UNIVERSIDADE NOVE DE JULHO - UNINOVE
PROGRAMA DE PÓS GRADUAÇÃO EM INFORMÁTICA E GESTÃO DO CONHECIMENTO - PPGIGC

ANGELO SCHRANKO DE OLIVEIRA

A NEW ANDROID MALWARE DETECTION METHOD BASED ON MULTIMODAL DEEP LEARNING AND HYBRID ANALYSIS

São Paulo

2022

ANGELO SCHRANKO DE OLIVEIRA

A NEW ANDROID MALWARE DETECTION METHOD BASED ON MULTIMODAL DEEP LEARNING AND HYBRID ANALYSIS

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Informática e Gestão do Conhecimento - PPGIGC da Universidade Nove de Julho - UNINOVE, como requisito parcial para obtenção do título de Doutor em Informática e Gestão do Conhecimento.

Orientador: Prof. Dr. Renato José Sassi

São Paulo

2022

Oliveira, Angelo Schranko de.
A new android malware detection method based on multimodal deep learning and hybrid analysis. / Angelo Schranko de Oliveira. 2022.
95 f.

Tese (Doutorado) - Universidade Nove de Julho - UNINOVE, São Paulo, 2022.
Orientador (a): Prof. Dr. Renato José Sassi.

1. Android malware detection. 2. Multimodal deep learning. 3. Computer security.
I. Sassi, Renato José. II. Título.
CDU 004

UNINOVE
Universidade Nove de Julho

PARECER – EXAME DE DEFESA

Parecer da Comissão Examinadora designada para o exame de defesa do Programa de Pós-Graduação em Informática e Gestão do Conhecimento, a qual se submeteu o aluno regularmente matriculado Angelo Schranko de Oliveira.

Tendo examinado o trabalho apresentado para obtenção do título de "Doutor em Informática e Gestão do Conhecimento", com Tese intitulada "A NEW ANDROID MALWARE DETECTION METHOD BASED ON MULTIMODAL DEEP LEARNING AND HYBRID ANALYSIS", a Comissão Examinadora considerou o trabalho:

(X) Aprovado	**( ) Aprovado condicionalmente**
( ) Reprovado com direito a novo exame	**( ) Reprovado**

Parecer: Aprovado. A comissão examinadora contemplou a tese de doutorado com Menção Honrosa.

EXAMINADORES

Prof. Dr. Renato José Sassi (Orientador – PPGI/UNINOVE)

[illegible]

Prof. Dr. Fabio Silva Lopes (Participante externo – Mackenzie)

Fabio S. Lopes

Prof. Dr. Leandro Augusto da Silva (Participante Externo – Mackenzie)

[illegible]

Prof. Dr. Cleber Gustavo Dias (Participante Interno – PPGI/UNINOVE)

[illegible]

Prof. Dr. Fellipe Silva Martins (Membro Interno – PPGI/UNINOVE)

[illegible]

São Paulo, 17 de março de 2022.

DEDICATION

I dedicate this doctoral dissertation to:

My dear mother, Expedita Maria de Oliveira.

The memory of my father, Edson Schranko de Oliveira.

My beloved wife, Svetlana Atiunina Schranko.

My feline daughters, Misty, Sophie, and Misha.

DEDICATÓRIA

Eu dedico esta tese de doutorado para:

Minha querida mãe, Expedita Maria de Oliveira.

Em memória ao meu pai, Edson Schranko de Oliveira.

Minha amada esposa, Svetlana Atiunina Schranko.

Minhas filhas felinas, Misty, Sophie e Misha.

ACKNOWLEDGEMENTS

Thanks to everyone who helped directly or indirectly throughout this journey.

Thanks to Universidade Nove de Julho (UNINOVE) and the Coordination for the Improvement of Higher Education Personnel (CAPES) for the research grants, PPGIGC Doctoral Scholarship and CAPES PROSUP-II.

Thanks to the professors of the Examination Committees for all the invaluable contributions to this dissertation.

Special thanks to my advisor, Prof. Dr. Renato José Sassi, for all his guidance, support, friendship, and patience.

RESUMO

No mundo atual, onde quase toda a informação está digitalizada, o cibercrime está em ascensão e os criminosos continuam a desenvolver novas maneiras de explorar sistemas de informação. Uma das principais ferramentas utilizadas para operações de crimes cibernéticos são malware ou softwares maliciosos. A detecção de malware é um problema desafiador que tem sido ativamente explorado tanto pela indústria quanto pela academia utilizando para isso métodos inteligentes. Por um lado, a detecção de malware utilizando técnicas tradicionais de aprendizado de máquina depende da engenharia manual de variáveis, o que requer conhecimento especializado na área. Por outro lado, os métodos de detecção de malware utilizando aprendizado profundo possibilitam a extração automática de variáveis, mas geralmente exigem muito mais dados e poder de processamento. Além disso, existem várias modalidades de dados oriundos da análise de malware que podem ser utilizados para fins de detecção. Assim, o objetivo geral desta tese foi desenvolver e avaliar um novo método de detecção de malware Android, chamado Chimera, baseado em aprendizado produndo multimodal e análise híbrida, utilizando diferentes modalidades de dados e combinando engenharia de variáveis manual e automática para para aumentar a taxa de detecção de malware Android. Com o objetivo de treinar, otimizar e avaliar os modelos, o processo de Descoberta de Conhecimento em Bancos de Dados foi implementado utilizando a base de dados Omnidroid, publicamente disponível contendo dados de análise estática e dinâmica extraídos de 22.000 amostras reais de malware e goodware. Através de uma fonte híbrida de informações para aprender representações de alto nível de variáveis para ambas as propriedades estáticas e dinâmicas de aplicativos Android, o desempenho do Chimera superou suas sub-redes unimodais, métodos de aprendizado de máquina clássicos e métodos de aprendizado de máquina ensemble, portanto, os resultados desta tese mostram que a combinação correta de dados multimodais, métodos de aprendizagem profunda especializados, e engenharia de variáveis manual e automática podem aumentar significativamente a taxa de detecção de malware Android.

Palavras-chave: Detecção de malware Android, Aprendizado profundo multimodal, Segurança cibernética

ABSTRACT

In the current world, whereby almost everything is digitized, cybercrime is on the rise as criminals continue to develop new ways to hack information systems. One of main tools used for cybercrime operations are malware, or malicious software. 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 learning but usually require much more data and processing power. Moreover, there are multiple data modalities of Malware Analysis (MA) data that can be used for detection purposes. Thus, the general objective of this dissertation was to develop and evaluate a new Android malware detection method, named Chimera, based on Multimodal Deep Learning (MDL) and Hybrid Analysis (HA), using different data modalities and combining both manual and automatic feature engineering in order to increase Android malware detection rate. To train, optimize, and evaluate the models, the Knowledge Discovery in Databases (KDD) process was implemented using a new dataset based on the publicly available Android benchmark dataset Omnidroid containing Static Analysis (SA) and Dynamic Analysis (DA) data extracted from 22000 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 performance outperformed its unimodal DL subnetworks, classical ML methods, and Ensemble ML methods, thus, the results of this dissertation show that the right combination of multimodal data, specialized DL methods, manual and automatic feature engineering can significantly increase Android malware detection rate.

Keywords: Android Malware Detection, Multimodal Deep Learning, Computer Security

Contents

List of Acronyms

ANN	Artificial Neural Network
APK	Android Application Package
CHD	Chimera-D (Chimera Dynamic)
CHR	Chimera-R (Chimera Raw)
CHS	Chimera-S (Chimera Static)
CNN	Convolutional Neural Network
CSV	Comma Separated Values
DA	Dynamic Analysis
DEX	Dalvik Executable
DL	Deep Learning
DNN	Feedforward Deep Neural Network
HA	Hybrid Analysis
JSON	Javascript Object Notation
KDD	Knowledge Discovery in Databases
KNN	K-Nearest Neighbors
LSTM	Long-short Term Memory Network
MA	Malware Analysis
MDL	Multimodal Deep Learning
ML	Machine Learning
MLP	Multilayer Perceptron
NLP	Natural Language Processing
RBF	Radial Basis Function
ROC AUC	Area under the Receiver Operating Characteristic Curve
SA	Static Analysis
SVM	Support Vector Machines
TNN	Transformer Neural Network
VM	Virtual Machine

List of Figures

CHAPTER 1

INTRODUCTION

Cybersecurity protects industry and governmental information systems, personally identifiable information, intellectual property, sensitive information, and protected health information from damage or theft. In the current world, whereby almost everything is digitized, cybercrime is on the rise as criminals continue to develop new ways to hack information systems. It is impossible for an organization to defend itself against such occurrences if it lacks a proper cybersecurity program (ANI; HE; TIWARI, 2017). The latter also helps in tracking and capturing cybercriminals. Breach risks continue to increase as global connectivity is enhanced. Sensitive information can now be stored in cloud services and bank transactions are conducted over the Internet. As cybercriminals become more sophisticated and cloud services are configured poorly, organizations are bound to be victims of serious cyber attacks. Initially, businesses used basic cybersecurity solutions such as firewalls and antivirus software to prevent attacks or threats. However, sophistication in cybercrime has forced organizations to develop stringent measures that are developed in a comprehensive cybersecurity program (ANI; HE; TIWARI, 2017). Apart from having cybersecurity professionals, other employees should be trained about different malware attacks and how to respond to them or escalate them.

