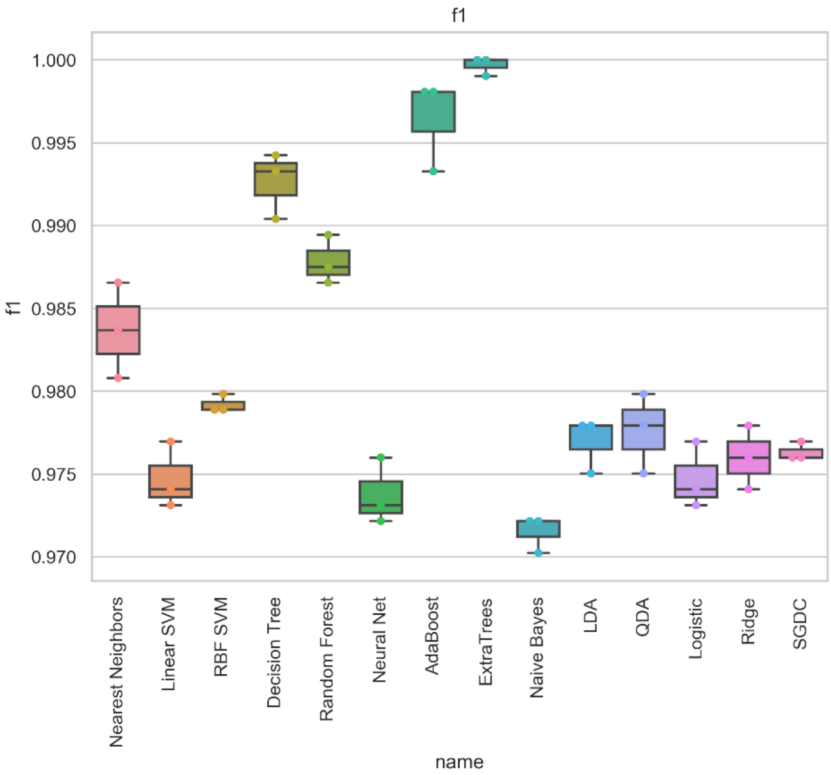

Figure 10. Malware detection performance of deep learning ensemble model by final stage meta-learner classifier: F1-score.

Finally, we present the confusion matrix of the best ensemble model (with the ET classifier as the meta-learner) in Figure 11.

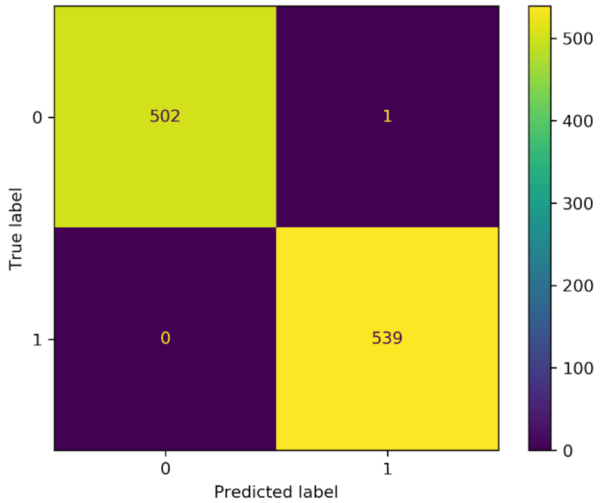

Figure 11. Confusion matrix of the best ensemble model (with the ET classifier as meta-learner).

4.5. Statistical Analysis

To perform the statistical analysis of the experimental results, we adopted the Friedman test and the Nemenyi test. The results are presented as critical difference (CD) diagrams in Figures 12–14. If the difference between the mean ranks of the meta-learners is smaller than the CD, then it is not statistically significant. The results of the Nemenyi test again show that the ExtraTrees meta-learner allows us to achieve the best performance; however, the performance of AdaBoost and Decision Tree meta-learners is not significantly different.

