Hunting Android Malware Using Multimodal Deep Learning and Hybrid Analysis Data

Angelo Schranko de Oliveira
PPGIGC
Universidade Nove de Julho
São Paulo, Brasil
ORCID 0000-0003-2933-9676

Renato José Sassi
PPGIGC
Universidade Nove de Julho
São Paulo, Brasil
ORCID 0000-0001-5276-4895

***Abstract*—In this work, we propose a new multimodal Deep Learning (DL) Android malware detection method, Chimera, that combines both manual and automatic feature engineering by using the DL architectures, Convolutional Neural Networks (CNN), feedforward Deep Neural Networks (DNN), and Transformer Networks (TN) to perform feature learning from raw data (Dalvik Executables (DEX)), 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. 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, and Recall outperform classical Machine Learning (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.**

***Index Terms*—Android Malware Detection, Computer Security, Multimodal Deep Learning.**

I. Introduction

Malware, or malicious software, is any software intentionally designed to cause harm to a computer, user, or network [1]. 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 [2]. Static Analysis provides a set of tools and techniques to understand how malware works without executing it [2]. 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 [2]. 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 [3]. The most common, faster, and simpler way of detecting malware is using signature-based methods [3]. Some disadvantages of signature-based methods are their inefficiency in detecting polymorphic and metamorphic malware [4], zero-day malware, and the high rate of false positives.

In order to increase the accuracy of malware detection and classification methods, several ML and DL methods have been proposed [5], [6]. 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 [7]. 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 [7]. 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 [8]–[12]. 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 [13]. 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 [7] to learn high-level feature representations from Android Intents & Permissions using an early fusion layer. (2) Chimera-Raw (Chimera-R), a CNN [14] to learn high-level feature representations from raw data transformed into DEX grayscale images, and (3) Chimera-Dynamic (Chimera-D), a TN encoder [15] 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

Coordination for the Improvement of Higher Education Personnel (CAPES)

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.

II. Related Work

The first Android malware detection method based on multimodal deep learning was introduced by [8]. 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. [9] introduced a multimodal DL method that implements an intermediate fusion layer composed of features extracted from Static Analysis data: Permissions and Hardware Features. The authors evaluated the performance of using each modality separately and in the multimodal setting. [10] 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. [11] 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. [12] 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.

III. Proposed Method and Methodology

This work follows the Knowledge Discovery in Databases (KDD) process [16] and Supervised ML methodology [17] for model selection, training, and evaluation. KDD is an iterative, interactive, and non-trivial process composed of several stages for extracting (useful) patterns from large databases. Figure 5 depicts the KDD process implementation for Chimera. 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. 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 main advantage of Chimera over the other methods is the use of three different data modalities, i.e., raw data, static analysis data, and dynamic analysis data, to tune, train, and evaluate the model and each one of its subnetworks. Moreover, each subnetwork implements a specialized architecture designed for its data modality in order to perform better feature learning, and therefore improving the method's performance.

A. Selection, Preprocessing and Transformation

In this work, we used the Android benchmark dataset Omnidroid, introduced by [18]. Omnidroid is a balanced dataset composed of pre-static, static, and dynamic analysis information extracted from 22,000 real malware and benign Android applications. We also downloaded the APK files required for Chimera-R from the online malware repositories Androzoo[1].

1) Chimera-S: Inspired by the work presented in [19]–[21], Chimera-S and Chimera implement an early fusion layer to combine both Android Intents and Android Permissions for malware detection. 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 [17].

2) Chimera-R: DEX files contain the executable code in Dalvik (bytecode) format. The methods proposed by [22], [23] 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. The Transformation stage was performed by resampling the data to image representations of 1x128x128 pixels (channel, width, height) using the Lanczos resampling algorithm [24], and by applying Scaling and Standardization procedures [17].

