CHIMERA: AN ANDROID MALWARE DETECTION METHOD BASED ON MULTIMODAL DEEP LEARNING AND HYBRID ANALYSIS

A PREPRINT

Angelo S. Oliveira
PPGIGC
Universidade Nove de Julho
São Paulo, SP, Brazil
ftwr@angelo5d0.com

Renato J. Sassi
PPGIGC
Universidade Nove de Julho
São Paulo, SP, Brazil
sassi@uni9.pro.br

December 10, 2020

ABSTRACT

The Android Operating System (OS) everywhere, computers, cars, homes, and, of course, personal and corporate smartphones. A recent survey from the International Data Corporation (IDC) reveals that the Android platform holds 85% of the smartphone market share. Its popularity and open nature make it an attractive target for malware. According to AV-TEST, by November 2020, 2.87M new Android malware instances were identified in the wild. Malware detection is a challenging problem that has been actively explored by both the industry and academia using intelligent methods. On the one hand, traditional machine learning (ML) malware detection methods rely on manual feature engineering that requires expert knowledge. On the other hand, deep learning (DL) malware detection methods perform automatic feature extraction but usually require much more data and processing power. In this work, we propose a new multimodal DL Android malware detection method, Chimera, that combines both manual and automatic feature engineering by using the DL architectures, Convolutional Neural Networks (CNN), Deep Neural Networks (DNN), and Transformer Networks (TN) to perform feature learning from raw data (Dalvik Executable (DEX) grayscale images), static analysis data (Android Intents & Permissions), and dynamic analysis data (system call sequences) respectively. To train and evaluate our model, we implemented the Knowledge Discovery in Databases (KDD) process and used the publicly available Android benchmark dataset Omnidroid, which contains static and dynamic analysis data extracted from 22,000 real malware and goodware samples. By leveraging a hybrid source of information to learn high-level feature representations for both the static and dynamic properties of Android applications, Chimera's detection Accuracy, Precision, Recall, and ROC AUC outperform classical ML algorithms, state-of-the-art Ensemble, and Voting Ensembles ML methods, as well as unimodal DL methods using CNNs, DNNs, TNs, and Long-Short Term Memory Networks (LSTM). To the best of our knowledge, this is the first work that successfully applies multimodal DL to combine those three different modalities of data using DNNs, CNNs, and TNs to learn a shared representation that can be used in Android malware detection tasks.

Keywords Android Malware Detection · Multimodal Deep Learning · Ensemble Machine Learning · Computer Security

1 Introduction

Malware, or malicious software, is any software intentionally designed to cause harm to a computer, user, or network [1]. Some malware examples include but are not limited to adwares, backdoors, ransomwares, rootkits, trojan horses, viruses, and worms. Each kind (or family) of malware shares common properties regarding the exploitation techniques used in the victim's environment. In general, Android malware can be characterized by their malicious payloads main targets as financial charges, personal information stealing, privilege escalation, and remote control [2]. Malware poses a significant threat to both end-users and corporations. While end-users might have their personal and financial data stolen or encrypted by ransomwares, corporations might have their security perimeters breached by backdoors and rootkits. The development of defensive mechanisms against malware depends on knowing how malware works. In order to understand malware internals and behavior, one can make use of Malware Analysis techniques.

Malware Analysis is a set of techniques used to dissect malware and understand how it works in order to identify and defeat it [1]. It is based on a subset of techniques known as Static Analysis, Dynamic Analysis, and Hybrid Analysis [3]. Static Analysis provides a set of tools and techniques to understand how malware works without executing it [3]. The main advantage of Static Analysis is that there is no need to execute the malicious file to analyze it. Specialized tools such as decompilers and disassemblers make the information extraction process fast and straightforward. One disadvantage of Static Analysis is related to malware code obfuscation. When the malicious code is obfuscated or encrypted, direct Analysis of the disassembled code is practically impossible. In this scenario, it is helpful to make use of Dynamic Analysis. Dynamic Analysis provides a set of tools and techniques to understand how malware works by executing it in a controlled, isolated environment known as sandbox [3]. By collecting information from the malware execution traces such as system calls, API calls, network activity, file system activity, and so on, it is possible to draw a profile of the application based on its actual behavior. The main advantage of Dynamic Analysis is that its immune to code is obfuscation. Some disadvantages of Dynamic Analysis include its low code coverage and high time consumption. Hybrid Analysis leverages both Static Analysis and Dynamic Analysis to understand malware effectively by taking advantage of both Static and Dynamic Analysis resources.

The information gathered using Malware Analysis can be leveraged for malware detection and classification tasks [4]. The most common, faster, and simpler way of detecting malware is using signature-based methods [4]. Signature-based methods rely on the extraction of malicious patterns from known malware using Malware Analysis techniques. Once the malicious patterns are collected, their presence or absence can be quickly verified in the suspicious files. Some disadvantages of signature-based methods are their inefficiency in detecting polymorphic and metamorphic malware [5], zero-day malware, and the high rate of false positives. Moreover, since polymorphic and metamorphic malware can change their code implementations in runtime, the malicious patterns known by signature-based methods can also be changed in the process, increasing the number of false-negatives.

In order to increase the accuracy of malware detection and classification methods, several ML and DL methods have been proposed. For a comprehensive review, please refer to [6, 7]. The main advantage of ML malware detection methods is their capability of learning malicious patterns from data (e.g., real-world malware samples); however, ML techniques frequently require manual feature engineering in order to achieve higher accuracy, which usually requires a highly specialized workforce, and it is time-consuming. Deep Learning malware detection methods leverage specialized architectures designed for image processing, speech recognition, sequence learning, and so on [8]. The main advantage of DL methods is their capability of performing automatic feature learning from structured and non-structured data of different domains, thus decreasing the work associated with manual feature engineering [8]. In fact, DL methods have achieved state-of-the-art results in image recognition tasks, speech recognition, and natural language processing [8]. DL methods' main disadvantage is the requirement of large volumes of data and processing power to achieve higher accuracy.

More recently, Android malware detection methods using multimodal DL have also been proposed [9, 10, 11, 12, 13]. Multimodal DL uses independent, specialized DL subnetworks to extract high-level feature representations from different data modalities and combines the resulting embeddings into a shared representation that can be used for classification and regression tasks [14]. For example, in Multimodal DL, it is possible to combine both audio and video data for a classification task and achieve better accuracy than using audio and video independently in a unimodal architecture for the same task [14]. In this work, we propose Chimera, a new Android malware detection method based on multimodal DL and hybrid Analysis. As we can see in Figure 1, Chimera is composed of 3 independent DL subnetworks: (1) Chimera-Static (Chimera-S), a DNN [8] to learn high-level feature representations from Android Intents & Permissions using an early fusion layer. (2) Chimera-Raw (Chimera-R), a CNN [15] to learn high-level feature representations from raw data transformed into DEX grayscale images, and (3) Chimera-Dynamic (Chimera-D), a TN encoder [16] to learn high-level feature representations from the system call sequences. Finally, the intermediate fusion network is responsible for implementing shared high-level feature representations using each subnetwork's internal representations and learning correlations that can be leveraged for Android malware detection. Our experiments' results indicate that Chimera's detection Accuracy, Precision, Recall, and ROC AUC outperform classical ML algorithms, state-of-the-art Ensemble and Voting Ensembles ML methods, as well as unimodal DL methods using CNNs, DNNs, TNs, and LSTMs. To the best of our knowledge, this is the first work that uses a combination of raw data, static analysis data, and dynamic analysis data inputs for a multimodal DL method based on CNNs, DNNs, and TNs to learn shared representations that can be applied to Android malware detection. The rest of this paper is organized as follows: Section 2 details the proposed method and the methodology adopted in the research. Section 3 provides the evaluation methods,