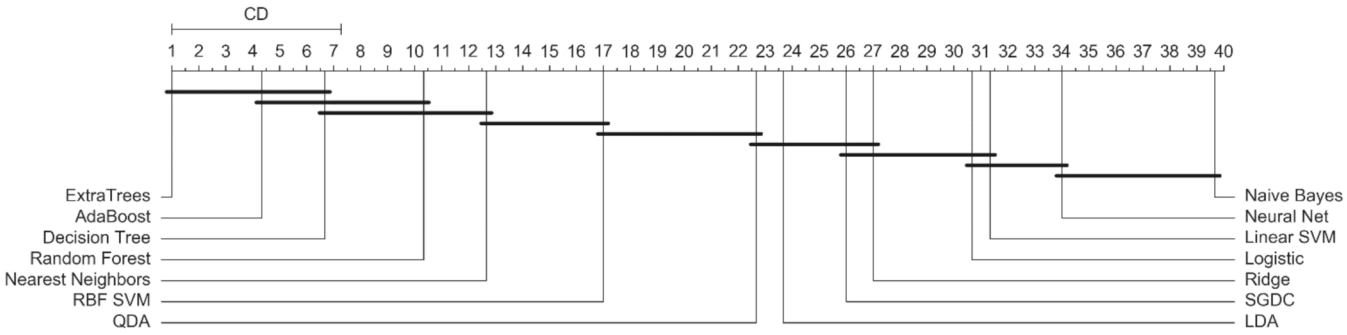

Figure 12. Comparison of mean ranks of meta-learners based on their accuracy performance: results of Nemenyi test.

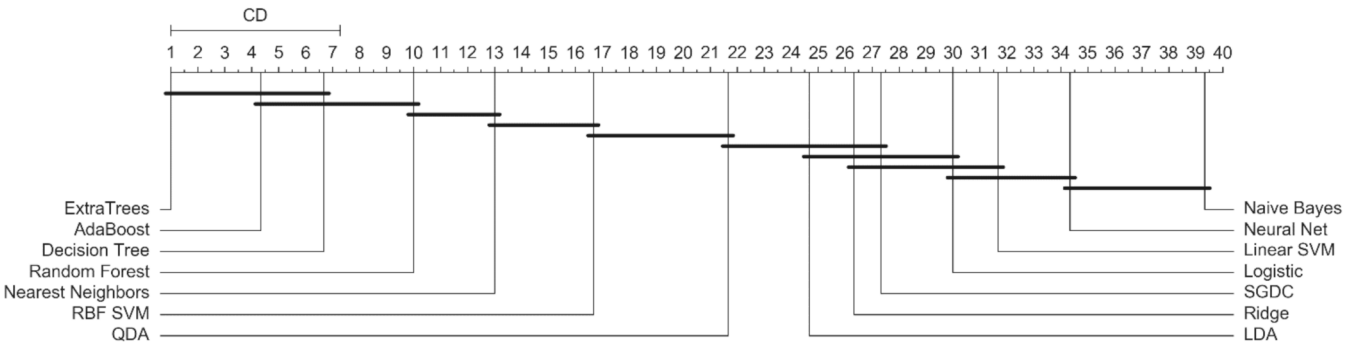

Figure 13. Comparison of mean ranks of meta-learners based on their AUC performance: results of Nemenyi test.

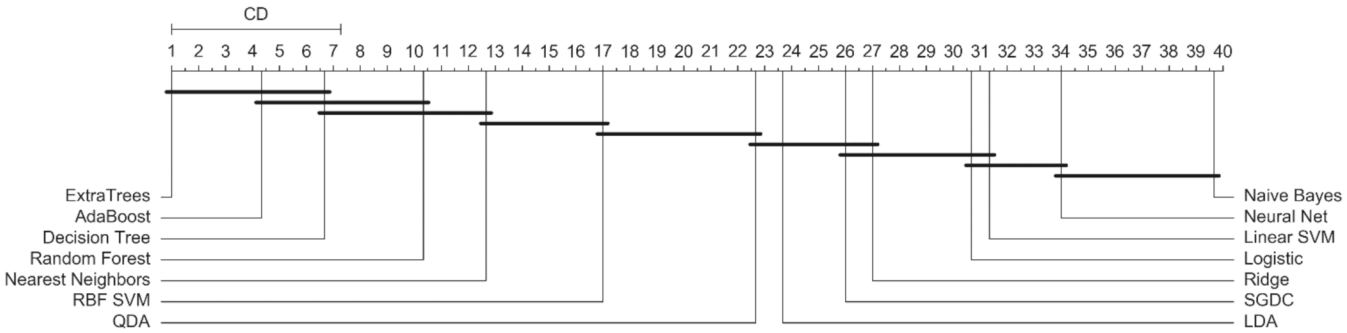

Figure 14. Comparison of mean ranks of meta-learners based on their F1-score performance: results of Nemenyi test.