3) Chimera-D: System call sequences represent the application's interaction with the hardware by calling low-level functions exposed by the OS. Figure 4 depicts the system call sequences of benign applications and malware overlapped. Based on the work introduced by [25], Chimera-D and Chimera also depend on data collected using Dynamic

[1] https://androzoo.uni.lu/

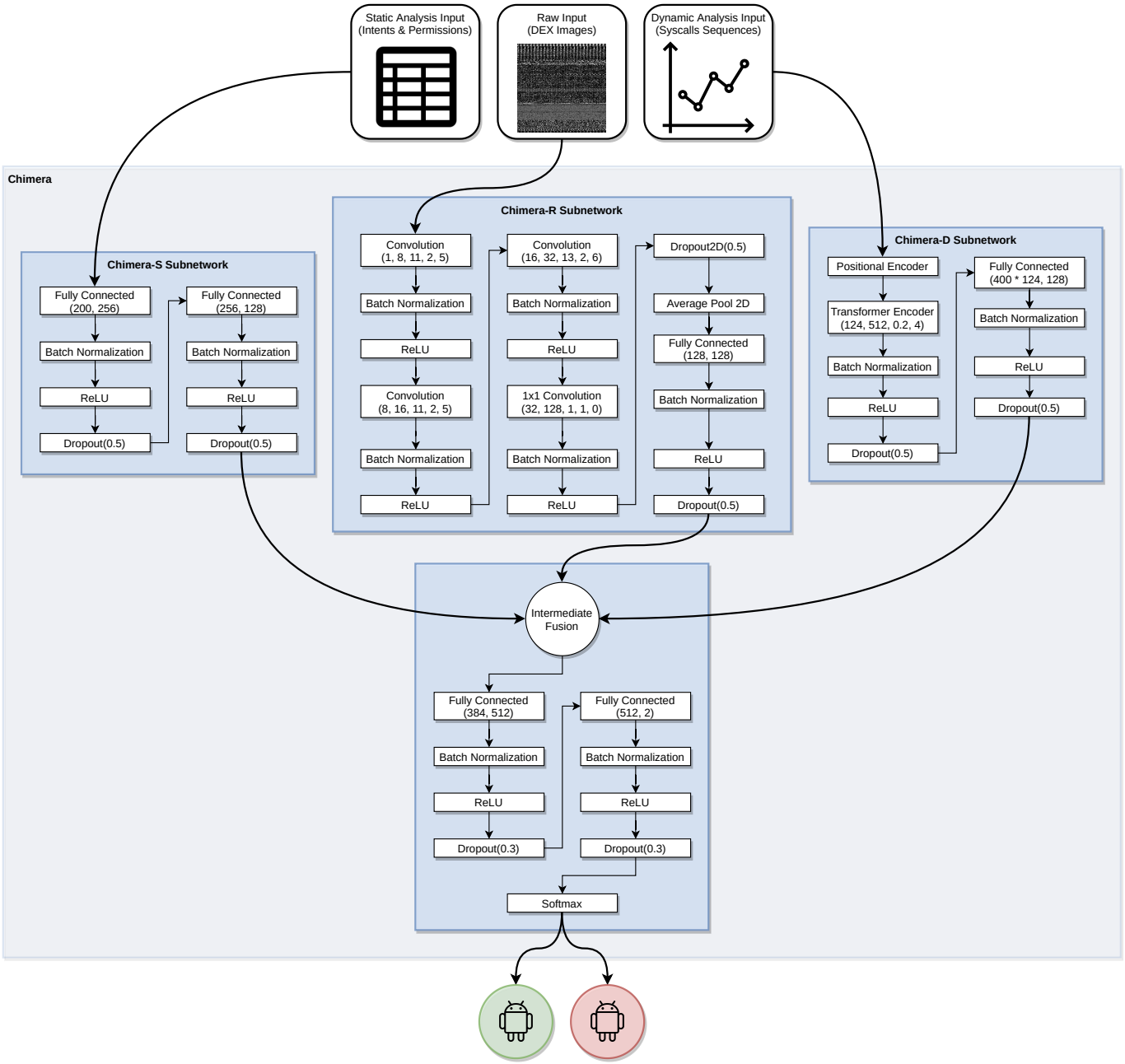

Fig. 1: Chimera Android malware detection method architecture.