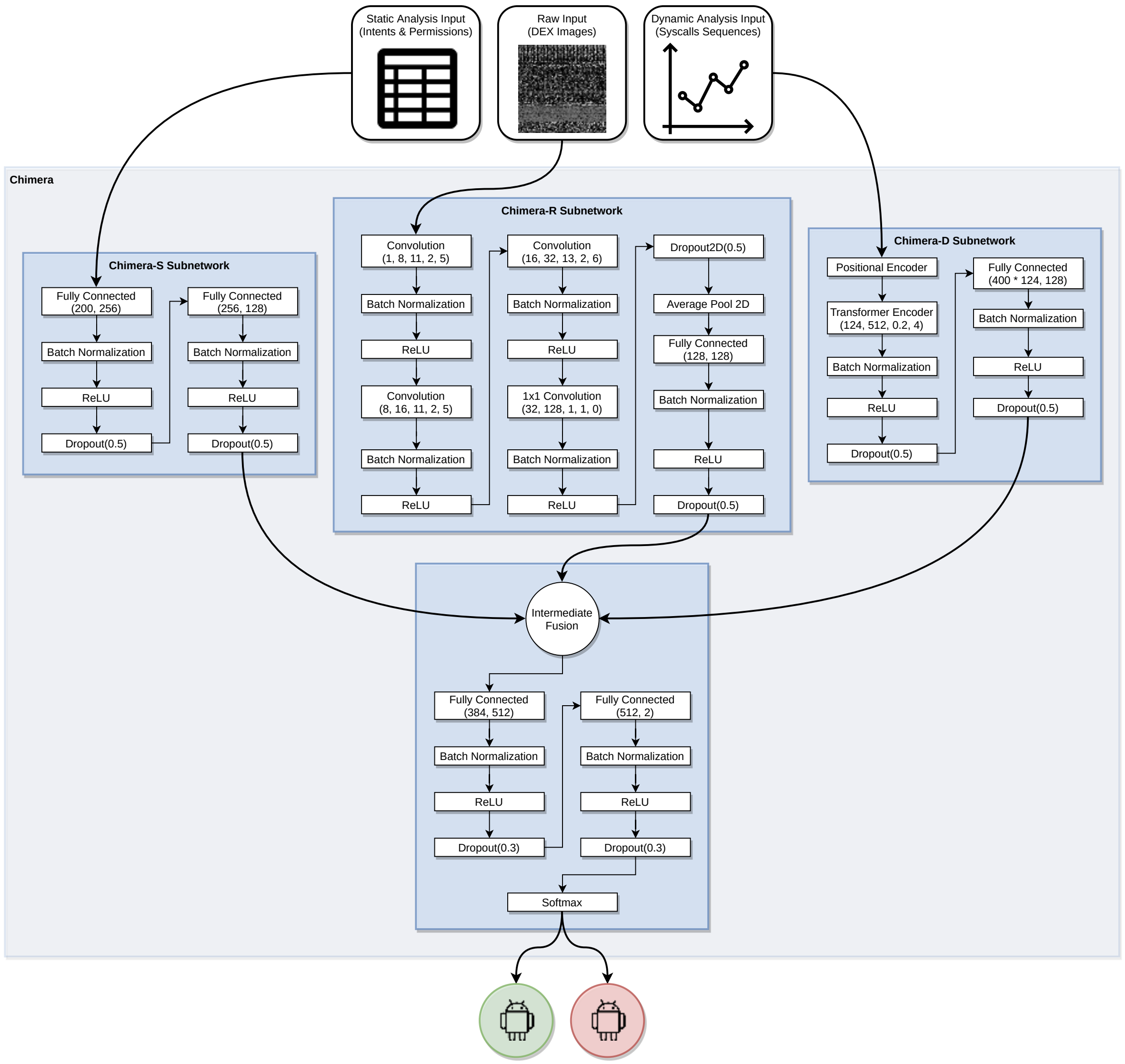


Figure 1: Chimera Android malware detection method architecture.

metrics, results, and performance comparisons between our method and different DL, ML, Ensemble ML, and Voting Ensemble ML methods, as well as a discussion of the obtained results, limitations, and future work. Section 4 outlines the related work. Lastly, Section 5 summarizes the results and impact of our research.

2 Proposed Method and Methodology

This work follows the Knowledge Discovery in Databases (KDD) process [17] and Supervised ML methodology [18] for model selection, training, and evaluation. Supervised ML performs the task of learning a function that maps an input to an output using input-output pairs to adjust the model's parameters. Therefore, in Supervised ML, it is essential to define a labeled training set containing a representative number of instances of each class to promote learning, i.e., low generalization error on the evaluation/test set following the same training set distribution [18]. KDD is an iterative, interactive, and non-trivial process composed of several stages for extracting (useful) patterns from large databases. Figure 2 depicts the KDD process implementation for Chimera. Each KDD stage, i.e., Selection, Preprocessing, Transformation, Data Mining, and Interpretation, is represented by a blue box containing the implementation steps (white boxes) performed in that particular stage. Notice that the KDD stages Selection, Preprocessing, and Transformation contain implementation steps related to data preparation, and the Data Mining and Interpretation stages contain implementation steps related to ML methodology such as model selection, training, and

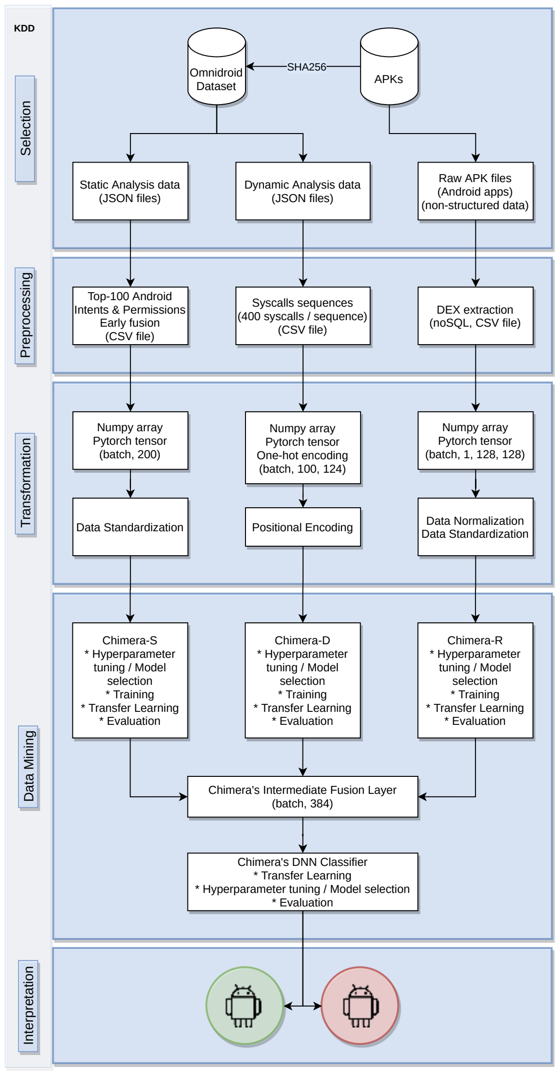

KDD
Selection
Preprocessing
Transformation
Data Mining
Interpretation
Omnidroid Dataset
SHA256
APKs
Static Analysis data (JSON files)
Dynamic Analysis data (JSON files)
Raw APK files (Android apps) (non-structured data)
Top-100 Android Intents & Permissions Early fusion (CSV file)
Syscalls sequences (400 syscalls / sequence) (CSV file)
DEX extraction (noSQL, CSV file)
Numpy array Pytorch tensor (batch, 200)
Numpy array Pytorch tensor One-hot encoding (batch, 100, 124)
Numpy array Pytorch tensor (batch, 1, 128, 128)
Data Standardization
Positional Encoding
Data Normalization Data Standardization
Chimera-S * Hyperparameter tuning / Model selection * Training * Transfer Learning * Evaluation
Chimera-D * Hyperparameter tuning / Model selection * Training * Transfer Learning * Evaluation
Chimera-R * Hyperparameter tuning / Model selection * Training * Transfer Learning * Evaluation
Chimera's Intermediate Fusion Layer (batch, 384)
Chimera's DNN Classifier * Transfer Learning * Hyperparameter tuning / Model selection * Evaluation

Figure 2: KDD process stages of Chimera Android malware detection method. The blue boxes represent each KDD process stage. The white boxes represent the implementation steps for each stage.

evaluation. Since Chimera is a multimodal method, each DL subnetwork (Chimera-S, Chimera-R, and Chimera-D) is responsible for feature extraction from a different data source. Therefore, the Selection, Preprocessing, Transformation, and Data Mining stages are performed independently for each DL subnetwork. Moreover, an additional Data Mining implementation step is executed over the intermediate fusion layer built using high-level feature representations learned by each DL subnetwork. Finally, the Interpretation stage is performed by the last Chimera's DNN classifier layer, resulting in a probability distribution used for binary classification or, more concretely, for Android malware detection. In the context of the KDD process, the knowledge produced by our method can be summarized by its generalization performance, i.e., the detection accuracy resulting from 10-fold cross-validation. The following sections detail the implementation of the KDD process and ML methodology for Chimera.