4.6. Ablation Study of the Ensemble

We also conducted the ablation study to evaluate the contribution of the individual parts in the proposed ensemble classification base framework for malware recognition. We compared and analyzed the impact of the ensemble size of the classification results. We analyzed the following ensembles, consisting of a smaller number (4) of neural networks models:

1. Case ENSEMBLE1: two Dense-2 (with (40,40) and (40,50) neurons) models and two 1D-CNN (with (25,25) and (30,35) neurons) models;
2. Case ENSEMBLE2: one Dense-1 (with 35 neurons) model, one Dense-2 (with (40,40) neurons) model and two 1D-CNN ((25,25), and (30,35) neurons) models;
3. Case ENSEMBLE3: one Dense-1 (with 35 neurons) model, two Dense-2 (with (40,40) and (40,50) neurons) models and one 1D-CNN (with (30;35) neurons) model.

The results are summarized and compared in Table 7. In all cases, the ExtraTrees Classifier was used as a meta-learner. The Full Model here corresponds to the five-model ensemble with PCA scaling of data. The results show that the best performance was achieved by the full five-model ensemble with data scaling using PCA and ExtraTrees as the meta-learner.

Table 7. Comparison of ensemble models. Best values are shown in bold.

Case	Acc	Prec	Rec	Spec	FPR	FNR	F1	AUC	MCC	Kappa
ENSEMBLE1	0.989	0.988	0.987	0.979	0.011	0.012	0.989	0.989	0.968	0.968
ENSEMBLE2	0.985	0.983	0.985	0.983	0.017	0.014	0.984	0.984	0.967	0.967
ENSEMBLE3	0.985	0.984	0.984	0.985	0.013	0.016	0.986	0.988	0.971	0.970
Full Model	**0.999**	**0.999**	**0.998**	**1.000**	**0.000**	**0.002**	**0.999**	**0.999**	**0.999**	**0.999**

4.7. Comparison with Related Work.

Finally, we compare our results with some of the related work on classifying benign and malware files in Table 8 and explain in more detail below. Note that the methods working on different malware datasets were compared. Alzaylaee et al. [54] explored 2-, 3- and 4-layer fully connected neural networks on a dataset of 31,125 Android apps, with 420 static and dynamic features, while comparing the results to machine learning classifiers. The best results were achieved with a three-layer network with 200 neurons in each layer. Bakour and Ünver [55] suggested a visualization-based approach that converted software characteristics into grayscale images and then applied local and global image features as voters in an ensemble voting classifier. Cai et al. [56] used information gain for feature selection and weight mapping functions derived by machine learning methods, which were optimized by the differential evolution algorithm. Chen et al. [57] used an attention network architecture based on CNN to classify apps based on their Application Programming Interface (API) call sequences. Fang et al. [58] used a DeepDetectNet deep learning model for static PE malware detection model, and an adversarial generation

network RLAttackNet based on reinforcement learning, which was trained to bypass DeepDetectNet. The generated adversarial samples were used to retrain DeepDetectNet, which allowed the improvement of malware recognition accuracy.

Imtiaz et al. [59] proposed a deep multi-layer fully connected Artificial Neural Network (ANN) that has an input layer, few hidden layers and an output layer. The approach has been validated with the CICInvesAndMal2019 dataset of Android malware. Jeon and Moon [60] proposed a convolutional recurrent neural network (CRNN), which uses the opcode sequences of software as input. The front-end CNN performs opcode compression, and the back end dynamic recurrent neural network (DRNN) detects malware from the compressed sequence.