Malware, or malicious software, is any software intentionally designed to cause harm to a computer, user, or network (SIKORSKI; HONIG, 2012). It is designed specifically for such functions, and different cybercriminals use them according to the crime they intend to commit.

To understand malware internals and behavior, one can make use of Malware Analisys (MA) techniques. MA introduces a set of techniques that can be used to dissect malware and understand how it works as a means to identify and defeat it (SIKORSKI; HONIG, 2012). It is based on a subset of techniques known as Static Analysis (SA), Dynamic Analysis (DA), and Hybrid Analysis (HA) (EGELE et al., 2008). On the one hand, SA provides a set of tools and techniques to understand how malware works without executing it (EGELE et al., 2008). On the other hand, DA provides a set of tools and techniques to understand how malware works by executing it in a controlled, isolated environment known as sandbox (EGELE et al., 2008). HA combines both SA and DA to understand malware effectively by taking advantage of both SA and DA resources. The information gathered using MA can be leveraged for malware detection and classification tasks (IDIKA; MATHUR, 2007).

The most common, faster, and simpler way of detecting malware is using signature-based methods (IDIKA; MATHUR, 2007). Signature-based methods rely on the extraction of malicious patterns from known malware using MA techniques. Once the malicious patterns are collected, their presence or absence can be quickly verified in the suspicious files; However, signature-based methods present a high rate of false negatives, specially when dealing with polymorphic and metamorphic malware, which are malware that take advantage of advanced obfuscation and encryption techniques to avoid detection (LI; LOH; TAN, 2011).

With the purpose of increasing the accuracy of malware detection and classification methods, several Machine Learning (ML) and Deep Learning (DL) methods have been proposed (NARUDIN et al., 2016; WANG; LIU; CHI, 2020). In general, ML malware detection methods have the capability of learning malicious patterns from data, i.e. 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. DL malware detection methods leverage specialized architectures designed for image processing, speech recognition, sequence learning, and so on (LECUN; BENGIO; HINTON, 2015). Contrary to traditional ML methods, DL methods have the capability of performing automatic feature learning from structured and unstructured data of different domains, thus decreasing the amount of work associated with manual feature engineering (LECUN; BENGIO; HINTON, 2015). In fact, DL methods have achieved state-of-the-art results in image recognition tasks, speech recognition, and natural language processing (NLP) (LECUN; BENGIO; HINTON, 2015). Hovewer, DL methods usually require large volumes of data and processing power to achieve higher accuracy.

More recently, Android malware detection methods using Multimodal Deep Learning (MDL) have also been proposed (KIM et al., 2019; MCGIFF et al., 2019; VASU; PARI, 2019; ZHU et al., 2019; AMRUTHA; BALAGOPAL, 2020). MDL uses independent, specialized DL subnetworks to extract high-level feature representations from different data modalities and combines the resulting vectors into a shared representation that can be used for classification and regression tasks, achieving even higher accuracy than traditional ML methods and unimodal DL methods (NGIAM et al., 2011). Taking that into account, the development and evaluation of new Android malware detection methods based on MDL and HA, using different data modalities and combining both manual and automatic feature engineering in order to increase Android malware detection rate can protect even more corporations and end users.

1.1 Justification and Motivation

Android malware poses a significant threat to both end-users and corporations (SUAREZ-TANGIL; STRINGHINI, 2020). While end-users might have their personal and financial data stolen or encrypted, corporations might have their security perimeters breached and the whole network compromised. This scenario is even harder to avoid when employees use their personal smartphones, or any other Android device, inside the company's perimeter. If their devices are infected with malware and a connection to the internal corporate wireless network takes place, malware instances might propagate in the intranet, infecting servers and others employees' devices more easily and quickly.

According to the Cybercrime Magazine (VENTURES, 2021), Cybersecurity Ventures expects cybercrime to cost the world USD 10,5 Trillion per year by 2025. An amount larger than the damaged caused by natural disasters. Cybercriminals typically use malware to extract data that they can leverage over victims for financial gain. From financial data, such as credit card numbers to healthcare records, to personal emails, credentials, and passwords.

Another kind of malware that has been gaining momentum is called Ransomware (HUMAYUN et al., 2021; ROŠKOT; WANASIKA; KROUPOVA, 2020), which is a type of malware that prevents users from accessing their personal files by encrypting them and demanding ransom payment to reveal the decryption key. Ransomware in corporate environments can shutdown the entire business operation. As as example, a recent Ransomware incident in the Brazilian scenario (CANALTECH, 2021) asked for more than R$ 1,0 Billion after a successful database encryption attack. Although Ransomware is less common on Android platforms, Android devices can be used as a gateway to infect the target platforms inside the perimeter. It is therefore clear that Android malware are an essential tool for cybercrime operations and their early detection and removal can avoid costly security incidents and damage to the society.

Dealing with the rapid increase in number, variability, and complexity of malware requires the research and development of new intelligent and automatic malware detection methods that can leverage the information collected by various monitoring systems in the network at scale. Consequently, it is necessary to develop new methods capable of correlating different types of information such as spatial, temporal and relational data aiming to improve the detection accuracy.

1.2 Research Gap

Cybersecurity and Data Science professionals are exposed to terabytes of information generated from several cybersecurity monitoring tools centralized in the Security Information and Event Management (SIEM) software. Since that information is generated by different data sources and under different conditions, its nature is multimodal. Multimodal Deep Learning (MDL) 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 (NGIAM et al., 2011). For example, in MDL, it is possible to combine 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 (NGIAM et al., 2011). Leveraging MDL for the development of efficient methods that can make use that information for malware detection purposes can result in direct benefits for companies and for the end users. Moreover, the development of new MDL Android malware detection methods that combine multiple data modalities and uses different DL subnetworks for feature learning also contributes to the academic knowledge due to the necessity of the development of novel techniques and algorithms.

The research gap explored in this dissertation is the use of a new combination of different data modalities and specialized DL subnetworks to develop a MDL Android malware detection method with superior performance. As detailed in Section 2.6, the main technical novelty of this dissertation is to develop and evaluate a new MDL Android malware detection method that makes use of the three distinct data modalities, i.e., raw data, SA data, and DA data, represented by DEX grayscale images, Android Intents & Permissions, and system call sequences respectively, to train, optimize, and evaluate the model and each one of its subnetworks, on top of the use of specialized subnetwork architectures designed for efficient feature learning from each data modality, and therefore improving even more the method's detection rate compared to traditional ML methods, unimodal deep learning methods, and MDL methods. Moreover, in an effort to develop the new method, a new multimodal dataset was build (Section 3.3) and an implementation of the Knowledge Discovery in Databases (KDD) process (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996) was also proposed.

1.3 Research Problem and Question

The literature review in Section 2.6 conducted as a part of this dissertation revealed that several Android malware detection methods using MDL have been recently proposed (KIM et al., 2019; VASU; PARI, 2019; ZHU et al., 2019; JIMÉNEZ; GOSEVA-POPSTOJANOVA, 2020); However, none of them makes use of the same distinct data modalities, DL architectures, and the KDD implementation proposed in this dissertation. Thus, the research question this dissertation seeks to find an answer to is the following: How the development and evaluation of a new Android malware detection method, based on MDL and HA, using different data modalities and combining both manual and automatic feature engineering, can increase Android malware detection rate?

1.4 Objectives

1.4.1 General

The general objective of this dissertation was to develop and evaluate a new Android malware detection method, named Chimera, based on MDL and HA, using different data modalities and combining both manual and automatic feature engineering in order to increase Android malware detection rate.

1.4.2 Specifics

The specific objetives of this dissertation are the following:

1. To build a dataset containing HA data collected from the Omnidroid dataset (MARTÍN; LARA-CABRERA; CAMACHO, 2019) and Android DEX images extracted from Android malware downloaded from the online repositories (ANDROZOO, 2021) and (KOODOUS, 2021).

2. To implement the KDD process for feature selection, data preprocessing, and data transformation.

3. To identify the best DL architectures for Chimera's subnetworks by using hyperparameter tuning/model selection strategies.