2.1 Selection, Preprocessing and Transformation

In this work, we used the Android benchmark dataset Omnidroid, introduced by [19]. Omnidroid is a balanced dataset composed of pre-static, static, and dynamic analysis information extracted from 22,000 real malware and benign Android applications. After applying several data preparation and consolidation techniques [19], it presents the information in a structured format (Comma Separated Values (CSV) and Javascript Object Notation (JSON) files). However, it is relevant to note that the Omnidroid dataset does not include the APK (Android Application Package) files used to extract the information due to legal restrictions. Since Chimera-R and Chimera use data from the DEX files,

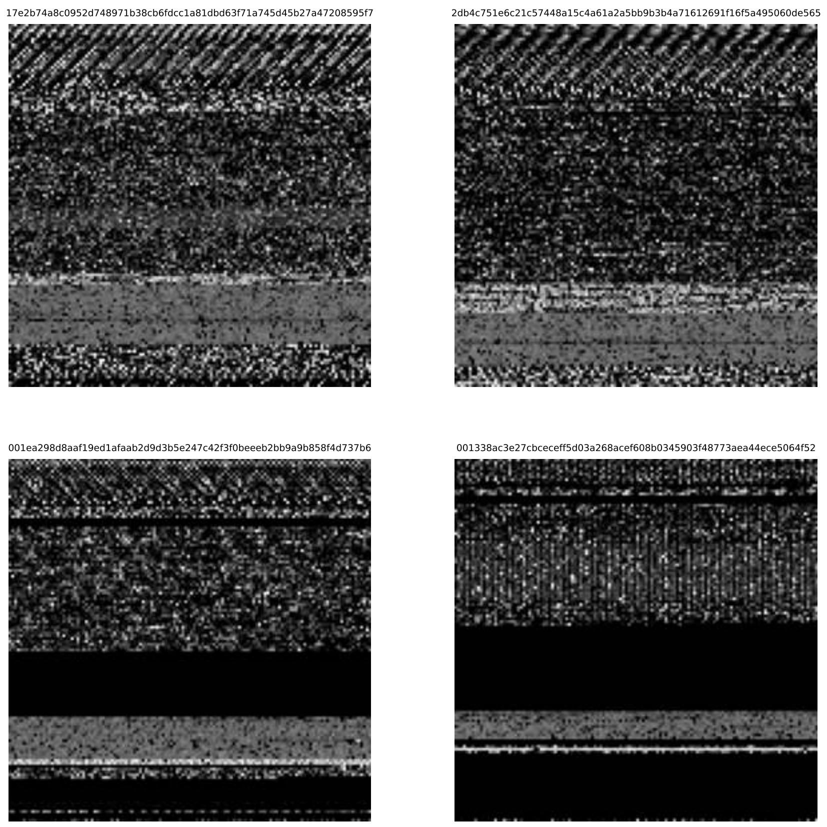

17e2b74a8c0952d748971b38cb6fdcc1a81dbd63f71a745d45b27a47208595f7
2db4c751e6c21c57448a15c4a61a2a5bb9b3b4a71612691f16f5a495060de565
001ea298d8aaf19ed1afaab2d9d3b5e247c42f3f0beeeb2bb9a9b858f4d737b6
001338ac3e27cbceceff5d03a268acef608b0345903f48773aea44ece5064f52

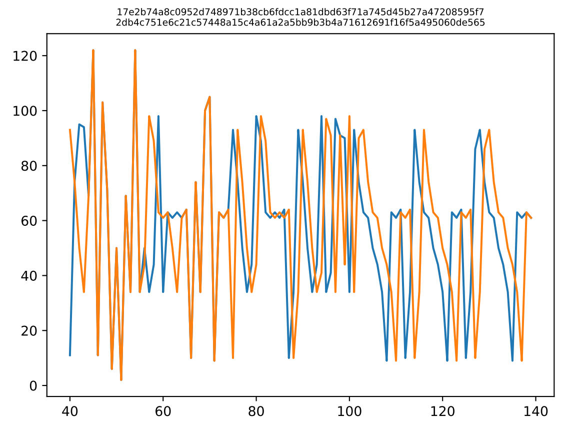
Figure 3: DEX grayscale images of two benign applications in the first row and two Trojan malware in the second row, including the SHA256 hash of each instance.

which are part of the APK files, we also downloaded the APK files from the online malware repositories Androzoo[1] using its free account access, and from Koodous[2] using its premium account access. As we can see in Figure 2 in the KDD Selection stage, we used the SHA256 hashes presented in Omnidroid to download the associated APKs from the online repositories.

2.1.1 Chimera-S

Inspired by the work presented in [20, 21, 22], Chimera-S and Chimera implement an early fusion layer to combine both Android Intents and Android Permissions for malware detection. Android Intents and Android Permissions play an essential role in the Android security architecture by controlling the actions that applications can perform on the OS and the communication between applications. Moreover, as shown by [20], Android Intents and Android Permissions present high discriminative power and low correlation. Thus, features designed using Android Intents and Permissions have high predictive quality. Omnidroid includes the set of Android Intents and Android Permissions for each application. We extracted the top-100 Android Intents and the top-100 Android Permissions from Omnidroid's JSON files, concatenated them into a 200-dimensional feature vector, and saved the result into a CSV file using binary encoding to indicate the presence or absence of a particular Android Intent or Android Permission for each instance. Finally, the Transformation stage was performed by applying a Standardization procedure [18] to set the mean value of each feature to 0 and its standard deviation to 1. This procedure is used to speed up training convergence. Figure 2 summarizes the implementation steps of the KDD process stages Selection, Preprocessing, and Transformation for Chimera-S and Chimera.

[1]https://androzoo.uni.lu/
[2]https://koodous.com/



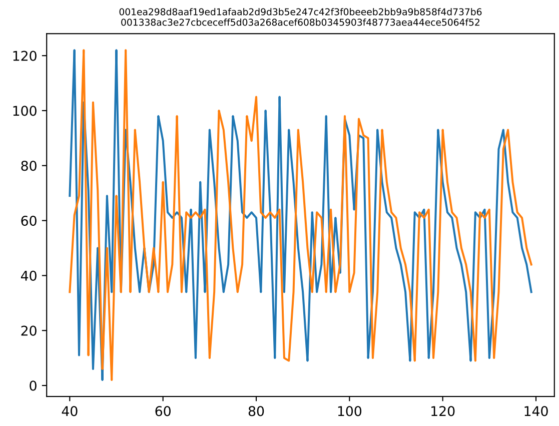


Figure 4: System call sequences of two benign applications in the first column and two Trojan malware in the second column, including the SHA256 hash of each instance. The x-axis represents the time step. The y-axis represents the system call number.

2.1.2 Chimera-R

DEX files contain the executable code in Dalvik format. Similar to the Java Virtual Machine (JVM), the Dalvik Virtual Machine translates DEX opcodes (bytecodes) into native CPU instructions. The DEX format provides a compact and optimized executable module [23]. The methods proposed by [24, 25] make use of DEX bytecodes as images for malware detection and classification tasks. Figure 3 depicts grayscale images representing two benign Android applications and two Android malware instances. Motivated by their work, Chimera-R and Chimera also use data from the DEX files for Android malware detection to perform automatic feature extraction from DEX grayscale images using CNNs. Since DEX files contain non-structured data, we extracted the DEX files from each APK file and saved their content into a NoSQL database. Finally, the Transformation stage was performed by resampling the data to image representations of 1x128x128 pixels (channel, width, height) using the Lanczos resampling algorithm [26], and by applying a Scaling procedure [18] in order to set the values of the features to the same scale (between 0 and 1), and a Standardization procedure [18] in order to set the mean value of the grayscale channel to 0 and its standard deviation to 1. Scaling and Standardization procedures are used to speed up training convergence. We chose the image dimensions to be 1x128x128 since preliminary experiments indicated that smaller images promoted underfitting, while larger images consumed four times more resources (RAM, GPU memory, and processing power) without presenting significant performance improvements. The Lanczos resampling algorithm was chosen because it is considered to present the best compromise for image resampling tasks [26]. Figure 2 summarizes the implementation steps of the KDD process stages Selection, Preprocessing, and Transformation for Chimera-R and Chimera.

2.1.3 Chimera-D

Based on the work introduced by [27], Chimera-D and Chimera also depend on data collected using Dynamic Analysis, particularly system call sequences. System call sequences represent the application behavior through time. More specifically, system call sequences represent the application's interaction with the hardware by calling low-level functions exposed by the OS. Figure 4 presents the system call sequences of two benign Android applications and two Android malware instances, each containing 100-time steps. Omnidroid includes the system calls sequences logged by the strace tool and saved into CSV files. To reduce noise and avoid loops, we removed all the consecutive repeating system calls. Then, we trimmed the sequences to 400-time steps since the smallest resulting sequence after removing the consecutive repeating system calls contained 415-time steps. The information was extracted from Omnidroid's CSV files and saved into a CSV file using an integer encoding where each number is associated with a system call. In total, 124 unique system calls were identified. Finally, the Transformation stage is performed by converting the integer encoded feature to its one-hot encoding representation [18] during the training and evaluation processes. That is a necessary step since each system call should be treated as a categorical feature, converted to its one-hot encoding representation, and then used as input to Chimera-D, which is based on a TN encoder. Figure 2 summarizes the implementation steps of the KDD process stages Selection, Preprocessing, and Transformation for Chimera-D and Chimera.