Jha et al. [61] proposed using RNN with feature vectors obtained by skip-grams of the Word2Vec embedding model for malware recognition. Namavar Jahromi et al. [62] proposed a modified Two-hidden-layered Extreme Learning Machine (TELM), which was tested on Ransomware, Windows, Internet of Things (IoT) and other malware datasets.

Narayanan and Davuluru [63] suggested using CNNs and Long Short-Term Memory (LSTM) networks for feature extraction and SVM or LR for the classification of malware based on their machine language opcodes. The approach was validated on Microsoft's Malware Classification Challenge (BIG 2015) dataset with nine malware classes. Song et al. [64] proposed a JavaScript malware detection based on the Bidirectional LSTM neural network. Wang et al. [65] suggested CrowdNet, a radial basis function network, as a malware predictor. Yen and Sun [66] extracted instruction code and applied hashing to extract features. Then, the features were transformed into images and used to train a CNN.

Table 8. Comparison with other known deep learning approaches for malware recognition. n/a—data were not provided.

Reference	Benign	Malware	Acc. (%)	Prec. (%)	Recall (%)	F-Score (%)
Alzaylaee et al. [54]	19,620	11,505	98.5	98.09	99.56	98.82
Bakour and Ünver [55]	-	4850	98.14	n/a	n/a	n/a
Cai et al. [56]	3000	3000	96.92	96.75	97.23	96.99
Chen et al. [57]	4596	4596	97.23	98.69	98.69	98.69
Fang et al. [58]	749	726	n/a	n/a	98.07	n/a
Imtiaz et al. [59]	5065	426	93.4	93.5	93.4	93.2
Jeon and Moon [60]	1000	1000	96	n/a	95	n/a
Jha et al. [61]	20,000	20,000	91.91	n/a	n/a	91.76
Namavar Jahromi et al. [62]	-	18,831	99.03	n/a	n/a	n/a
Narayanan and Davuluru [63]	-	7826	99.8	n/a	n/a	n/a
Song et al. [64]	30,487	29,893	97.71	n/a	n/a	98.29
Wang et al. [65]	6375	6375	n/a	94	n/a	n/a
Yen and Sun [66]	720	720	92.67	n/a	n/a	n/a
This paper	2488	2722	99.99	99.99	99.98	99.99

5. Conclusions

There is an increase in demand for smart methods that detect new malware variants, because the existing methods are time-consuming and vulnerable to many errors. This paper analyzed various machine learning algorithms and models of neural networks, which are smart approaches that can be used for malware detection. With neural networks used as base learners, we proposed an ensemble learning-based architecture and explored 14 machine learning algorithms as meta-learners. As baseline models, we used machine learning algorithms for comparison. We conducted our experiments on a dataset that included malware and benign files from Windows Portable Executables (PE).

In this paper, we analyzed and experimentally validated the use of ensemble learning to combine the malware prediction results given by different machine learning and deep learning models. The aim of this practice is to improve the recognition of Windows PE malware. With ensemble methods, it is not required to select any specific machine learning model. Instead, the prediction capability of each combination of the machine learning models is aggregated to create a learning procedure that achieves the best malware

detection performance. We explored our proposed ensemble classification framework with lightweight fully connected and convolutional neural network architectures, and combined deep learning and machine learning techniques to learn effective and efficient malware detection models. We conducted extensive experiments on various lightweight deep learning architectures and machine learning models within the framework of ensemble learning under the same conditions for a fair comparison.

The results achieved show that the malware detection ability of ensemble stacking exceeds the ability of other machine-learning methods, including neural networks. We showed that the ensemble learning framework based on lightweight deep models could successfully tackle the problem of malware detection. The results obtained indicate that ensemble learning methods can be implemented and used as intelligent techniques for the identification of malware. The classification system with the Extra Trees algorithm as a meta-learner and an ensemble of dense ANN and 1-D CNN models obtained the best accuracy value for the classification procedure, outperforming other machine learning classification methods. Our proposed framework can lead to highly accurate malware detection models that are adapted for real-world Windows PE malware.