4. To determine the best training strategy for Chimera: End-to-end training or Transfer Learning.

5. To assess the performance of Chimera and compare it to the following classes of intelligent methods using the Accuracy, Precision, Recall, and ROC AUC metrics:

    (a) Unimodal Chimera's DL subnetworks, i.e., Chimera-S (CHS), Chimera-R (CHR), and Chimera-D (CHD).

    (b) Classical ML methods (ALPAYDIN, 2020): Support Vector Machines (SVM), Logistic Regression, Multilayer Perceptron (MLP), Decision Tree, Naive Bayes, and K-Nearest Neighbors (KNN).

    (c) State-of-the-art Ensemble ML methods (ZHOU, 2012): Extra Trees and Random Forest.

1.5 Research Scope and Delimitation

Although Windows and Linux platforms are widely spread, following the reasons presented in Section 1.1 this dissertation takes into consideration the problem of detecting malware on the Android platform only. Moreover, Chimera is by definition a binary classifier designed for Android malware detection, therefore this dissertation considers only the class of malicious applications, or malware, and the class of benign or normal applications. Thus, classifying to which malware family a malware instance belongs to is out of the scope of this dissertation. Finally, the dataset used for training and evaluation of all the ML, DL and MDL methods present in this dissertation was built based solely on the Omnidroid dataset and augmented with the DEX grayscale images extracted from the online repositories (ANDROZOO, 2021) and (KOODOUS, 2021).

1.6 Document Structure

This dissertation is organized as follows: Chapter 1 presents the Introduction, Justification and Motivation, Research Gap, Research Problem and Question, Objectives, and Research Scope and Delimitation. Chapter 2 introduces the Theoretical Background necessary to understand the proposed method. It includes subsections on the Information Security and Cybersecurity, Android Platform, MA, KDD, and DL techniques. In addition, it provides a Literature Review on MDL methods for Android malware detection. Chapter 3 details the Material and Methods adopted in the research, including the Metodology, Proposed Method, Datasets and Experimental Setup, and the proposed implementation of the KDD for Chimera's training, optimization and evaluation. Chapter 4 presents the Results and Discussion, including performance comparisons and analyses. Finally, Chapter 5 summarizes conclusions, contributions, and limitations of this dissertation.

Chapter 2

Theoretical Background

This chapter introduces the fundamental concepts in which this dissertation is based on. The first three sections are related to Information Security concepts. Section 2.1, introduces the fields of Information Security and Cybersecurity. Next, Section 2.2 presents the concepts of the Android security architecture. Then, Section 2.3 explains the concepts of Malware Analysis (MA), Static Analysis (SA), Dynamic Analysis (DA), and Hybrid Analysis (HA). The next two sections are related to knowledge discovery and intelligent methods. First, Section 2.4 introduces the KDD process. Then, Section 2.5 summarizes the DL techniques used in this dissertation. Finally, Section 2.6 provides a literature review on MDL methods for Android malware detection.

2.1 Information Security and Cybersecurity

Information security refers to the principles, policies, and practices involved in the protection of digital data (DAWSON; THOMSON, 2018). Therefore, information assets are protected by established and monitored processes. Digital data can be compromised while being transferred or when it is stored. When that happens, an organization can suffer serious consequences such as loss of money, bad reputation, or it can lose its competitive advantage.

An overall cybersecurity program is highly critical to avert or mitigate such risks (DAWSON; THOMSON, 2018). Therefore, information security is a form of a risk management protocol. When data is secure, an organization can minimize negative outcomes while maximizing positive ones. The three pillars of information security are confidentiality, integrity, and availability (DAWSON; THOMSON, 2018).

Confidentiality refers to the ability to keep valuable information private. Therefore, data is encrypted to ensure unauthorized people do not access the information. Confidential data include the accounts of the company, information about the organization's partnership deals, competitive advantage information, and other valuable assets (DAWSON; THOMSON, 2018). Authorized parties are those who are directly involved in the private processes or assets. Confidential information must not be exposed to people who do not require them because they can use it to destroy the organization's reputation, steal money or assets, or leverage it to the detriment of the organization. Social engineering or hacking are examples of a breach of confidentiality in the context of information systems.

Data integrity is the existence of data in its original state without anyone tampering with it or degrading it. The modification may be unintentional or intentional. Nevertheless, it downgrades the quality of data such that it cannot be fully trusted to fulfill the intended purpose (DAWSON; THOMSON, 2018). This creates an opportunity for risks that can harm the organization if they are not mitigated in time. Data integrity can be diminished when data is tampered with when being transmitted or during its storage.

Availability refers to the existence of the data when it is needed by authorized parties. Data might be deleted intentionally or accidentally. Therefore, information security programs ensure that such instances do not occur. Deleted data can compromise business operations or even bring them to a standstill (DAWSON; THOMSON, 2018). Therefore, communication channels, security controls, and computing systems should function optimally. Medical equipment, power generation, and safety systems are dependent on availability. Therefore, a cybersecurity program should make them resilient against attacks or threats.

2.2 Android Security Architecture Fundamentals

The Android OS is based on the Linux kernel; However, its developers have extensively modified some of the basic mechanisms, which ultimately led to increased protection (MAYRHOFER et al., 2021). More specifically, the Android OS includes specialized implementations for networking stack support, hardware drivers, file system management, and mechanisms for managing memory, CPU, and power consumption. All these mechanisms are implemented using libraries written in C/C++, but all Android applications are executed in the Dalvik Virtual Machine (VM), which, in essence, is a subset of the JVM; However, unlike Java, Android uses its own class libraries and a more compact method of saving executable files, the DEX file format, which have the .dex extension. Android applications are deployed in special packages, which have the extension .apk and are very similar to Java jar files.

Each Android application has its own ID and runs in its own VM. For each VM, the principle of process and thread isolation applies. The Android OS launches the master process when loaded into memory, which spawns new instances of Dalvik VM - one for each application. All interaction of individual processes occurs only through the Linux kernel via system calls, and not directly. In addition, during OS startup, several processes are launched, which implement all the necessary services of the operating system.

The Android OS implements a different privilege control mechanism from the one found on standard Linux platforms, called permissions (KHARIWAL; SINGH; ARORA, 2020). Permissions control what an application can do on the device. There are permissions for working with the mobile network, for example (CALL_PHONE), working with the built-in camera (CAMERA) or accessing the Internet (INTERNET), and in order to get certain permissions, the application must declare them in its manifest file, which contains metadata about the application. When the application is installed, the set of these permissions is checked and the user is prompted to confirm them.

Another security concept introduced by the Android OS is the concept of intentions or intents (KHARIWAL; SINGH; ARORA, 2020). Intentions are an abstract concept of work or functionality that can be performed by an application in the future. In other words, this is something the application needs to do. Basically, intentions are made up of the following elements: 1) Actions are what the intent needs to achieve, such as dialing a number, opening a link, or transmitting data. 2) Data are the resources that the intent operates on. They are expressed as URI objects in Android. The type of data required for the intent varies depending on the action. The ability to combine actions and data allows Android to know exactly what the intent is going to do and what it needs to work with.

All these mechanisms solve the main problem of Linux - the omnipotence of the root user. Since each application is launched under its own identifier, the running processes can be easily classified by applications and rules for controlling access to OS objects can be defined for each of them.

2.3 Malware Analysis

Malware Analysis (MA) is a set of tools and techniques used to dissect malware and understand how it works in order to identify and defeat it (SIKORSKI; HONIG, 2012). It is comprised of Static Analysis (SA), Dynamic Analysis (DA), and Hybrid Analysis (HA).

SA is composed of a set of tools and techniques the perform the analysis of a file without launching it for execution (SIKORSKI; HONIG, 2012). It can be basic or advanced. In the first case, the machine code instructions (Assembly code) in the file are not analyzed directly, instead, its performed a search for artifacts that are atypical for ordinary files, such as specific code patterns, strings, and API functions calls. In the latter case, the Assembly code is analyzed in detail in order to determine what exactly the program does. The main advantage of SA 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 SA 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 DA.

DA is composed of a set of tools and techniques to examine a file while running it in an isolate environment (SIKORSKI; HONIG, 2012). It can also be basic or advanced. Basic DA is the study of a file and its launch without using debugging tools; it consists in tracking events associated with this file, for example, access to the registry, disk operations, interaction with the network, etc. Advanced DA consists of examining the behavior of a running file using debugging tools. The main advantage of DA is that its immune to code is obfuscation. Some disadvantages of DA include its low code coverage and high time consumption.

Finally, HA leverages both SA and DA to understand malware effectively by taking advantage of both SA and DA resources.