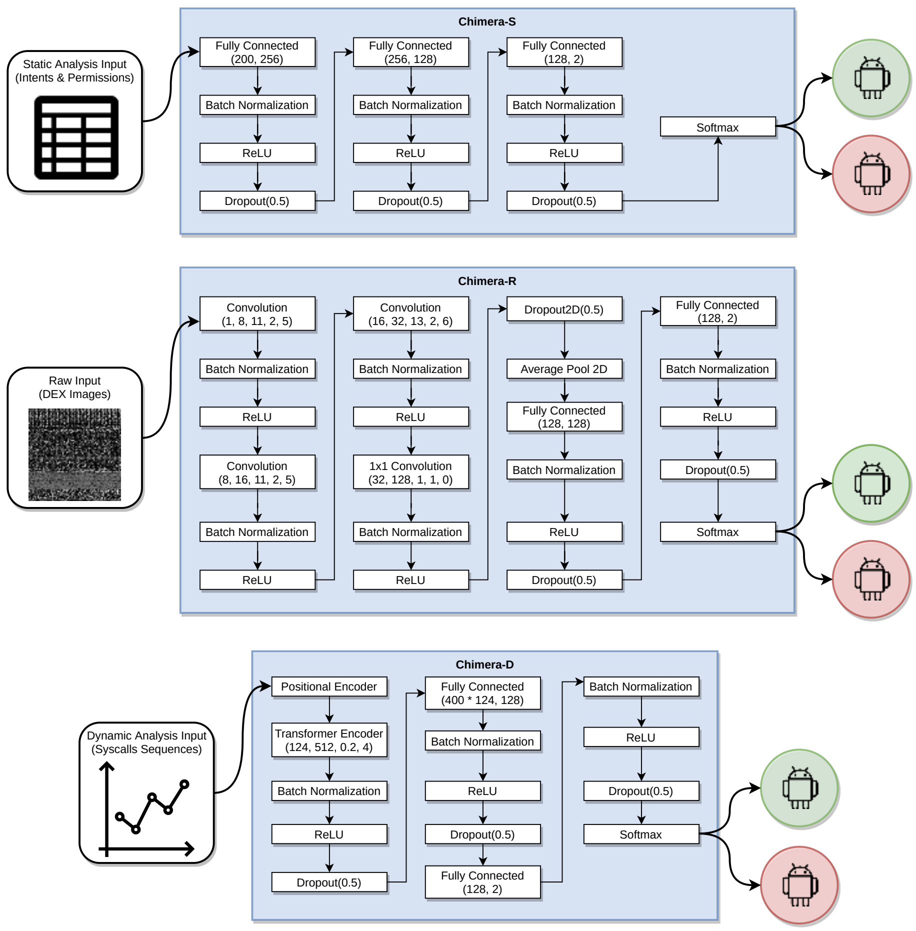


Figure 5: Chimera's Android malware detection method DL subnetworks for Static Analysis data (Chimera-S), DEX grayscale images (Chimera-R), and Dynamic Analysis data (Chimera-D).

2.2 Data Mining and Interpretation

Supervised DL is composed of several techniques for data mining of structured and non-structured data and can be used for classification and regression tasks [28]. DL also provides several techniques to mitigate underfitting and overfitting. Overfitting is a common problem found when building and evaluating DL models since, by definition, DL models are composed of multiple layers and a large number of parameters. Overfitting takes place when a model loses its generalization power. More precisely, the training error reaches a low value, meaning that the network was able to fit the training set accurately; however, at the same time, the evaluation error reaches a high value, meaning that the network was not able to generalize or learn the required patterns from the training set. Commonly used approaches to mitigating overfitting are increasing the dataset's size and variability, decreasing the model's complexity (layers and the total number of parameters) or using specialized DL techniques designed for data standardization in the hidden layers and reduction of the model's complexity. The most common techniques used for mitigating overfitting are Dropout [29] and Batch Normalization [30]. Dropout is used as a regularization mechanism for reducing overfitting by randomly zeroing out the activations' values to prevent complex co-adaptations on the training data, resulting in the thinning of the model's weights by the Backpropagation algorithm [29]. Batch Normalization is used to mitigate internal covariate shift, which is the change in the distribution of the network's activation values, resulting in training stability and slow convergence. Our experiments indicated that both Batch Normalization and Dropout layers play an important role in mitigating overfitting and the training stability in Chimera-S, Chimera-R, Chimera-D, and Chimera fully connected layers, and Batch Normalization plays a similar role for the convolutional layers of Chimera-R and Chimera. Finally, we choose to apply by default the activation function Rectified Linear Unit (ReLU) [31] to introduce non-linearity, help mitigate the vanishing gradient problem, and speed up training convergence.

Taking into account that Chimera-S, Chimera-R, Chimera-D, and Chimera are binary classifiers that can be extended to multiclass classifiers in future work, we included the Softmax activation function after the output layer to encode the high-level feature representations into a probability distribution that can be used for binary classification [28]. Consequently, the Cross-Entropy loss function [28] was introduced to quantify the training and evaluation errors during the learning process. Finally, we chose the Adaptive Moment Estimation (Adam) optimizer [32] to train the models. Adam is a state-of-the-art adaptive learning rate optimization algorithm designed for training DL networks. Adam leverages both momentum and learning rate adaptation to accelerate convergence and avoid local minima and plateaus.

In order to choose the best architectures for Chimera-S, Chimera-R, Chimera-D, and Chimera, we performed model selection (or hyperparameter tuning) using grid search cross-validation with 10-fold cross-validation to estimate the generalization error [33]. Grid search cross-validation exhaustively generates candidate architectures using a supplied grid of hyperparameters values and applies a 10-fold cross-validation procedure to estimate the model's generalization error based on a selected performance metric. In 10-fold cross-validation, the dataset is initially shuffled and split into ten parts containing approximately the same number of instances and the same proportion of malware and goodware instances each. Next, the selected model is trained on nine parts and evaluated on the remaining part. The process repeats until all the parts have been selected for evaluation. Finally, the estimation of the model's performance is calculated by averaging the results of each evaluation. In this work, we choose the Accuracy metric (See Equation 1) to guide the model selection process since the Ominidroid dataset is balanced; thus, the Accuracy metric represents the percentage of correct predictions on the evaluation sets.

2.2.1 Chimera-S

As depicted in Figure 5, Chimera-S introduces a DNN architecture [28] with one input layer containing 200 neurons: 100 neurons for Android Intentions features and 100 neurons for Android Permissions features. Two hidden layers containing 256 and 128 neurons respectively, and one output layer containing two neurons followed by a Softmax layer. Each fully connected layer is followed by a ReLU activation function. We included Dropout and Batch Normalization layers between the fully connected layers and between the output layer and the Softmax layer to mitigate overfitting. In addition, the best results were found when using the step decay schedule for the learning rate decay strategy. In the step decay schedule, the learning rate is reduced by a factor every predefined number of epochs, which might result in faster training convergence.

The following hyperparameters were considered for model selection:

- Number of neurons in the first hidden layer: {128, 256, 512}
- Number of neurons in the second hidden layer: {128, 256, 512}
- Dropout probability: {0.1, 0.3, 0.5}
- Number of epochs: {10, 20, 30, 50, 100}
- Learning rate step decay schedule factor: {0.1, 0.5}

After model selection using 10-fold cross-validation on 216 candidate architectures (2160 fits), the best set of hyperparameters found was:

- Number of neurons in the first hidden layer: 256
- Number of neurons in the second hidden layer: 128
- Dropout probability: 0.5
- Number of epochs: 50
- Learning rate step decay schedule factor: 0.5
- Learning rate step decay schedule steps: 10

Please see Figure 5 for the optimized Chimera-S architecture and Figure 1 for the optimized Chimera architecture. Please refer to Section 3 for detailed performance evaluation and discussion of the trained models Chimera-S and Chimera using the optimal hyperparameters presented in this section.