The application of explanatory artificial intelligence (XAI) [67] strategies to interpret the outcomes of deep learning models for malware detection will be carried out in the future to provide useful information for malware analysis researchers. We also intend to explore ensemble learning architectures and run further tests with larger databases of malware. We strive to improve the classification ability and accuracy of the ensemble learning model by refining the model architecture and validating it for multiple malware datasets in future work.

Author Contributions: Conceptualization and methodology, R.D. and A.V.; software, R.D.; validation, R.D., A.V. and J.T.; formal analysis, R.D. and J.T.; investigation, R.D., A.V. and Š.G.; resources, A.V. and J.T.; writing—original draft preparation, R.D., A.V., J.T. and Š.G.; writing—review and editing, R.D. and A.V.; visualization, R.D. and J.T.; supervision, A.V. All authors have read and agreed to the published version of the manuscript.

Funding: This paper was supported in part by European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 830892, project "Strategic programs for advanced research and technology in Europe" (SPARTA).

Data Availability Statement: The dataset is available from https://github.com/urwithajit9/ClaMP.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Lallie, H.S.; Shepherd, L.A.; Nurse, J.R.C.; Erola, A.; Epiphaniou, G.; Maple, C.; Bellekens, X.J.A. Cyber Security in the Age of COVID-19: A Timeline and Analysis of Cyber-Crime and Cyber-Attacks during the Pandemic. *arXiv* **2020**, arXiv:2006.11929.
2. Anderson, R.; Barton, C.; Böhme, R.; Clayton, R.; Van Eeten, M.J.G.; Levi, M.; Moore, T.; Savage, S. Measuring the Cost of Cybercrime. In *The Economics of Information Security and Privacy*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 265–300.
3. Bissell, K.; la Salle, R.; Dal, C.P. The 2020 Cyber Security Report. 2020. Available online: https://pages.checkpoint.com/cyber-security-report-2020 (accessed on 4 May 2020).
4. Chebyshev, V. Mobile Malware Evolution. 2018. Available online: https://securelist.com/mobile-malware-evolution-2018/89689/ (accessed on 18 June 2020).
5. Kaspersky Lab. The Great Bank Robbery. 2015. Available online: https://www.kaspersky.com/about/press-releases/2015_the-great-bank-robbery-carbanak-cybergang-steals--1bn-from-100-financial-institutions-worldwide (accessed on 17 February 2015).
6. Kingsoft. 2015–2016 Internet Security Research Report in China. 2016. Available online: https://cn.cmcm.com/news/media/2016-01-14/60.html (accessed on 14 January 2016).
7. Bissell, K.; la Salle, R.M.; Dal, C.P. The Cost of Cybercrime—Ninth Annual Cost of Cybercrime Study. Technical Report, Ponemon Institute LLC, Accenture. 2019. Available online: https://www.accenture.com/_acnmedia/pdf-96/accenture-2019-cost-of-cybercrime-study-final.pdf (accessed on 3 March 2020).
8. Cybersecurity Ventures. Cybercrime Damages Will Cost the World $6 Trillion Annually by 2021. 2017. Available online: https://cybersecurityventures.com/cybercrime-damages-6-trillion-by-2021/ (accessed on 21 December 2020).