Analysis, particularly the system call sequences. 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. 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 [17].

B. Data Mining and Interpretation

In order to choose the best architectures for Chimera-S, Chimera-R, Chimera-D, and Chimera, we performed model selection using grid search cross-validation with 10-fold cross-validation to estimate the generalization error [26] and the Accuracy metric (See Equation 1) to guide the hyperparameter tuning process.

1) Chimera-S: As depicted in Figure 2, Chimera-S introduces a DNN architecture [27] 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

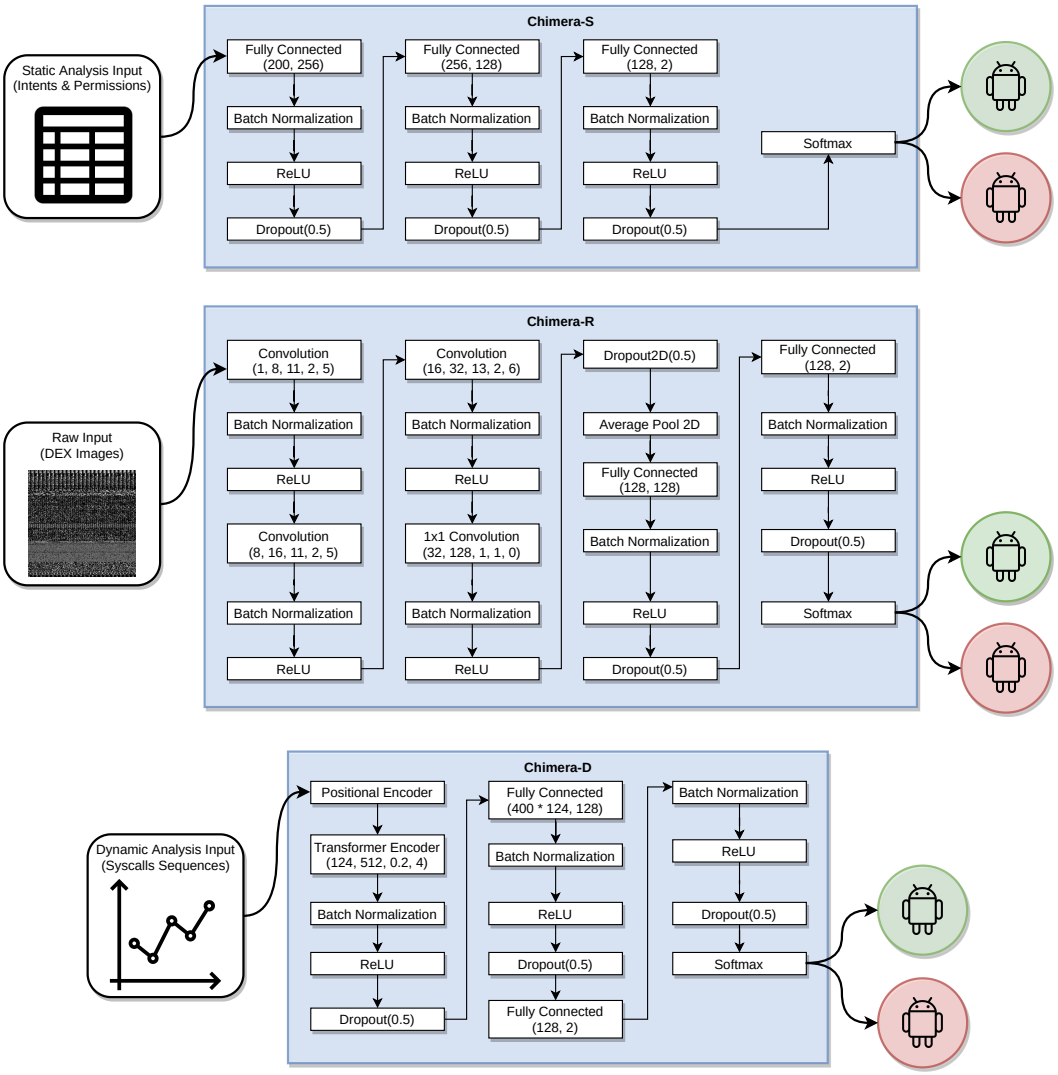

Fig. 2: Chimera's Android malware detection method DL subnetworks for Static Analysis data (Chimera-S), DEX grayscale images (Chimera-R), and Dynamic Analysis data (Chimera-D).