2.2.2 Chimera-R

As we can see in Figure 3, the 2nd row depicts two malware instances from the same family (Trojan). It is easy to see that both instances share common spatial (visual) patterns. The same holds for the benign instances in the 1st row. If the spatial patterns across the instances of a dataset have enough discriminative power to identify the instance's class, then it is possible to use ML or DL techniques to leverage the information contained in the spatial patterns for detection and classification tasks. In fact, [24] proposed a CNN architecture for Android malware classification using DEX grayscale

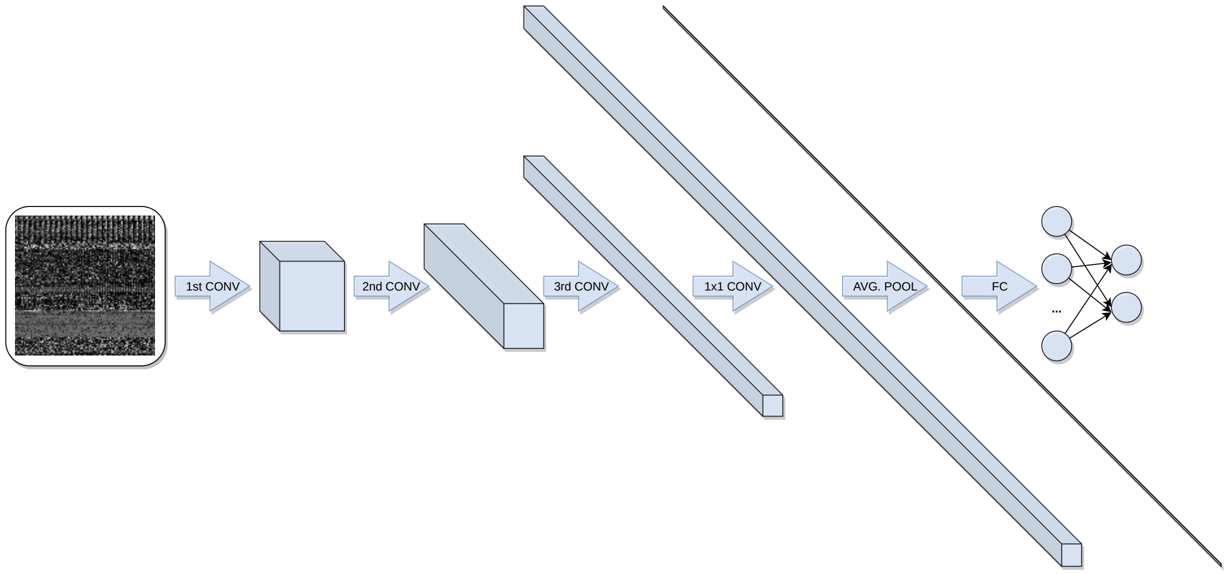


Figure 6: Chimera-R CNN architecture (simplified and out of scale).

images, and [25] introduced a CNN architecture for Android malware detection using DEX opcodes translated to RGB images. CNN is a class of DL network commonly applied to computer vision problems [15] and were inspired by the animal visual cortex. CNNs are shift-invariant and based on shared-weights. These properties allow CNNs to learn spatial patterns from images and reuse them to recognize those patterns independently of their positions. Moreover, shared-weights reduce overfitting and training/inference time. As an example, a CNN can learn a filter (or kernel) able to recognize a high-level feature such as an eye and another filter able to recognize another high-level feature such as a nose, and by using multiple convolutional layers, CNNs combine both high-level features into higher-level features that can be used for face recognition.

Our work follows a similar approach proposed by [24, 25] and introduces a new CNN architecture inspired by the Residual Networks (ResNet) architecture [34]. As we can see in Figure 5, Chimera-R is composed of 4 convolutional layers used for feature extraction and a final DNN used for Android malware detection. The 5-tuple that defines each convolutional layer comprises the number of input channels, the number of output channels, the filter (or kernel) size, the stride of the filter, and the padding [28]. Notice that the number of output channels doubles in the second and third layers, and finally, the number of output channels is multiplied by 4 in the last 1x1 convolutional layer [35]. Also, notice that the stride used in the first, second, and third layers is equal to 2. The effect of those hyperparameters in the architecture is as follows: After the 1st or second convolutional layer processes the image, its dimensions (width and height) are reduced by a factor of 2, and the number of extracted feature maps (depth) is increased by a factor of 2, thus, at the same time increasing the number of feature maps containing high-level feature representations and reducing their dimensionality. Finally, the Global Average Pooling operation [28] is applied after the 1x1 convolution to collapse the resulting tensor of feature maps into a tensor of real numbers that summarize each feature map. See Figure 6 for a simplified, out of scale representation of the Chimera-R CNN architecture. From this point on, the information is passed to a DNN with one hidden layer containing 128 neurons and one output layer containing two neurons, followed by a Softmax activation function. Each convolutional layer and fully connected layer is followed by a ReLU activation function to introduce non-linearity and prevent the vanishing gradient problem. To mitigate overfitting, similar to Chimera-S, we included Dropout and Batch Normalization layers between the fully connected layers and Batch Normalization between the convolutional layers. Contrary to what was verified for Chimera-S, our experiments indicated that the use of both Dropout and Batch Normalization layers between the convolutional layers led to overfitting and training instability. In addition, similar to Chimera-S (2.2.1), best results were found when using the step decay schedule for the learning rate decay strategy.

The following hyperparameters were considered for model selection:

- Filter size of the 1st convolutional layer: {3, 5, 7, 9, 11, 13}
- Filter size of the 2nd convolutional layer: {3, 5, 7, 9, 11, 13}
- Filter size of the 3rd convolutional layer: {3, 5, 7, 9, 11, 13}

- Dropout probability: {0.1, 0.3, 0.5}
- Number of epochs: {30, 40, 50}
- Learning rate step decay schedule factor: {0.1, 0.5}

After model selection using 10-fold cross-validation on 3888 candidate architectures (38880 fits), the best set of hyperparameters found was:

- Filter size of the 1st convolutional layer: 11
- Filter size of the 2nd convolutional layer: 11
- Filter size of the 3rd convolutional layer: 13
- Dropout probability: 0.5
- Number of epochs: 40
- Learning rate step decay schedule factor: 0.1
- Learning rate step decay schedule steps: 10

Please see Figure 5 for the optimized Chimera-R architecture and Figure 1 for the optimized Chimera architecture. Please refer to Section 3 for detailed performance evaluation and discussion of the trained models Chimera-R and Chimera using the optimal hyperparameters presented in this section.

2.2.3 Chimera-D

As we can see in Figure 4, the 2nd column depicts two malware instances' system call sequences overlapped. Also, notice that those two malware instances belong to the same family (Trojan). It is easy to see that both instances share common temporal patterns. The same holds for the benign instances in the 1st column. Suppose the temporal patterns across the instances of a dataset have enough discriminative power to identify the instance's class. In that case, it is possible to use ML or DL techniques to leverage the information contained in the temporal patterns for detection and classification tasks. In fact, [27] proposed an LSTM architecture to implement a neural probabilistic language model for Android malware detection using system call sequences. LSTM is a class of DL networks based on Recurrent Neural Networks (RNN) and capable of learning long-term dependencies on temporal/sequential data [36]. Our work is based on a different architecture for sequence learning: the Transformer Networks (TN) [16]. TN is a state-of-the-art encoder-decoder DL architecture designed to handle sequential data, such as natural language. Unlike RNNs, Transformers do not need to process sequential data in order. Due to this feature, Transformers facilitate parallelization during training time. Also, Transformers implement the Attention mechanism. Attention is used to let the network access any previous states and weights and learn which ones are more relevant for the task at hand. As we can see in Figure 5, Chimera-D is composed of a positional encoder that is used to add positional information to the inputs represented as 124-dimensional one-hot encoding vectors (2.1.3). The result is passed to the TN encoder layer for sequence learning and temporal feature extraction. Finally, a DNN is used for Android malware detection. Notice that Chimera-D and Chimera only make use of the encoder part of the TN. The TN encoder comprises an input layer of 124 neurons, a feedforward layer of 512 neurons, and four attention heads. The DNN contains three layers. The first layer has 400 * 124 neurons representing the high-level features extracted by the TN encoder. The second layer is composed of 128 neurons, and the output layer contains two neurons, followed by a Softmax activation function. Similar to Chimera-S and Chimera-R, we included Dropout and Batch Normalization layers after the TN encoder and the fully connected layers to mitigate overfitting and increase training stability. We used the ReLU activation function in the TN encoder and in Chimera-D to introduce non-linearity. To train Chimera-D, we used a different learning rate scheduling strategy, the learning rate warm-up, to increase the learning rate after every epoch by a constant factor. Learning rate warm-up mitigates premature convergence, and it is an essential technique for training TNs [16].

The following hyperparameters were considered for model selection:

- Number of neurons in the TN feedforward layer: {128, 256, 512, 1024}
- Number of neurons in the DNN hidden layer: {128, 256, 512}
- TN Dropout probability: {0.1, 0.2, 0.3, 0.4, 0.5}
- DNN Dropout probability: {0.1, 0.3, 0.4, 0.5}
- Number of epochs: {30, 40, 50}

After model selection using 10-fold cross-validation on 900 candidate architectures (9000 fits), the best set of hyperparameters found was:

- Number of neurons in the TN feedforward layer: 512
- Number of neurons in the DNN hidden layer: 128
- TN Dropout probability: 0.2
- DNN Dropout probability: 0.5
- Number of epochs: 30
- Learning rate warm-up schedule factor: 1.033
- Learning rate warm-up schedule steps: 1

Please see Figure 5 for the optimized Chimera-D architecture and Figure 1 for the optimized Chimera architecture. Please refer to Section 3 for detailed performance results of the trained models Chimera-D and Chimera using the optimal hyperparameters presented in this section.