9. Williams, C.M.; Chaturvedi, R.; Chakravarthy, K. Cybersecurity Risks in a Pandemic. *J. Med. Internet Res.* **2020**, *22*, e23692. [CrossRef]
10. Hakak, S.; Khan, W.Z.; Imran, M.; Choo, K.-K.R.; Shoaib, M. Have You Been a Victim of COVID-19-Related Cyber Incidents? Survey, Taxonomy, and Mitigation Strategies. *IEEE Access* **2020**, *8*, 124134–124144. [CrossRef]
11. Seh, A.H.; Zarour, M.; Alenezi, M.; Sarkar, A.K.; Agrawal, A.; Kumar, R.; Khan, R.A. Healthcare Data Breaches: Insights and Implications. *Healthcare* **2020**, *8*, 133. [CrossRef]
12. Pierazzi, F.; Mezzour, G.; Han, Q.; Colajanni, M.; Subrahmanian, V.S. A Data-driven Characterization of Modern Android Spyware. *ACM Trans. Manag. Inf. Syst.* **2020**, *11*, 1–38. [CrossRef]
13. Odusami, M.; Abayomi-Alli, O.; Misra, S.; Shobayo, O.; Damasevicius, R.; Maskeliunas, R. Android Malware Detection: A Survey. In *Communications in Computer and Information Science*; Springer International Publishing: Cham, Switzerland, 2018; Volume 942, pp. 255–266. [CrossRef]
14. Subairu, S.O.; Alhassan, J.; Misra, S.; Abayomi-Alli, O.; Ahuja, R.; Damasevicius, R.; Maskeliunas, R. An Experimental Approach to Unravel Effects of Malware on System Network Interface. In *Lecture Notes in Electrical Engineering*; Springer International Publishing: Cham, Switzerland, 2019; Volume 612, pp. 225–235. [CrossRef]
15. Alsoghyer, S.; Almomani, I. Ransomware Detection System for Android Applications. *Electronics* **2019**, *8*, 868. [CrossRef]
16. Hindy, H.; Atkinson, R.; Tachtatzis, C.; Colin, J.-N.; Bayne, E.; Bellekens, X. Utilising Deep Learning Techniques for Effective Zero-Day Attack Detection. *Electronics* **2020**, *9*, 1684. [CrossRef]
17. Martín, I.; Hernández, J.A.; Santos, S.D.L. Machine-Learning based analysis and classification of Android malware signatures. *Futur. Gener. Comput. Syst.* **2019**, *97*, 295–305. [CrossRef]
18. Aslan, O.; Samet, R. A Comprehensive Review on Malware Detection Approaches. *IEEE Access* **2020**, *8*, 6249–6271. [CrossRef]
19. Souri, A.; Hosseini, R. A state-of-the-art survey of malware detection approaches using data mining techniques. *Hum. Cent. Comput. Inf. Sci.* **2018**, *8*, 3. [CrossRef]
20. Ye, Y.; Li, T.; Adjeroh, D.; Iyengar, S.S. A Survey on Malware Detection Using Data Mining Techniques. *ACM Comput. Surv.* **2017**, *50*, 1–40. [CrossRef]
21. Ucci, D.; Aniello, L.; Baldoni, R. Survey of machine learning techniques for malware analysis. *Comput. Secur.* **2019**, *81*, 123–147. [CrossRef]
22. Liu, K.; Xu, S.; Xu, G.; Zhang, M.; Sun, D.; Liu, H. A Review of Android Malware Detection Approaches Based on Machine Learning. *IEEE Access* **2020**, *8*, 124579–124607. [CrossRef]