It is worth noting that whereas Malware Analysis techniques aim to provide a set of tools for extracting and analysing malware, it does not focus on the problem of malware detection itself, however, malware detection heuristics and algorithms take advantage of the information collected using Malware Analysis.

2.4 Knowledge Discovery in Databases

Knowledge Discovery in Databases (KDD) is an iterative, interactive, and nontrivial process composed of several stages for extracting knowledge from large databases (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996). The resulting knowledge can be rules describing relationships between data properties (decision trees), common patterns (association rules), as well as results of classification (neural networks) and data clustering (Kohonen maps).

The KDD process consists of the following steps:

- Preparing the original dataset: This stage consists in creating a dataset, including from various sources, choosing a training sample, etc. For this, there must be developed tools for accessing various data sources. It is desirable to have support for working with data warehouses and the presence of a semantic layer that allows you to use not technical terms, but business concepts for preparing the initial data.

- Data preprocessing: In order to effectively apply data mining techniques, it is necessary to pay attention to the issues of data preprocessing. Data may contain gaps, noise, abnormal values, etc. In addition, the data may be redundant. In some problems, it is required to supplement the data with some a priori information. Therefore, the first stage of KDD is data preprocessing. Moreover, sometimes the dimension of the original space can be very large, and then it is desirable to use special algorithms for dimensionality reduction. This is both the selection of significant features and the mapping of data into a space of a lower dimension.

- Data transformation: This step is necessary to bring the information into a form suitable for subsequent analysis. In addition, there are some analysis methods that require the original data to be in some specific form. Neural networks, for example, work only with numerical data, and they must be normalized or encoded. Other DL networks requires data in tensor or sequential format.

- Data Mining: In this step, various algorithms are applied to extract knowledge. These are the traditional ML methods, DL methods, clustering algorithms, etc.

- Post-processing of data: Interpretation of the results and application of the acquired knowledge in business or scientific applications.

The KDD process does not specify a set of processing methods or algorithms suitable for analysis, rather it defines the sequence of actions that must be performed in order to obtain knowledge from the data source. This approach is universal and does not depend on the subject area. Figure 2.1 illustrates the KDD process steps.

This dissertation leverages of the KDD process in order to define the ML workflow for training and evaluation of the MDL method Chimera. Each subnetwork implements the process for training and evaluation, which can be performed in parallel. Finally, the MDL network implements the process in sequence.

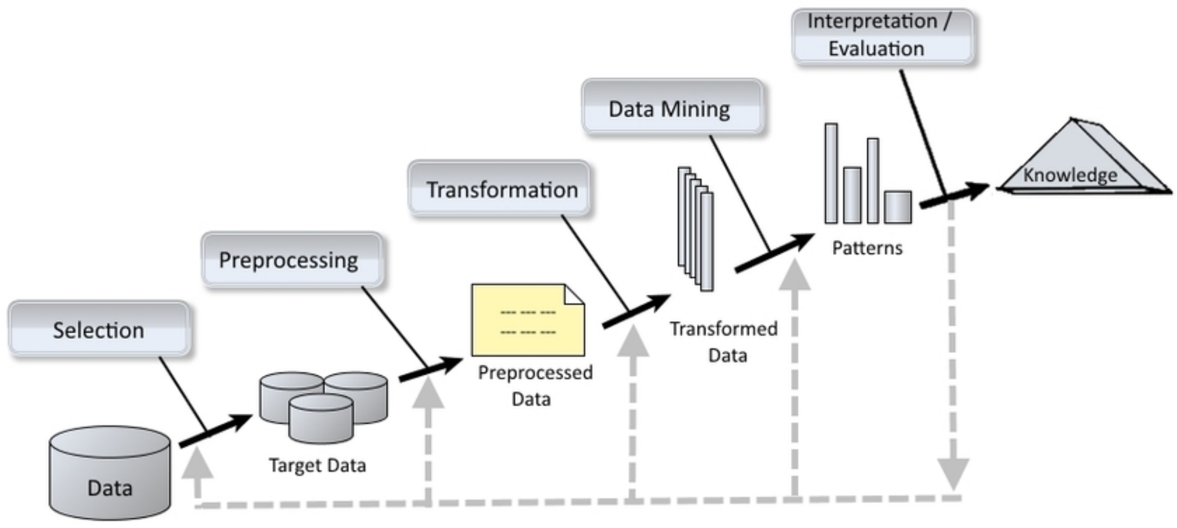


Figure 2.1: *The Knowledge Discovery in Databases process steps.*

Source : Adapted by the author from (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996)

2.5 DEEP LEARNING TECHNIQUES

This section is composed of five subsections. Subsection 2.5.1 explains the concept of Deep Learning (DL). Subsection 2.5.2 presents a step-by-step process for training Deep Neural Networks (DNNs). Subsections 2.5.3, 2.5.4, and 2.5.5 introduce the specialized DL architectures Convolutional Neural Networks (CNN), Transformer Neural Networks (TNN), and Multimodal Deep Learning (MDL) used in this dissertation.

2.5.1 THE CONCEPT OF DEEP LEARNING

DL is a type of ML that involves extracting, or modeling, data features using complex multi-layered networks (GOODFELLOW; BENGIO; COURVILLE, 2016). Since DL is a very general approach of modeling, it is capable of solving complex problems such as computer vision and NLP tasks. The DL approach is significantly different from both traditional programming and other ML methods. Traditional ML methods are trained to learn how to classify or predict an already selected, preprocessed/transformed set of features in a dataset, whereas DL methods are able to perform automatic feature learning or extraction in the earlier network layers and classification or prediction in the last layers, thus, DL performs what is called hierarchical representation learning. DL not only can give results where other methods will not work, but also allows building a more accurate model or reduce the time to create it since manual feature engineering can be minimized or completely avoided, and sometimes requires highly specialized workforce. Nevertheless, the two major disadvantages of DL are the computational resources needed for training and optimization and the difficulty in interpreting the resulting models. In spite of that, DL models have achieved state-of-the-art results in several different fields involving unstructured data such as image, audio, and video recognition, NLP, automatic language translation, automatic game playing, and so on.

The defining characteristic of DL is having more than one processing layer with trainable weights between input and output. Early layers are responsible for feature learning or extraction. Later layers are responsible for classification or prediction. The deeper in the network, the higher the feature level or abstraction. Therefore, each layer transforms the input data of the previous layer into a more abstract representation, and the output layer combines those representations to make predictions. Figure 2.2 depicts process of feature learning in DL and contrasts it with the traditional ML process.

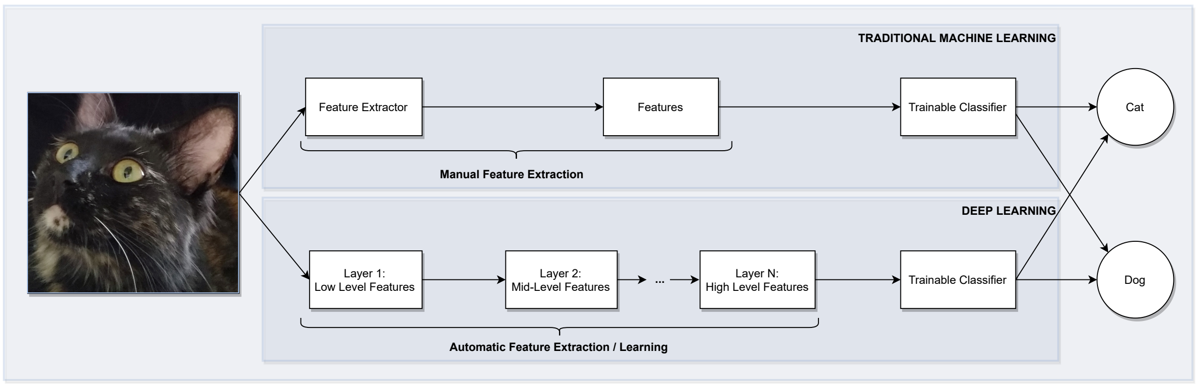

TRADITIONAL MACHINE LEARNING
Feature Extractor
Features
Trainable Classifier
Cat
Manual Feature Extraction
DEEP LEARNING
Layer 1: Low Level Features
Layer 2: Mid-Level Features
...
Layer N: High Level Features
Trainable Classifier
Dog
Automatic Feature Extraction / Learning

Figure 2.2: *Traditional Machine Learning flow contrasted with Deep Learning flow for an image classification task.*

Source : Adapted by the author from (GOODFELLOW; BENGIO; COURVILLE, 2016)

There are several specialized types of DL networks composed of specialized layers for feature learning from different data sources, from unstructured data such as images and audio, semi-structure data such as JSON and XML files, to structured and relational data such as database tables. this dissertation makes use of the following DL networks: 1) DNNs, which are the extension of the classical ML MLP architecture. 2) CNNs, which are specialized DL networks for image feature learning, and 3) TNNs, which are specialized DL networks for sequence learning, largely used in NLP and automatic language translation tasks. Moreover, this dissertation combines those DL architectures in a MDL architecture in order to increase Android malware detection rate by combining the high level feature representations extracted using the each of the unimodal DL networks aforementioned.