2.2.4 Chimera

As we can see in Figure 1, once the subnetworks Chimera-S, Chimera-R, and Chimera-D have forward propagated their inputs, a shared representation layer is implemented by concatenating (feature-wise) their results and passed to the last Chimera's DNN classifier for Android malware detection. Similar to what was verified by [9, 37], we found out that training Chimera as a single model resulted in underfitting one subnetwork and overfitting of the other subnetworks. Taking that into account, we trained Chimera-S, Chimera-R, and Chimera-D separately using the optimized hyperparameters presented in the Sections 2.2.1, 2.2.2, and 2.2.3 respectively, and used Transfer Learning [38] to combine them into the final Chimera architecture. Transfer Learning works by training each model separately and saving their weights for later use and integration with other models. During training time, the weights of the pre-trained models are frozen, and the weights of the new model are trained, taking advantage of what was previously learned by the pre-trained models. Notice that Chimera's subnetworks do not include the last fully connected layers from their counterparts since Chimera itself needs to be optimized and trained to classify the high-level feature representations from the intermediate fusion layer. Finally, as we can see in Figure 1, Chimera's DNN classifier is composed of one input layer containing 384 neurons, one hidden layer containing 512, and one output layer containing two neurons followed by a Softmax activation function. A ReLU activation function follows each fully connected layer to introduce non-linearity. We included Dropout and Batch Normalization layers between the fully connected layers to mitigate overfitting. Moreover, similar to Chimera-S (2.2.1) and Chimera-R (2.2.2), best results were found when using the step decay schedule for the learning rate decay strategy.

The following hyperparameters were considered for model selection:

- Number of neurons in the hidden layer of the DNN: {128, 256, 512}
- Dropout probability: {0.1, 0.3, 0.5}
- Number of epochs: {30, 40, 50}
- Learning rate step decay schedule factor: {0.1, 0.5}

After model selection using 10-fold cross-validation on 54 candidate architectures (540 fits), the best set of hyperparameters found was:

- Number of neurons in the hidden layer of the DNN: 512
- Dropout probability: 0.3
- Number of epochs: 30
- Learning rate step decay schedule factor: 0.5
- Learning rate step decay schedule steps: 10

Please see Figure 1 for the optimized Chimera architecture. Please refer to Section 3 for detailed performance results of the trained Chimera model using the optimal hyperparameters presented in this section.

3 Performance Evaluation and Discussion

To evaluate our method, we performed 10-fold cross-validation using the preprocessed/transformed Omnidroid dataset (See 2.1) on the following ML/DL algorithms/methods:

Classifier	Accuracy	Precision	Recall	AUC ROC	Fit Time (s)
Chimera	**0.909 ± (0.001)**	**0.944 ± (0.003)**	**0.866 ± (0.003)**	**0.972 ± (0.000)**	240.041
Random Forest	**0.835 ± (0.003)**	0.862 ± (0.002)	**0.792 ± (0.004)**	**0.913 ± (0.002)**	7.350
Extra Trees	0.835 ± (0.002)	0.867 ± (0.002)	0.787 ± (0.004)	0.906 ± (0.002)	11.164
Chimera-S	0.832 ± (0.002)	**0.887 ± (0.004)**	0.755 ± (0.003)	0.908 ± (0.002)	42.330
Bagging	0.825 ± (0.002)	0.850 ± (0.002)	0.784 ± (0.004)	0.905 ± (0.002)	43.018
MLP	0.823 ± (0.003)	0.842 ± (0.005)	0.789 ± (0.004)	0.883 ± (0.002)	37.947
RBF SVM	0.815 ± (0.002)	0.844 ± (0.003)	0.766 ± (0.003)	0.876 ± (0.002)	168.844
K-NN	0.811 ± (0.001)	0.835 ± (0.002)	0.767 ± (0.002)	0.883 ± (0.002)	4.419
Decision Tree	0.807 ± (0.003)	0.831 ± (0.004)	0.764 ± (0.005)	0.824 ± (0.003)	**0.947**
Gradient Boosting	0.803 ± (0.002)	0.813 ± (0.002)	0.778 ± (0.004)	0.882 ± (0.002)	37.854
Logistic Regression	0.792 ± (0.002)	0.803 ± (0.002)	0.764 ± (0.003)	0.866 ± (0.002)	2.535
SVM	0.788 ± (0.002)	0.799 ± (0.003)	0.763 ± (0.003)	0.860 ± (0.003)	36.207
AdaBoost	0.783 ± (0.003)	0.792 ± (0.003)	0.759 ± (0.004)	0.862 ± (0.002)	14.430
Naive Bayes	0.585 ± (0.002)	0.859 ± (0.008)	0.189 ± (0.003)	0.783 ± (0.003)	**0.325**

Table 1: 10-fold cross-validation results of different methods on Static Analysis data (Android Intents & Permissions). The text in bold indicates the best mean values for each metric. The dark gray row indicates the best performing method. The light gray row indicates the performance of Chimera-S.

1. Classical ML algorithms [18]: Decision Tree, Gaussian Naive Bayes, K-Nearest Neighbors (K-NN), Logistic Regression, Multi-layer Perceptron (MLP), Support Vector Machines (SVM), and Radial Basis Function (RBF) SVM.
2. State-of-the-art Ensemble ML algorithms [39]: AdaBoost, Bagging, Extra Trees, Gradient Boosting, and Random Forest.
3. Chimera's DL subnetworks: Chimera-S, Chimera-R, and Chimera-D. See Figure 5.
4. DL for sequence classification: LSTM networks.

Notice that we set the number of CPU cores to four to all the ML and Ensemble ML methods. The number of estimators to 100 for the Ensemble ML methods and all the other hyperparameters were set to the default values used in the scikit-learn library [40]. The LSTM network architecture was defined as follows: LSTM hidden layer of 256 neurons and a fully connected layer of 256 neurons, followed by a Softmax activation function. Also, we applied Dropout and Batch Normalization to mitigate overfitting. Finally, we compared Chimera's performance results to the state-of-the-art Voting Ensemble ML method results presented in [19].

The following performance metrics were chosen for results evaluation and comparisons:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

Where TP, TN, FP, FN stand for True Positive, True Negative, False Positive, and False Negative, respectively.

In the context of this work, the Accuracy metric (See Equation 1) represents the total number of correct detections over the total number of instances. Since the Omnidroid dataset is a balanced dataset [19], the Accuracy metric directly represents the percentage of correct detections. Notice that if the number of false positives and the number of false negatives are equal to zero, the method achieved the highest possible Accuracy. Therefore, the higher the Accuracy, the better is the overall method's performance. The Precision metric (See Equation 2) represents the total number of correct malware detections over the total number of malware detections. According to Equation 2, if the number of false positives is equal to zero, then the method achieved the highest possible Precision. The Recall metric (See Equation 3) represents the total number of correct malware detections over the total number of malware instances. According to Equation 3, if the number of false negatives is equal to zero, then the method achieved the highest possible Recall. It is

Classifier	Accuracy	Precision	Recall	AUC ROC	Fit Time (s)
Chimera	**0.909 ± (0.001)**	**0.944 ± (0.003)**	**0.866 ± (0.003)**	**0.972 ± (0.000)**	240.041
Chimera-R	**0.801 ± (0.002)**	**0.816 ± (0.004)**	0.777 ± (0.004)	**0.885 ± (0.001)**	177.506
RBF SVM	0.789 ± (0.002)	0.797 ± (0.004)	0.774 ± (0.003)	0.868 ± (0.002)	17,748.879
Bagging	0.765 ± (0.002)	0.766 ± (0.004)	0.763 ± (0.003)	0.856 ± (0.002)	20,045.508
Extra Trees	0.765 ± (0.002)	0.765 ± (0.003)	0.764 ± (0.004)	0.856 ± (0.002)	147.221
Random Forest	0.765 ± (0.003)	0.762 ± (0.004)	0.769 ± (0.004)	0.856 ± (0.002)	174.734
MLP	0.758 ± (0.003)	0.762 ± (0.003)	0.748 ± (0.006)	0.841 ± (0.003)	798.400
Gradient Boosting	0.756 ± (0.003)	0.742 ± (0.003)	0.783 ± (0.004)	0.838 ± (0.002)	5,941.192
AdaBoost	0.705 ± (0.002)	0.695 ± (0.002)	0.730 ± (0.005)	0.786 ± (0.002)	1,835.485
Logistic Regression	0.701 ± (0.002)	0.699 ± (0.003)	0.707 ± (0.005)	0.776 ± (0.002)	**50.412**
SVM	0.690 ± (0.003)	0.688 ± (0.004)	0.696 ± (0.005)	0.734 ± (0.002)	453.465
Decision Tree	0.672 ± (0.002)	0.668 ± (0.002)	0.681 ± (0.003)	0.671 ± (0.002)	275.743
K-NN	0.647 ± (0.002)	0.593 ± (0.002)	**0.938 ± (0.005)**	0.767 ± (0.009)	76.280
Naive Bayes	0.624 ± (0.003)	0.586 ± (0.002)	0.846 ± (0.005)	0.641 ± (0.004)	**16.494**