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. 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 1st/2nd hidden layer: 256/128
- Dropout probability: 0.5
- Number of epochs: 50
- Learning rate step decay schedule factor/steps: 0.5/10

2) Chimera-R: As we can see in Figure 3, the second 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 first 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, [22] proposed a CNN architecture for Android malware classification using DEX grayscale images, and [23] introduced a CNN architecture for Android malware detection using DEX opcodes translated to RGB images.

Our work follows a similar approach proposed by [22], [23] and introduces a new CNN architecture inspired by the Residual Networks (ResNet) architecture [28]. As we can see in Figure 2, 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 [27]. The Global Average Pooling operation [27] 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. From this point

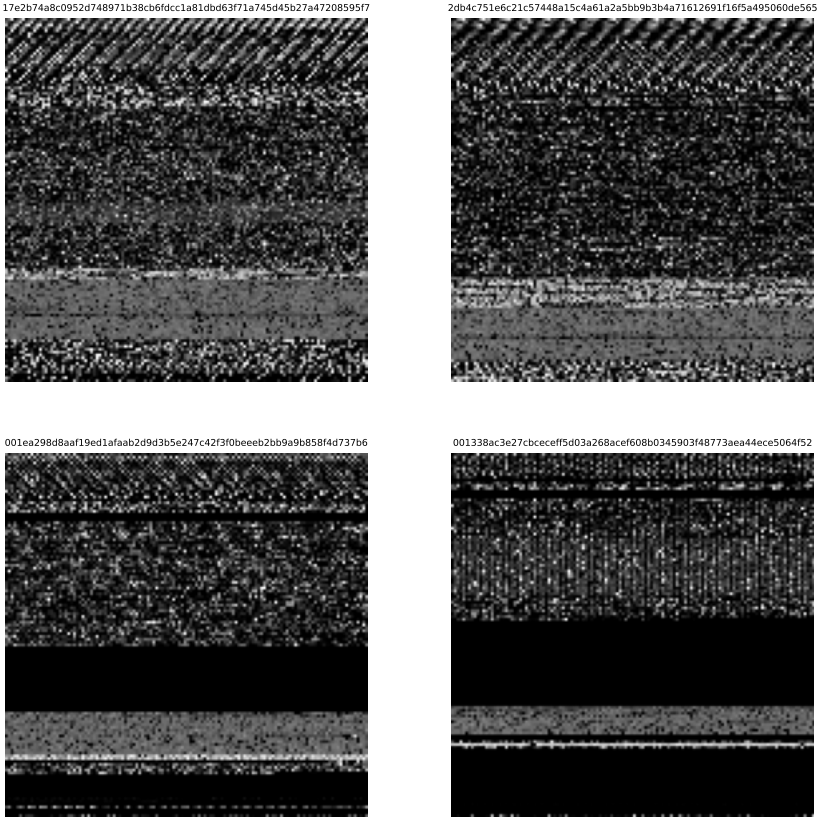

Fig. 3: DEX grayscale images of two benign applications in the first row and two Trojan malware in the second row, including the SHA256 hash of each instance.