2.5.2 Training Deep Neural Networks

This subsection is organized following the usual steps necessary to train a DNN (GOODFELLOW; BENGIO; COURVILLE, 2016), where each subsection defines and explains a processing step in the training process. In general, first the inputs are forward-propagated (Subsection 2.5.2.1) and activated (Subsection 2.5.2.2) iteratively until the last layer of the network is reached. Then, a loss or error function (Subsection 2.5.2.4) is applied to the last layer's result (Subsection 2.5.2.3) in order to measure the network error. Next, the network error is used by the Backpropagation algorithm (Subsection 2.5.2.5) to estimate the gradient of the loss function that, in turn, can be used by an optimization algorithm to apply and control the corrections on the network weights with the aim of minimizing the network error (Subsection 2.5.2.6). Finally, Subsection 2.5.2.7 presents DL techniques used to mitigate overfitting.

2.5.2.1 Feedforward Deep Neural Networks

In general, in feedforward DNNs, neurons are the basic processing units, and are organized into separate layers: one input layer, any number of hidden processing layers, and one output layer (GOODFELLOW; BENGIO; COURVILLE, 2016). The outputs from each layer only go to the next layer after being processed following the forward propagation step and the application an activation function, used to introduce nonlinearity to the network. Figure 2.3 depicts a L-layer DNN where each layer is composed of an arbitrary number of neurons.

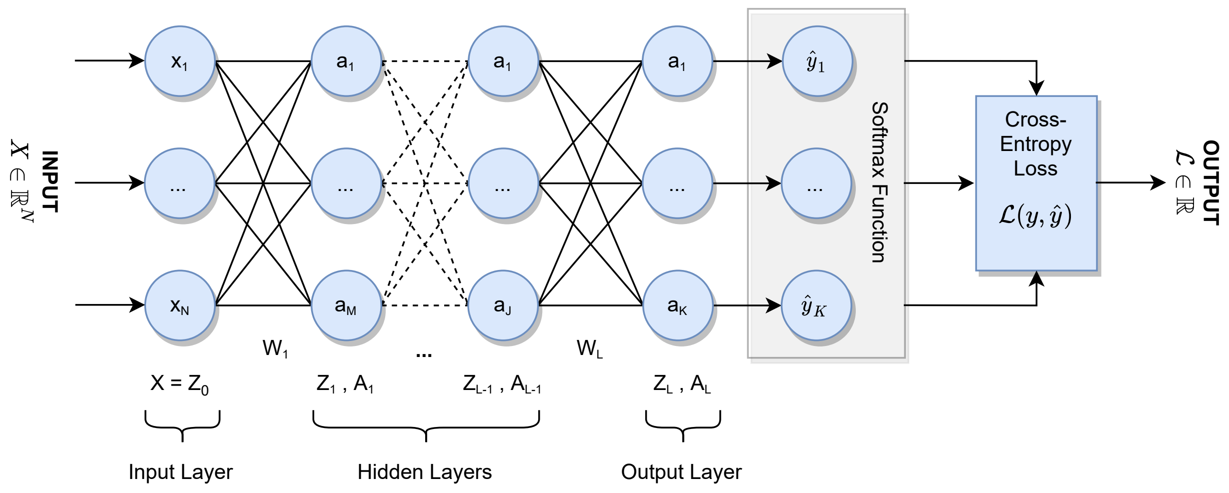


Figure 2.3: *Forward propagation on a Deep Neural Network designed for multiclass classification.*

Source : Adapted by the author from (GOODFELLOW; BENGIO; COURVILLE, 2016)

Equation 2.1 defines the forward propagation operation followed by the application of the activation function, where Z_i represents the matrices resulting from the linear transformations for each layer, A_i represents represents the matrices of activations for each layer, W_i represents the weight matrices for each layer, f represents the activation function, and $\ell \in \{1, ..., L\}$ is the layer number. Figure 2.3 presents the forward propagation matrices for each layer.

$$\begin{aligned} Z_\ell &= W_\ell Z_{\ell-1} \\ A_\ell &= f(Z_\ell) \end{aligned} \tag{2.1}$$

2.5.2.2 Rectified Linear Unit (ReLU) Activation Function

In this dissertation, the Rectified Linear Unit (ReLU) activation function is used to introduce nonlinearity to the networks (GOODFELLOW; BENGIO; COURVILLE, 2016). The ReLU activation function is largely used in DL models due to its low computational cost for both the forward propagation and the backpropation steps since it is defined as a piecewise function composed of two linear functions, as defined by Equation 2.2 and shown in Figure 2.4.

$$f(x) = ReLU(x) = max(0, x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \implies \frac{\partial f}{\partial x} = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{2.2}$$

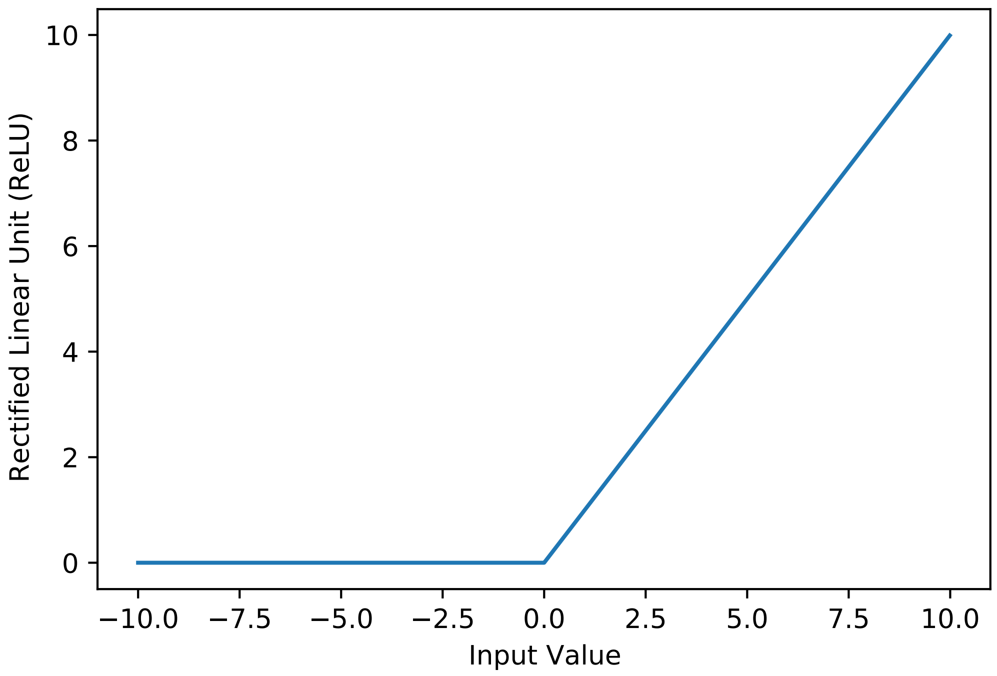


Figure 2.4: *Rectified Linear Unit (ReLU) activation function plot.*

Source : Adapted by the author from (GOODFELLOW; BENGIO; COURVILLE, 2016)

Consequently, its gradient can be easily calculated, as presented by Equation 2.2. Notice that although the ReLU activation function is a combination of two linear functions, the functions iteration that take place in the forward propagation process given by the Equation 2.1 can generate complex, nonlinear decision boundaries, as depicted in Figure 2.5.