Table 2: 10-fold cross-validation results of different methods on Static Analysis data (DEX grayscale images). The text in bold indicates the best mean values for each metric. The dark gray row indicates the best performing method. The light gray row indicates the performance of Chimera-R.

important to notice that, in the context of malware detection methods, false negatives pose a much more significant threat to the users than false positives. On the one hand, a false positive means that goodware was detected as malware, which usually does not cause any harm; on the other hand, a false negative means malware was detected as a goodware, thus, bypassing the detection method. Also, we used the Area Under the Receiver Operating Characteristic Curve (AUC ROC) metric [41] and the Fit Time for performance comparisons. The AUC ROC metric is used to summarize a binary classifier's performance as its discrimination threshold is varied. Fit Time refers to the amount of time (in seconds) necessary to train each method.

As we can see in Tables 1, 2, and 3, the results of 10-fold cross-validation using Static Analysis data (Android Intents & Permissions), Static Analysis data (DEX grayscale images), and Dynamic Analysis data (system call sequences) show that Chimera achieved the best performance for all the considered metrics except the Fit Time. Chimera uses a shared representation layer for Android malware detection; thus, it takes advantage of multiple data modalities. Moreover, it also takes advantage of automatic feature engineering by using DL architectures and manual feature engineering applied to (1) The early fusion layer of Chimera-S, (2) The system call sequences of Chimera-D, and (3) The DEX images resampling of Chimera-R. Regarding the Fit Time metric, it is possible to accelerate training/inference by using a faster Graphical Processing Unit (GPU). It is important to notice that Chimera achieved higher performance than its subnetworks evaluated independently as we can see in Tables 1, 2, and 3. The reason for that is because Chimera learned to correlate information from multiple modalities of data, increasing true positives and true negatives, and decreasing the number of false positives and false negatives, thus increasing its Accuracy, Precision, Recall, and AUC ROC.

As presented in Table 1, Chimera-S achieved fourth place, showing better performance than all the classical ML algorithms and some of the Ensemble ML algorithms. Notice that Chimera-S achieved the second-best Precision. Although Chimera-S implements a DNN, its architecture is relatively shallow. Future work will investigate more deep architectures for Chimera-S.

As presented in Table 2, Chimera-R achieved second place, showing better performance than all the classical and Ensemble ML algorithms. Chimera-R is based on CNNs, which are specialized DL architectures for image processing. Interestingly enough, the RBF SVM achieved third place, showing better performance than all the classical and Ensemble ML algorithms; however, since the RBF SVM presents $\mathcal{O}(n^3)$ time complexity and $\mathcal{O}(n^2)$ space complexity, it does not scale well for problems with large feature vectors, consequently requiring a large amount of resources and processing time. Our experiment took approximately five hours for fitting the RBF SVM, approximately three minutes for fitting Chimera-R, and approximately four minutes for fitting Chimera. Notice that the K-NN algorithm achieved the best Recall. Also, notice that its Accuracy and Precision present low values. This scenario indicates that the K-NN algorithm became biased towards malware instances, thus classifying most instances as malware, which increased the Recall and decreased the Precision, Accuracy, and AUC ROC.

As presented in Table 3, Chimera-D achieved fifth place, showing better performance than all the classical ML algorithms and some Ensemble ML algorithms. Chimera-D is based on TNs, which are state-of-the-art DL architectures

Classifier	Accuracy	Precision	Recall	AUC ROC	Fit Time (s)
Chimera	**0.909 ± (0.001)**	**0.944 ± (0.003)**	**0.866 ± (0.003)**	**0.972 ± (0.000)**	240.041
Gradient Boosting	**0.717 ± (0.003)**	0.735 ± (0.004)	0.666 ± (0.004)	**0.787 ± (0.002)**	1,061.624
AdaBoost	0.712 ± (0.003)	0.733 ± (0.004)	0.654 ± (0.003)	0.777 ± (0.003)	413.008
Bagging	0.712 ± (0.003)	0.723 ± (0.003)	**0.672 ± (0.004)**	0.773 ± (0.002)	2,259.014
Chimera-D	0.711 ± (0.002)	**0.751 ± (0.005)**	0.620 ± (0.006)	0.783 ± (0.003)	220.275
RBF SVM	0.711 ± (0.003)	0.724 ± (0.003)	0.666 ± (0.003)	0.774 ± (0.003)	8,125.626
Logistic Regression	0.704 ± (0.002)	0.715 ± (0.003)	0.663 ± (0.005)	0.766 ± (0.003)	67.017
Random Forest	0.701 ± (0.003)	0.707 ± (0.003)	0.670 ± (0.004)	0.770 ± (0.003)	61.147
Extra Trees	0.697 ± (0.003)	0.700 ± (0.004)	0.672 ± (0.003)	0.765 ± (0.003)	85.011
SVM	0.684 ± (0.003)	0.697 ± (0.005)	0.637 ± (0.010)	0.736 ± (0.003)	**38.018**
MLP	0.666 ± (0.004)	0.668 ± (0.005)	0.644 ± (0.006)	0.726 ± (0.004)	669.931
K-NN	0.658 ± (0.003)	0.683 ± (0.005)	0.569 ± (0.006)	0.699 ± (0.004)	41.194
LSTM	0.647 ± (0.017)	0.702 ± (0.025)	0.573 ± (0.054)	0.719 ± (0.003)	82.175
Decision Tree	0.621 ± (0.003)	0.620 ± (0.004)	0.597 ± (0.004)	0.623 ± (0.003)	77.641
Naive Bayes	0.518 ± (0.001)	0.711 ± (0.017)	0.034 ± (0.002)	0.510 ± (0.001)	**7.531**

Table 3: 10-fold cross-validation results of different methods on Dynamic Analysis data (System call sequences). The text in bold indicates the best mean values for each metric. The dark gray row indicates the best performing method. The light gray row indicates the performance of Chimera-D.

for sequence learning. Once again, the RBF SVM achieved a good position, the 6th place, showing better performance than all the classical ML algorithms; however, since the RBF SVM presents $\mathcal{O}(n^3)$ time complexity and $\mathcal{O}(n^2)$ space complexity, it does not scale well for problems with large feature vectors, consequently requiring a large amount of resources and processing time. Our experiment took approximately 2.5 hours for fitting the RBF SVM and approximately 3.5 minutes for fitting Chimera-D. Surprisingly enough, LSTM networks, which are specialized DL architectures for sequence learning, showed poor performance. The results presented in Table 3 show that all the algorithms except Chimera achieved less than 72% Accuracy. A possible reason for that is that the sequences of system calls do not present a high discriminative power. Another reason can be related to the size of the sequences used (2.1.3). The results presented in Table 3 clearly show one advantage of using a multimodal method such as Chimera, which does not depend only on one data modality.

Finally, we also compared Chimera to the state-of-the-art Voting classifier proposed by [19]. First of all, both Chimera and the Voting classifier use the same dataset, Omnidroid, introduced in the same paper [19]. The Voting classifier includes a Random Forest classifier for the static features and a Bagging classifier for the dynamic features. Each classifier contributes to the final classification result according to their weights chosen using model selection. Using 100 estimators for each classifier, the Voting classifier achieved an Accuracy of 0.897 ± (0.008) and a Precision of 0.897 ± (0.007).

We based our experimental platform on an Intel (R) Core (R) i7-8750H CPU @ 2.20GHz, 12 cores, 64 GB memory, and four Nvidia GeForce GTX 1080 Ti graphics cards. We mainly used PyTorch [42], Numpy [43], Pandas [44], and Skorch [45] for the implementations.

3.1 Limitations and Future Work

Chimera is, by definition, a binary classifier designed for Android malware detection. Malware analysts might take advantage of knowing to which family a malware belongs to perform incident response more effectively. To accomplish that, Chimera will be extended to malware multiclass classification in future work.

Chimera was explicitly designed for Android malware detection; however, our experiments indicate that it is possible to extend it to Windows malware detection and classification.