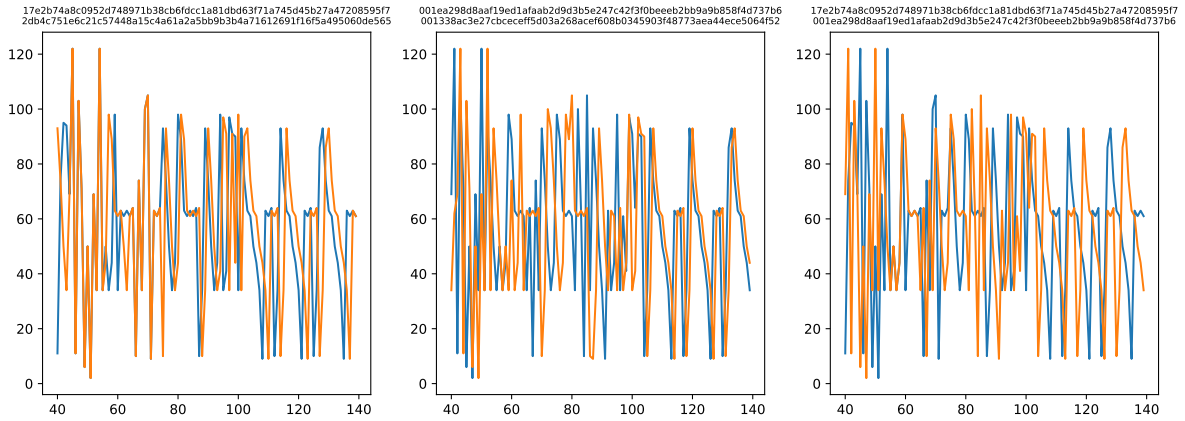

Fig. 4: System call sequences of two benign applications in the first column, two Trojan malware in the second column, and one benign application overlapped with one Trojan malware in the third column. The titles include the SHA256 hash of each instance. The x-axis represents the time step. The y-axis represents the system call number.

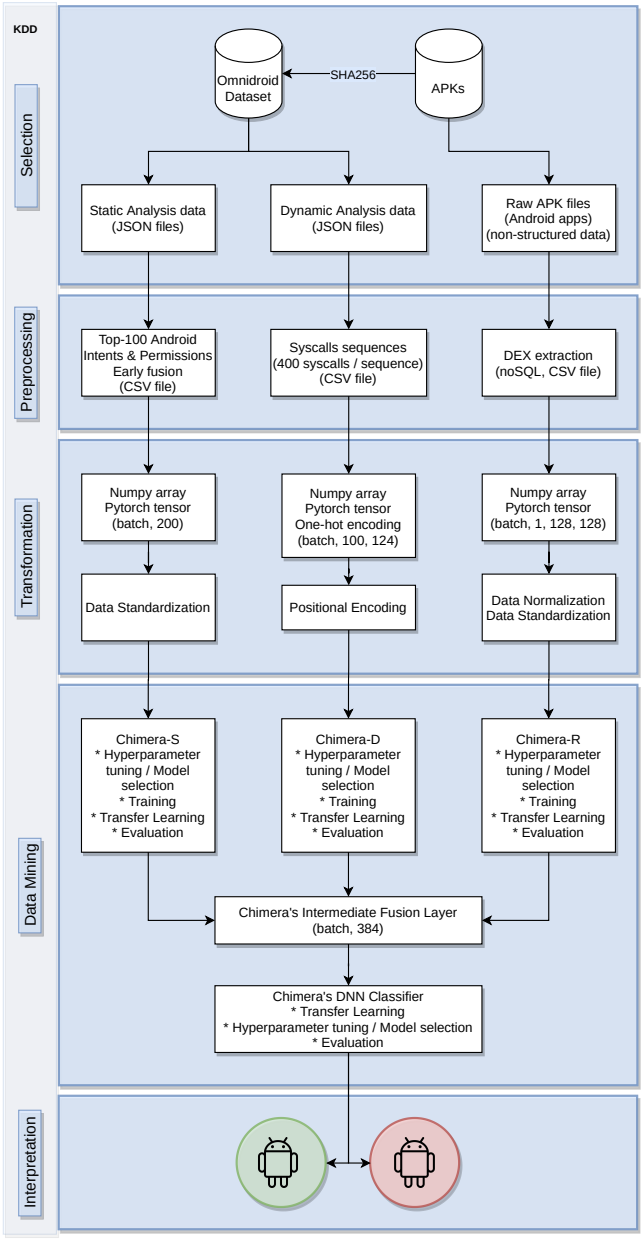

Fig. 5: KDD process stages of Chimera Android malware detection method. The blue boxes represent each KDD process stage. The white boxes represent the implementation steps for each stage.