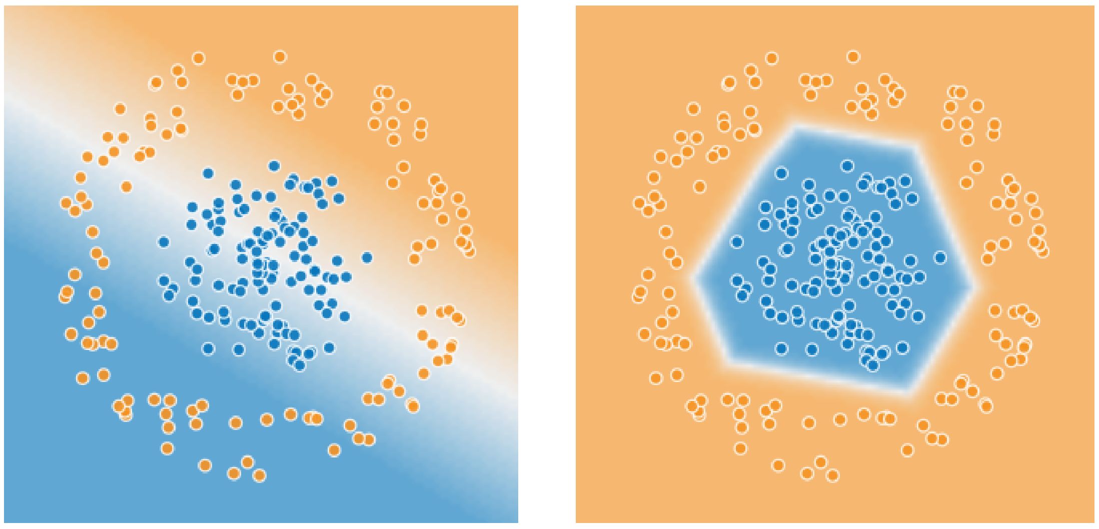


Figure 2.5: *Decision boundaries generated using of the identity linear activation function and the ReLU activation function.*

Source : Adapted by the author from (TENSORFLOW, 2021)

The same does not hold true for a linear activation function, since the composition of linear functions is still linear, therefore the generated decision boundaries are defined as hyperplanes.

2.5.2.3 Softmax Activation Function

Supervised learning of a DNN is done following the same steps as in traditional ML. Take a network with training data batches, compare the network output with the desired output or ground truth using a loss function - also know as cost or error function - to generate an error vector, and apply corrections to the network weights based on error vector using an optimization algorithm that in turn uses the Backpropagation algorithm to calculate the gradient of the loss function.

In this dissertation, since the proposed methods are designed to solve a (binary) classification task, the output of the network is calculated using the Softmax activation function and, consequently, the loss function is calculated using the Cross-Entropy Loss function.

The Softmax activation function is a generalization of the logistic function to multiple classes (GOODFELLOW; BENGIO; COURVILLE, 2016). As shown in Figure 2.3, it is used in multinomial logistic regression as the last activation function of a neural network to normalize the output to a probability distribution over predicted output classes. The Softmax function is defined following Equation 2.3, where Z represents a K-dimensional input vector representing the K-classes of the classification task and $k \in \{1, ..., K\}$.

$$\sigma(Z) = \frac{e^{Z}}{\sum_{k=1}^{K} e^{Z_k}} \tag{2.3}$$

2.5.2.4 Cross-Entropy Loss Function

The Cross-Entropy can be used to define a loss function in ML and optimization (GOODFELLOW; BENGIO; COURVILLE, 2016). Equation 2.4 defines the Cross-Entropy Loss function considering K classes, the true probability is the true label y, and the given distribution is the predicted value $\hat{y}$ of the current model resulting from the Softmax activation function. In particular, for $K = 2$ (binary classification problem), the equation can be reduced even further.

$$\mathcal{L}(y, \hat{y}) = -\sum_{k=1}^{K} y_k log(\hat{y}_k) \xRightarrow[True \equiv 1]{K=2} \mathcal{L}(y, \hat{y}) = -log(\hat{y}) \tag{2.4}$$

Figure 2.6 depicts the Cross-Entropy Loss for K = 2 and the true label equals to 1.

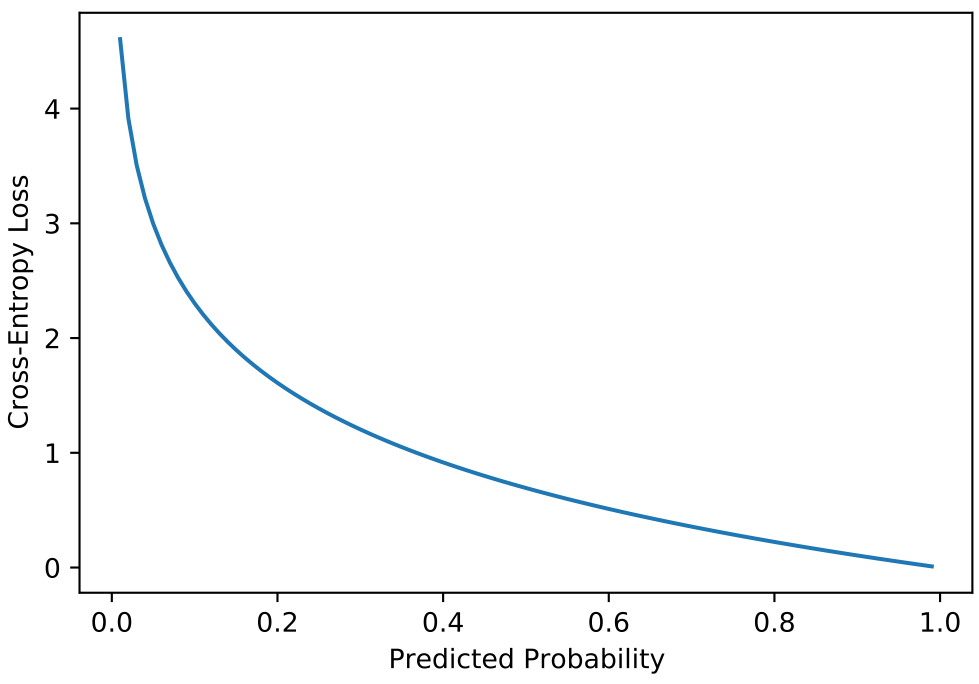


Figure 2.6: *Cross-Entropy Loss for a binary classification problem (K = 2) and the "True" label equals to 1.*

Source : Adapted by the author from (GOODFELLOW; BENGIO; COURVILLE, 2016)

Notice that as the predicted probability approaches 1, the Cross-Entropy Loss approaches 0, and as the predicted probability approaches 0, the Cross-Entropy Loss approaches $+\infty$. Figure 2.3 shows the Cross-Entropy Loss calculation as the last step in the forward propagation process.

2.5.2.5 Backpropagation Algorithm

The main algorithm used to calculate the corrections to the network weights is the Backpropagation algorithm (GOODFELLOW; BENGIO; COURVILLE, 2016). The Backpropagation algorithm is used to calculate the gradient vector of the loss function with respect to the model weights in order to find the right direction to minimize error. In general, the gradient vector of the loss function with respect to the model weights is given by the Equation 2.5, which can be expanded using the multivariate chain rule (STRANG; HERMAN, 2016) and assume different forms depending on the network architecture.

$$
\begin{gathered}
\vec{\nabla}\mathcal{L}(W_1, ..., W_L) = \left(\frac{\partial \mathcal{L}}{\partial W_1}, ..., \frac{\partial \mathcal{L}}{\partial W_L}\right), \text{ where} \\
\frac{\partial \mathcal{L}}{\partial W_i} = \frac{\partial \mathcal{L}}{\partial \hat{Y}} \frac{\partial \hat{Y}}{\partial A_L} \frac{\partial A_L}{\partial Z_L} \frac{\partial Z_L}{\partial A_{L-1}} \cdots \frac{\partial Z_i}{\partial W_i}, i \in \{1, ..., L\}
\end{gathered}
\tag{2.5}
$$

As shown in Figure 2.7, the gradient vector points to the direction of steepest ascent. This information can be leveraged by optimization algorithms in order to minimize the network error by adjusting the weights accordingly.

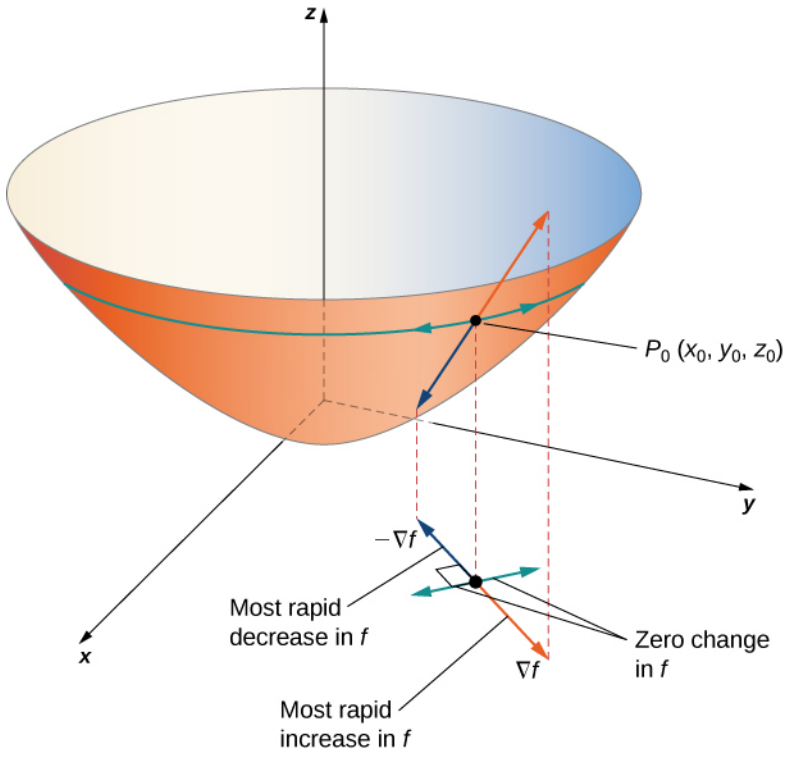


Figure 2.7: *The gradient vector of an arbitrary function f.*

Source : Adapted by the author from (STRANG; HERMAN, 2016)