23. Truong, T.C.; Diep, Q.B.; Zelinka, I. Artificial Intelligence in the Cyber Domain: Offense and Defense. *Symmetry* **2020**, *12*, 410. [CrossRef]
24. Ngo, Q.-D.; Nguyen, H.-T.; Le, V.-H.; Nguyen, D.-H. A survey of IoT malware and detection methods based on static features. *ICT Express* **2020**, *6*, 280–286. [CrossRef]
25. Egele, M.; Scholte, T.; Kirda, E.; Kruegel, C. A survey on automated dynamic malware-analysis techniques and tools. *ACM Comput. Surv.* **2012**, *44*, 1–42. [CrossRef]
26. Ye, Y.; Wang, D.; Li, T.; Ye, D.; Jiang, Q. An intelligent PE-malware detection system based on association mining. *J. Comput. Virol.* **2008**, *4*, 323–334. [CrossRef]
27. Nisa, M.; Shah, J.H.; Kanwal, S.; Raza, M.; Khan, M.A.; Damaševičius, R.; Blažauskas, T. Hybrid Malware Classification Method Using Segmentation-Based Fractal Texture Analysis and Deep Convolution Neural Network Features. *Appl. Sci.* **2020**, *10*, 4966. [CrossRef]
28. Yong, B.; Wei, W.; Li, K.; Shen, J.; Zhou, Q.; Wozniak, M.; Połap, D.; Damaševičius, R. Ensemble machine learning approaches for webshell detection in Internet of things environments. *Trans. Emerg. Telecommun. Technol.* **2020**. [CrossRef]
29. Wei, W.; Woźniak, M.; Damaševičius, R.; Fan, X.; Li, Y. Algorithm research of known-plaintext attack on double random phase mask based on WSNs. *J. Internet Technol.* **2019**, *20*, 39–48. [CrossRef]
30. Berman, D.S.; Buczak, A.L.; Chavis, J.S.; Corbett, C.L. A Survey of Deep Learning Methods for Cyber Security. *Information* **2019**, *10*, 122. [CrossRef]
31. Ren, Z.; Wu, H.; Ning, Q.; Hussain, I.; Chen, B. End-to-end malware detection for android IoT devices using deep learning. *Ad Hoc Netw.* **2020**, *101*, 102098. [CrossRef]
32. Yuxin, D.; Siyi, Z. Malware detection based on deep learning algorithm. *Neural Comput. Appl.* **2017**, *31*, 461–472. [CrossRef]
33. Pei, X.; Yu, L.; Tian, S. AMalNet: A deep learning framework based on graph convolutional networks for malware detection. *Comput. Secur.* **2020**, *93*, 101792. [CrossRef]
34. Čeponis, D.; Goranin, N. Investigation of Dual-Flow Deep Learning Models LSTM-FCN and GRU-FCN Efficiency against Single-Flow CNN Models for the Host-Based Intrusion and Malware Detection Task on Univariate Times Series Data. *Appl. Sci.* **2020**, *10*, 2373. [CrossRef]
35. Huang, X.; Ma, L.; Yang, W.; Zhong, Y. A Method for Windows Malware Detection Based on Deep Learning. *J. Signal Process. Syst.* **2020**, 1–9. [CrossRef]
36. Martins, N.; Cruz, J.M.; Cruz, T.; Abreu, P.H. Adversarial Machine Learning Applied to Intrusion and Malware Scenarios: A Systematic Review. *IEEE Access* **2020**, *8*, 35403–35419. [CrossRef]
37. Zador, A.M. A critique of pure learning and what artificial neural networks can learn from animal brains. *Nat. Commun.* **2019**, *10*, 1–7. [CrossRef] [PubMed]