Finally, Chimera, Chimera-S, Chimera-R, and Chimera-D are black-box methods. Malware analysts might take advantage of knowing why an instance was detected as malware and why it belongs to a particular family. Future work will investigate DL interpretability methods and how to apply them to Chimera.

4 Related Work

The first Android malware detection method based on multimodal deep learning was introduced by [9]. The authors used Static Analysis data extracted from different sources: Android Manifest files, decompiled DEX files and disassembled shared libraries. A multimodal DL architecture based on several DNNs was proposed for feature extraction, and an intermediate fusion layer was introduced to build shared representations that can be used for malware detection. In addition, the authors also introduced two types of feature vector generation methods: existence-based and similarity-based. [10] introduced a multimodal DL method that implements an intermediate fusion layer composed of features extracted from Static Analysis data: Permissions and Hardware Features. DNNs were used to implement both the feature extractors and the classifier. The authors evaluated the performance of using each modality separately and in the multimodal setting. [11] proposed a multimodal DL method that implements an early fusion layer composed of features extracted from Static Analysis data: Android Manifest files and Java API modules. Moreover, the authors also included the Sigpid (Significant Permission Identification) as a feature since SigPid was evaluated to have high discriminative power. Finally, a DNN was introduced for feature extraction and malware detection. [12] introduced a multimodal DL method to extract features from Static Analysis data: Android Manifest files and decompiled DEX files. The multimodal DL architecture is based on CNNs for feature extraction, an intermediate fusion layer to build shared representations of the extracted features and a DNN for malware detection. Also, the authors designed a backtracking module for an interpretable explanation. [13] developed a multimodal DL method to extract features from Static and Dynamic Analysis data and proposed a multimodal DL architecture based on several DNNs and an intermediate fusion layer to build the shared representations that can be used for malware detection.

Although our work is concerned with the Android malware detection problem, several key insights were gained from the work [37]. In [37], the authors introduced Hydra, a multimodal DL framework for Windows malware classification. The authors developed a multimodal DL architecture to extract features from three modalities of Static Analysis data: API-based, byte-based, and opcode-based. A DNN was introduced to extract features from the API-based data, and a CNN was designed to extract features from the byte-based and opcode-based data. Finally, an intermediate fusion layer was developed to build shared representations passed to a DNN for malware classification.

5 Conclusion

In this work, we proposed Chimera, a new Android malware detection method based on multimodal DL and Hybrid Analysis. We combined five different approaches to achieve superior performance: (1) Multimodal DL to generate and classify the intermediate fusion layer containing shared representations of high-level features extracted from different data sources. (2) Specialized DL architectures able to extract high-level feature representations from relational, spatial, and temporal data. (3) Hybrid Analysis results from a source of information (the Omnidroid benchmark dataset) containing high-quality data extracted from real-world Android applications using Static and Dynamic Analysis techniques. (4) A combination of manual and automatic feature engineering techniques for each data modality, and (5) The use of the KDD process and ML methodology for the methods implementation, model selection, training, and evaluation. The result of our experiments showed that Chimera's Accuracy, Precision, Recall, and AUC ROC outperform the classical ML methods Decision Tree, Gaussian Naive Bayes, K-NN, Logistic Regression, MLP, SVMs, RBF SVMs; the state-of-the-art Ensemble ML algorithms AdaBoost, Bagging, Extra Trees, Gradient Boosting, and Random Forest; the Chimera's DL subnetworks: Chimera-S, Chimera-R, and Chimera-D, based on DNNs, CNNs, and TNs, respectively; DL sequence learning method based on LSTM, and the state-of-the-art Voting Ensemble method proposed by the creators of the Omnidroid dataset. Future work will expand Chimera to Android malware multiclass classification and include DL interpretability capabilities to support malware analysts and researchers fighting against Android malware. Code implementations and datasets will be available at http://www.angelo5d0.com/.

6 Acknowledgments

I would like to thank my Ph.D. advisor, Professor Renato José Sassi, for his support and guidance, and the Universidade Nove de Julho (UNINOVE) for the research grant CAPES PROSUP II.

References

[1] Michael Sikorski and Andrew Honig. *Practical malware analysis: the hands-on guide to dissecting malicious software*. no starch press, 2012.

[2] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *2012 IEEE symposium on security and privacy*, pages 95–109. IEEE, 2012.

[3] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM computing surveys (CSUR)*, 44(2):1–42, 2008.

[4] Nwokedi Idika and Aditya P Mathur. A survey of malware detection techniques. *Purdue University*, 48:2007–2, 2007.

[5] Xufang Li, Peter KK Loh, and Freddy Tan. Mechanisms of polymorphic and metamorphic viruses. In *2011 European intelligence and security informatics conference*, pages 149–154. IEEE, 2011.

[6] Fairuz Amalina Narudin, Ali Feizollah, Nor Badrul Anuar, and Abdullah Gani. Evaluation of machine learning classifiers for mobile malware detection. *Soft Computing*, 20(1):343–357, 2016.

[7] Zhiqiang Wang, Qian Liu, and Yaping Chi. Review of android malware detection based on deep learning. *IEEE Access*, 2020.

[8] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[9] TaeGuen Kim, BooJoong Kang, Mina Rho, Sakir Sezer, and Eul Gyu Im. A multimodal deep learning method for android malware detection using various features. *IEEE Transactions on Information Forensics and Security*, 14(3):773–788, 2018.

[10] Josh McGiff, William G Hatcher, James Nguyen, Wei Yu, Erik Blasch, and Chao Lu. Towards multimodal learning for android malware detection. In *2019 International Conference on Computing, Networking and Communications (ICNC)*, pages 432–436. IEEE, 2019.

[11] Balaji Vasu and Neelavathy Pari. Combining multimodal dnn and sigpid technique for detecting malicious android apps. In *2019 11th International Conference on Advanced Computing (ICoAC)*, pages 289–294. IEEE, 2019.

[12] Dali Zhu, Tong Xi, Pengfei Jing, Di Wu, Qing Xia, and Yiming Zhang. A transparent and multimodal malware detection method for android apps. In *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 51–60, 2019.

[13] N Amrutha and N Balagopal. Multimodal deep learning method for detection of malware in android using static and dynamic features. *CSI Journal of*, page 13, 2020.

[14] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. Multimodal deep learning. In *ICML*, 2011.

[15] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[17] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37–37, 1996.

[18] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[19] Alejandro Martín, Raúl Lara-Cabrera, and David Camacho. Android malware detection through hybrid features fusion and ensemble classifiers: the andropytool framework and the omnidroid dataset. *Information Fusion*, 52:128–142, 2019.

[20] Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, Guillermo Suarez-Tangil, and Steven Furnell. Androdialysis: Analysis of android intent effectiveness in malware detection. *computers & security*, 65:121–134, 2017.

[21] Fauzia Idrees and Muttukrishnan Rajarajan. Investigating the android intents and permissions for malware detection. In *2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 354–358. IEEE, 2014.

[22] Fauzia Idrees, Muttukrishnan Rajarajan, Thomas M Chen, Yogachandran Rahulamathavan, and Ayesha Naureen. Andropin: Correlating android permissions and intents for malware detection. In *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 394–399. IEEE, 2017.

[23] William Enck, Damien Octeau, Patrick D McDaniel, and Swarat Chaudhuri. A study of android application security. In *USENIX security symposium*, volume 2, page 2, 2011.

[24] TonTon Hsien-De Huang and Hung-Yu Kao. R2-d2: color-inspired convolutional neural network (cnn)-based android malware detections. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2633–2642. IEEE, 2018.

[25] Yuxin Ding, Xiao Zhang, Jieke Hu, and Wenting Xu. Android malware detection method based on bytecode image. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–10, 2020.

[26] Ken Turkowski. Filters for common resampling tasks. In *Graphics gems*, pages 147–165. Academic Press Professional, Inc., 1990.

[27] Xi Xiao, Shaofeng Zhang, Francesco Mercaldo, Guangwu Hu, and Arun Kumar Sangaiah. Android malware detection based on system call sequences and lstm. *Multimedia Tools and Applications*, 78(4):3979–3999, 2019.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[29] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[30] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[31] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[32] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[33] Sylvain Arlot, Alain Celisse, et al. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[35] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[36] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[37] Daniel Gibert, Carles Mateu, and Jordi Planes. Hydra: A multimodal deep learning framework for malware classification. *Computers & Security*, page 101873, 2020.

[38] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[39] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[41] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[42] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[43] Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'ıo, Mark Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[44] The pandas development team. *pandas-dev/pandas: Pandas*, February 2020.

[45] Marian Tietz, Thomas J. Fan, Daniel Nouri, Benjamin Bossan, and skorch Developers. *skorch: A scikit-learn compatible neural network library that wraps PyTorch*, July 2017.