2.5.2.6 Adaptive Moment Estimation (Adam)

The application of the corrections is controlled by an optimization algorithm and a learning rate variable, which usually needs to be small to guarantee convergence. Neural network optimizers use some form of gradient descent algorithm to control backpropagation; this often involves a mechanism that helps to avoid getting stuck in local minima, such as applying momentum corrections to the gradient. Several optimization algorithms also adapt the learning rate of the model parameters by looking at the gradient history. In this dissertation, the Adaptive

Moment Estimation (Adam) optimization algorithm (KINGMA; BA, 2014) was chosen as the optimization algorithm for all the models. Adam is a method that computes adaptive learning rates and stores an exponentially decaying average of past squared gradients in order to avoid getting stuck into local minima and presents good performance in training different types of DL networks. The optimizer was designed to be appropriate for problems with very noisy or sparse gradients. This is the case of the gradients calculated using the multimodal dataset implemented in this dissertation since it contains a significant number of binary and one-hot encoded features. The decaying averages of past and past squared gradients is calculated following Equation 2.6, where m_t and v_t are estimates of the mean (first moment) and the uncentered variance (second moment) of the gradients.

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\vec{\boldsymbol{\nabla}}\mathcal{L}(W_1(t), ..., W_L(t)) \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)\vec{\boldsymbol{\nabla}}\mathcal{L}^2(W_1(t), ..., W_L(t)) \end{aligned} \tag{2.6}$$

In addition, it was implemented a bias-correction factor according to Equation 2.7.

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \end{aligned} \tag{2.7}$$

Finally, Adam's update rule is given by Equation 2.8.

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{2.8}$$

It was proposed the following default values for the Adam's hyperparameters, $\beta_1 = 0,9$, $\beta_2 = 0,999$, $\epsilon = 10^{-8}$, and $\eta = 0,001$. Figure 2.8 presents the training cost performance comparison between state-of-the-art optimization algorithms including Adam. As we can in Figure 2.8, Adam presents faster convergence and good stability.

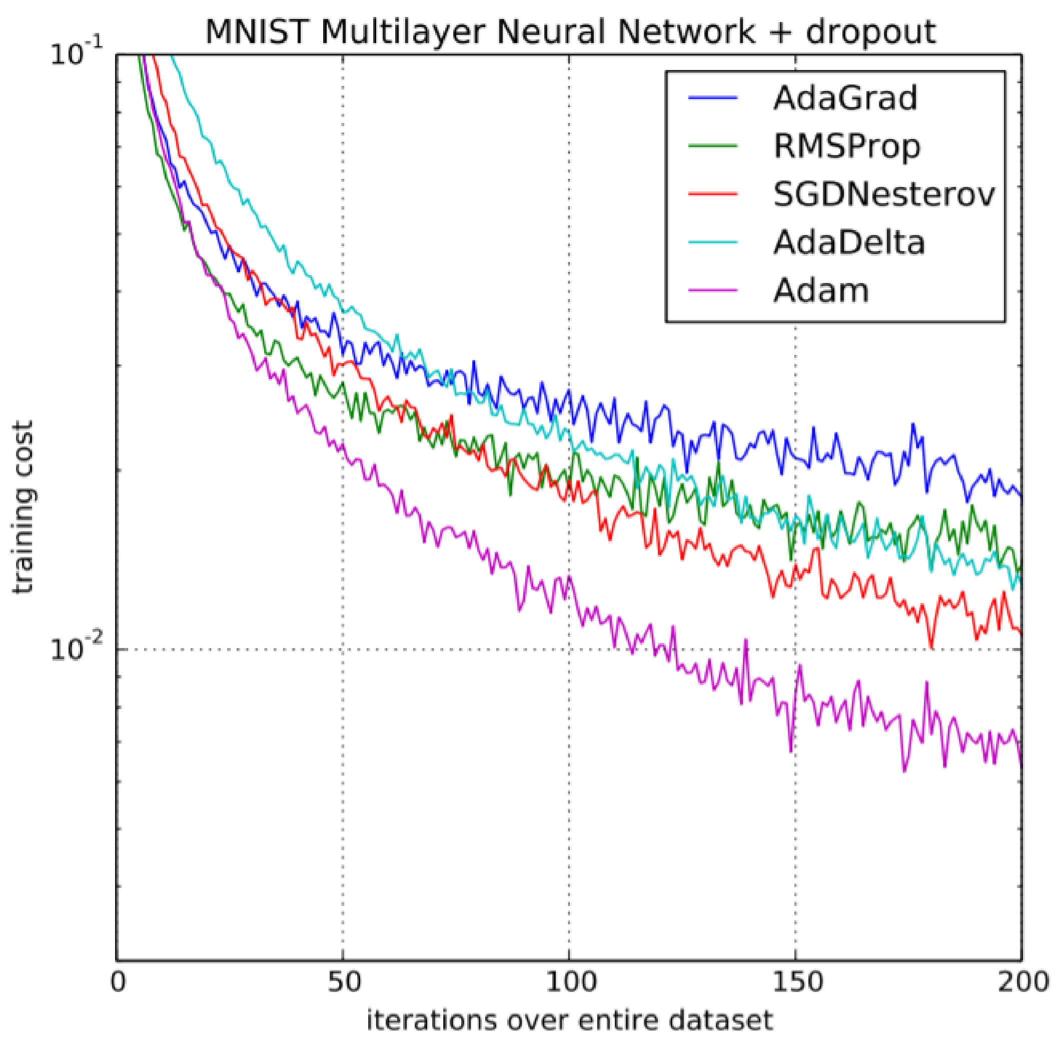


Figure 2.8: *Training cost performance comparison between state-of-the-art optimization algorithms.*

Source : Adapted by the author from (KINGMA; BA, 2014)

2.5.2.7 Mitigating Overfitting

Overfitting or high variance is a common problem found when building and evaluating ML and DL models. In particular, DL models are composed of multiple layers and a large number of parameters (GOODFELLOW; BENGIO; COURVILLE, 2016), which might increase overfitting.

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 validation error reaches a high value, meaning that the network was not able to generalize or learn the required patterns from the training set. Overfitting indicates that the model memorized the patterns instead of learning them. Commonly used approaches to mitigate 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.

Underfitting or high bias is a less common problem found in DL models, and indicates that the model was not able to generalize due to its low complexity or low number of parameters. Thus, the decision boundaries generated by the network are not complex enough to perform correct data separation into classes.

Figure 2.9 illustrates overfitting, underfitting and "just right" scenarios.

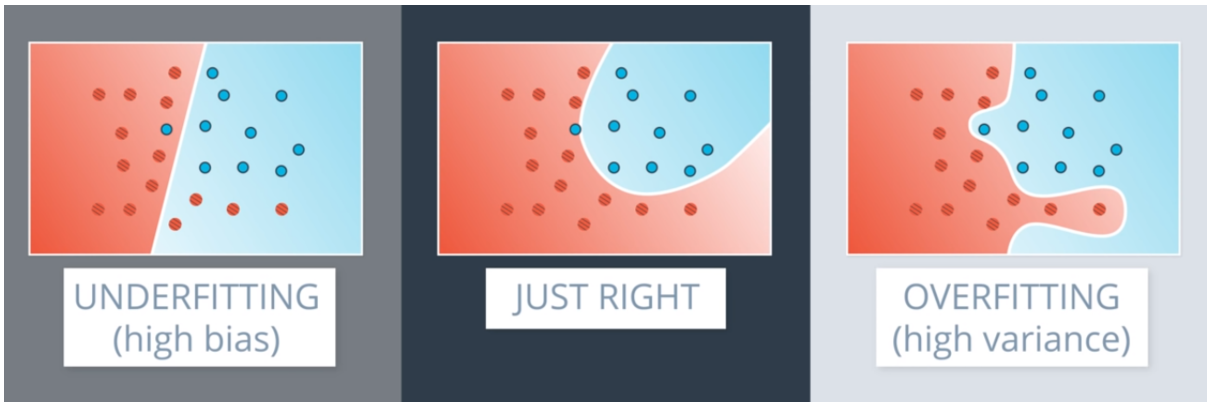


Figure 2.9: *Decision boundaries illustrating underfitting, "just right", and overfitting scenarios.*

Source : Adapted by the author from (PATEL, 2019)

The most common DL techniques used for mitigating overfitting are Dropout (SRIVASTAVA et al., 2014) and Batch Normalization (IOFFE; SZEGEDY, 2015). 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 (SRIVASTAVA et al., 2014). Figure 2.10 illustrates the effects of Dropout on an arbitrary network.