38. Idrees, F.; Rajarajan, M.; Conti, M.; Chen, T.M.; Rahulamathavan, Y. PIndroid: A novel Android malware detection system using ensemble learning methods. *Comput. Secur.* **2017**, *68*, 36–46. [CrossRef]
39. Feng, P.; Ma, J.; Sun, C.; Xu, X.; Ma, Y. A Novel Dynamic Android Malware Detection System with Ensemble Learning. *IEEE Access* **2018**, *6*, 30996–31011. [CrossRef]
40. Wang, W.; Li, Y.; Wang, X.; Liu, J.; Zhang, X. Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers. *Futur. Gener. Comput. Syst.* **2018**, *78*, 987–994. [CrossRef]
41. Yan, J.; Qi, Y.; Rao, Q. Detecting Malware with an Ensemble Method Based on Deep Neural Networks. *Secur. Commun. Netw.* **2018**, *2018*, 1–16. [CrossRef]
42. Gupta, D.; Rani, R. Improving malware detection using big data and ensemble learning. *Comput. Electr. Eng.* **2020**, *86*, 106729. [CrossRef]
43. Sagi, O.; Rokach, L. Ensemble learning: A survey. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **2018**, *8*, e1249. [CrossRef]
44. Basu, I. Malware detection based on source data using data mining: A survey. *Am. J. Adv. Comput.* **2016**, *3*, 18–37.
45. 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.
46. Ruder, S. An overview of gradient descent optimization algorithms. *arXiv* **2016**, arXiv:1609.04747.
47. Duchi, J.; Hazan, E.; Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.* **2011**, *12*, 2121–2159.
48. Tieleman, T.; Hinton, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *Coursera Neural Netw. Mach. Learn.* **2012**, *4*, 26–31.
49. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. In Proceedings of the International Conference on Learning Representation (ICLR), San Diego, CA, USA, 5–8 May 2015.
50. Ragab, M.; Abdulkadir, S.; Aziz, N.; Al-Tashi, Q.; Alyousifi, Y.; Alhussian, H.; Alqushaibi, A. A Novel One-Dimensional CNN with Exponential Adaptive Gradients for Air Pollution Index Prediction. *Sustainability* **2020**, *12*, 10090. [CrossRef]
51. Luo, L.; Xiong, Y.; Liu, Y.; Sun, X. Adaptive gradient methods with dynamic bound of learning rate. *arXiv* **2019**, arXiv:1902.09843.
52. Van der Laan, M.J.; Polley, E.C.; Hubbard, A.E. Super Learner. *Stat. Appl. Genet. Mol. Biol.* **2007**, *6*. [CrossRef] [PubMed]
53. Geurts, P.; Ernst, D.; Wehenkel, L. Extremely randomized trees. *Mach. Learn.* **2006**, *63*, 3–42. [CrossRef]
54. Alzaylaee, M.K.; Yerima, S.Y.; Sezer, S. DL-Droid: Deep learning based android malware detection using real devices. *Comput. Secur.* **2020**, *89*, 101663. [CrossRef]
55. Bakour, K.; Ünver, H.M. VisDroid: Android malware classification based on local and global image features, bag of visual words and machine learning techniques. *Neural Comput. Appl.* **2020**, *2020*, 1–21. [CrossRef]
56. Cai, L.; Li, Y.; Xiong, Z. JOWMDroid: Android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters. *Comput. Secur.* **2021**, *100*, 102086. [CrossRef]
57. Chen, J.; Guo, S.; Ma, X.; Li, H.; Guo, J.; Chen, M.; Pan, Z. SLAM: A Malware Detection Method Based on Sliding Local Attention Mechanism. *Secur. Commun. Netw.* **2020**, *2020*, 1–11. [CrossRef]
58. Fang, Y.; Zeng, Y.; Li, B.; Liu, L.; Zhang, L. DeepDetectNet vs RLAttackNet: An adversarial method to improve deep learning-based static malware detection model. *PLoS ONE* **2020**, *15*, e0231626. [CrossRef]
59. Imtiaz, S.I.; Rehman, S.U.; Javed, A.R.; Jalil, Z.; Liu, X.; Alnumay, W.S. DeepAMD: Detection and identification of Android malware using high-efficient Deep Artificial Neural Network. *Futur. Gener. Comput. Syst.* **2021**, *115*, 844–856. [CrossRef]
60. Jeon, S.; Moon, J. Malware-Detection Method with a Convolutional Recurrent Neural Network Using Opcode Sequences. *Inf. Sci.* **2020**, *535*, 1–15. [CrossRef]
61. Jha, S.; Prashar, D.; Long, H.V.; Taniar, D. Recurrent neural network for detecting malware. *Comput. Secur.* **2020**, *99*, 102037. [CrossRef]
62. Jahromi, A.N.; Hashemi, S.; Dehghantanha, A.; Choo, K.-K.R.; Karimipour, H.; Newton, D.E.; Parizi, R.M. An improved two-hidden-layer extreme learning machine for malware hunting. *Comput. Secur.* **2020**, *89*, 101655. [CrossRef]
63. Narayanan, B.N.; Davuluru, V.S.P. Ensemble Malware Classification System Using Deep Neural Networks. *Electronics* **2020**, *9*, 721. [CrossRef]
64. Song, X.; Chen, C.; Cui, B.; Fu, J. Malicious JavaScript Detection Based on Bidirectional LSTM Model. *Appl. Sci.* **2020**, *10*, 3440. [CrossRef]
65. Wang, X.; Li, C.; Song, D.; Wang, C. CrowdNet: Identifying Large-Scale Malicious Attacks Over Android Kernel Structures. *IEEE Access* **2020**, *8*, 15823–15837. [CrossRef]
66. Yen, Y.-S.; Sun, H.-M. An Android mutation malware detection based on deep learning using visualization of importance from codes. *Microelectron. Reliab.* **2019**, *93*, 109–114. [CrossRef]
67. Zanni-Merk, C. On the Need of an Explainable Artificial Intelligence. In Proceedings of the 40th Anniversary International Conference on Information Systems Architecture and Technology, Wroclaw, Poland, 15–17 September 2019; p. 3.