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 mitigate overfitting, we included Dropout and Batch Normalization layers. 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/2nd/3rd convolutional layer: 11/11/13
- Dropout probability: 0.5
- Number of epochs: 40
- Learning rate step decay schedule factor/steps: 0.1/10

3) Chimera-D: As we can see in Figure 4, the second 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 first 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 addition, the third column depicts a malware instance overlapped with a benign instance, and indicates a high negative correlation between them, which can be leveraged by the model for detection and classification purposes. In fact, [25] proposed an LSTM architecture to implement a neural probabilistic language model for Android malware detection using system call sequences.

Our work is based on a different architecture for sequence learning: the Transformer Networks (TN) [15]. TN is a state-of-the-art encoder-decoder DL architecture designed to handle sequential data, such as natural language. As we can see in Figure 2, 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. 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. 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. 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 [15]. 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/DNN layer: 512/128
- TN/DNN Dropout probability: 0.2/0.5
- Number of epochs: 30
- Learning rate warm-up schedule factor/steps: 1.033/1

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 feature-wise concatenation of the results and passed to the last Chimera's DNN classifier. Similar to what was verified by [8], [29], 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 III-B1, III-B2, and III-B3 respectively, and used Transfer Learning [30] to combine them into the final Chimera architecture. 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. 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/steps: 0.5/10

IV. Performance Evaluation and Discussion

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

1) Classical ML algorithms [17]: 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).
2) State-of-the-art Ensemble ML algorithms [31]: AdaBoost, Bagging, Extra Trees, Gradient Boosting, and Random Forest.
3) Chimera's DL subnetworks: Chimera-S, Chimera-R, and Chimera-D. See Figure 2.
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 [32]. In addition, we also compared Chimera's performance results to the state-of-the-art Voting Ensemble ML method results presented in [18].

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. 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. In the context of

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 I: 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.

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 II: 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.

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 III: 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.

malware detection methods, false negatives pose a much more significant threat to the users than false positives.

As we can see in Tables I, II, and III, 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 the correlation between multiple data modalities. Moreover, it also takes advantage of automatic feature engineering by using DL architectures and manual feature engineering applied. It is important to notice that Chimera achieved higher performance than its subnetworks evaluated independently as we can see in Tables I, II, and III. 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 I, Chimera-S achieved fourth place, showing better performance than all the classical ML algorithms and some of the Ensemble ML algorithms. As presented in Table II, Chimera-R achieved second place, showing better performance than all the classical and Ensemble ML algorithms. 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. As presented in Table III, Chimera-D achieved fifth place, showing better performance than all the classical ML algorithms and some Ensemble ML algorithms. Surprisingly enough, LSTM networks, which are specialized DL architectures for sequence learning, showed poor performance. The results presented in Table III 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 (III-A3). The results presented in Table III clearly show the main 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 [18]. The Voting classifier includes a Random Forest classifier for the static features and a Bagging classifier for the dynamic features. 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 [33] and the numpy stack [34] for the implementations.

A. Limitations and Future Work

Chimera achieved the best performance for all the considered metrics except the Fit Time. The main reason for that is the requirement of training of each subnetwork independently in addition to a final training session for the intermediate fusion layer using the features learned by each one of the subnetworks. One possible attempt for improvement is the simplification of the each subnetwork by reducing the number of parameters and layers, while keeping similar performance. Another promising attempt for improvement is the addition of learnable parameters associated to each subnetwork to weight their importance in order to mitigate the training problem described in III-B4, thus decreasing the Fit Time. Future work can investigate these possibilities.

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 can be extended to malware multiclass classification in future work.

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 can explore DL interpretability methods and how to apply them to Chimera.

Finally, both Windows malware and malicious documents also contain or generate raw data, static analysis data, and dynamic analysis data. Taking that into account, future work can research the applicability of Chimera and its subnetworks to those classes of malware.

V. 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 method 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, state-of-the-art Ensemble ML algorithms, the Chimera's DL subnetworks Chimera-S, Chimera-R, and Chimera-D, and the state-of-the-art Voting Ensemble method proposed by the creators of the Omnidroid dataset. Future work will tackle the training challenges and expand the method's applicability and usability.

References

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

[2] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM computing surveys (CSUR)*, vol. 44, no. 2, pp. 1–42, 2008.

[3] N. Idika and A. P. Mathur, "A survey of malware detection techniques," *Purdue University*, vol. 48, pp. 2007–2, 2007.

[4] X. Li, P. K. Loh, and F. Tan, "Mechanisms of polymorphic and metamorphic viruses," in *2011 European intelligence and security informatics conference*. IEEE, 2011, pp. 149–154.

[5] F. A. Narudin, A. Feizollah, N. B. Anuar, and A. Gani, "Evaluation of machine learning classifiers for mobile malware detection," *Soft Computing*, vol. 20, no. 1, pp. 343–357, 2016.

[6] Z. Wang, Q. Liu, and Y. Chi, "Review of android malware detection based on deep learning," *IEEE Access*, 2020.

[7] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[8] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for android malware detection using various features," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 773–788, 2018.

[9] J. McGiff, W. G. Hatcher, J. Nguyen, W. Yu, E. Blasch, and C. Lu, "Towards multimodal learning for android malware detection," in *2019 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2019, pp. 432–436.

[10] B. Vasu and N. Pari, "Combining multimodal dnn and sigpid technique for detecting malicious android apps," in *2019 11th International Conference on Advanced Computing (ICoAC)*. IEEE, 2019, pp. 289–294.

[11] D. Zhu, T. Xi, P. Jing, D. Wu, Q. Xia, and Y. 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*, 2019, pp. 51–60.

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

[13] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *ICML*, 2011.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[16] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From data mining to knowledge discovery in databases," *AI magazine*, vol. 17, no. 3, pp. 37–37, 1996.

[17] E. Alpaydin, *Introduction to machine learning*. MIT press, 2020.

[18] A. Martín, R. Lara-Cabrera, and D. Camacho, "Android malware detection through hybrid features fusion and ensemble classifiers: the andropytool framework and the omnidroid dataset," *Information Fusion*, vol. 52, pp. 128–142, 2019.

[19] A. Feizollah, N. B. Anuar, R. Salleh, G. Suarez-Tangil, and S. Furnell, "Androdialysis: Analysis of android intent effectiveness in malware detection," *computers & security*, vol. 65, pp. 121–134, 2017.

[20] F. Idrees and M. 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)*. IEEE, 2014, pp. 354–358.

[21] F. Idrees, M. Rajarajan, T. M. Chen, Y. Rahulamathavan, and A. Naureen, "Andropin: Correlating android permissions and intents for malware detection," in *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 2017, pp. 394–399.

[22] T. Hsien-De Huang and H.-Y. Kao, "R2-d2: color-inspired convolutional neural network (cnn)-based android malware detections," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 2633–2642.

[23] Y. Ding, X. Zhang, J. Hu, and W. Xu, "Android malware detection method based on bytecode image," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–10, 2020.

[24] K. Turkowski, "Filters for common resampling tasks," in *Graphics gems*. Academic Press Professional, Inc., 1990, pp. 147–165.

[25] X. Xiao, S. Zhang, F. Mercaldo, G. Hu, and A. K. Sangaiah, "Android malware detection based on system call sequences and lstm," *Multimedia Tools and Applications*, vol. 78, no. 4, pp. 3979–3999, 2019.

[26] S. Arlot, A. Celisse *et al.*, "A survey of cross-validation procedures for model selection," *Statistics surveys*, vol. 4, pp. 40–79, 2010.

[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[29] D. Gibert, C. Mateu, and J. Planes, "Hydra: A multimodal deep learning framework for malware classification," *Computers & Security*, p. 101873, 2020.

[30] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[31] Z.-H. Zhou, *Ensemble methods: foundations and algorithms*. CRC press, 2012.

[32] 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*, vol. 12, pp. 2825–2830, 2011.

[33] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[34] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2