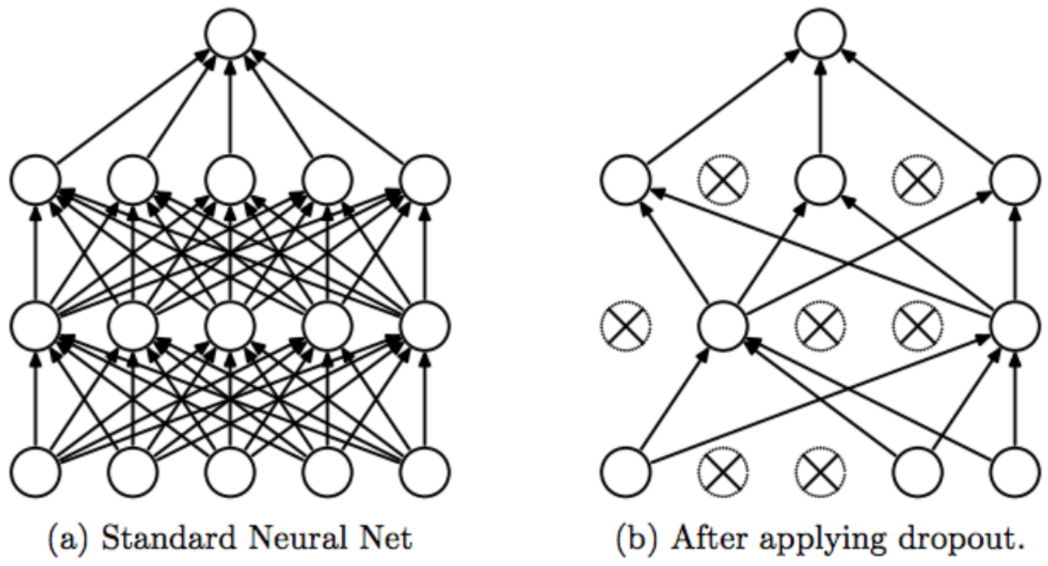


Figure 2.10: *Effects of Dropout on an arbitrary network.*

Source : Adapted by the author from (SRIVASTAVA et al., 2014)

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 instability and slow convergence. Equation 2.9 defines the Batch Normalization technique where $\mathcal{B}$ represents the mini-batches, $\hat{x}_i \in \mathcal{B}$, and β, γ are learnable parameters.

$$\begin{aligned} \hat{x}_i &= \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \\ \hat{y}_i &= \gamma \hat{x}_i + \beta \end{aligned} \tag{2.9}$$

2.5.3 Convolutional Neural Networks

The Convolutional Neural Network (CNN) architecture was inspired by the organization of the visual cortex and is similar to the connectivity architecture of neurons in the human brain (LECUN et al., 1998). Individual neurons respond to stimuli only in a limited area of the visual field, the receptive field. A combination of such fields is used to cover the entire area of the visual field. Using the Convolution operation, CNNs are able to capture the spatial dependencies in an image through the application of relevant filters or kernels. While the filters are handcrafted in traditional computer vision and ML techniques, CNNs with sufficient training are able to learn these filters in order to perform feature extraction. Moreover, CNNs require significantly less preprocessing compared to other DL networks since it performs a better fitting to the image dataset due to the reduction in the number of parameters involved and weight sharing. CNNs do not need to be limited to only one Convolutional layer. In practice, the first Convolutional layer is responsible for capturing the low-level features such as edges, gradient orientation, color, etc. Subsequent layers are responsible for capturing higher-level features such as the compositions of the lower-level features.

In addition to the Convolutional layer, the Pooling layer is responsible for reducing the spatial size of the features, consequently, reducing the computational power required to process the data through dimensionality reduction. Moreover, it is also useful for extracting dominant features which are positional invariant. There are two types of pooling operation: 1) Max Pooling is used to return the maximum value from the part of the image covered by the filter. 2) Average Pooling is used to return the average of all values from the part of the image covered by the filter.

The number of layers, their organization, and ordering defines the network architecture. After the application of a number of Convolutional and Pooling operations through several layers, the extracted features are flattened and passed to a final fully-connected layer for classification or regression purposes. Since Convolutional and the Pooling operations are differentiable, they can be used in DL networks and trained as described in Sections 2.5.2.5 and 2.5.2.6.

Figure 2.11 shows a CNN used to classify handwritten digits.

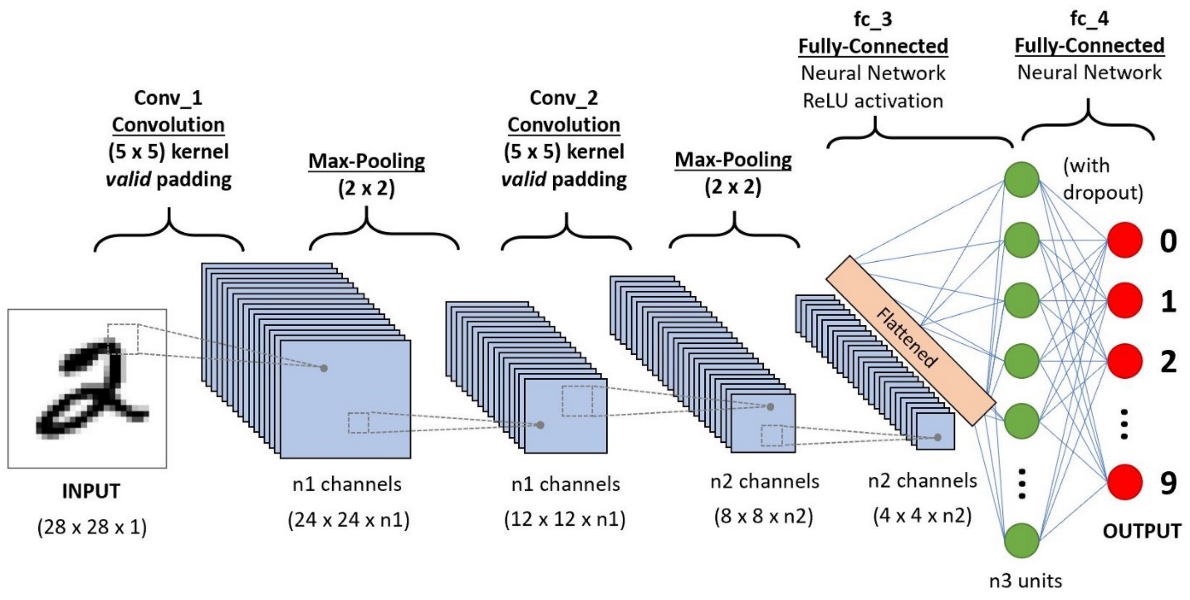


Figure 2.11: *A CNN architecture to classify handwritten digits.*

Source : Adapted by the author from (SRIVASTAVA et al., 2014)

2.5.4 Transformer Neural Networks

A Transformer Neural Network (TNN) is a DL model that implements the mechanism of attention to differentially weight the significance of each part of the input data (VASWANI et al., 2017). TNNs are designed to handle sequential input data, such as in NLP, for tasks such as text summarization and translation. However, unlike Recurrent Neural Networks (RNN), transformers do not need to process the data in order. Instead, the attention mechanism introduces context for any position in the input sequence. As an example, if the input data is a natural language sentence, the TNN does not need to process the beginning of the sentence before the end. Instead, it identifies the semantic meaning of each word in the sentence. Therefore, it allows for more parallelization than RNNs, thus reducing training times.

Since their introduction in 2017, TNNs are increasingly the model of choice for NLP problems, replacing RNN models. The additional training parallelization allows using larger datasets than was once possible, which led to the development of pretrained models such as BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer), trained with large language datasets.

TNNs adopts an encoder-decoder architecture. The encoder contains encoding layers that process the input iteratively, whereas the decoder consists of decoding layers that do the same operations to the encoder's output. Encoder layer generates encodings that contain information about which parts of the inputs are more relevant to each other. The encodings are them passed to the next encoder layer as inputs. Each decoder layer does the opposite, considering all the encodings and using their contextual information to generate an output sequence. Each encoder-decoder layer makes use of an attention mechanism, and contains a feed-forward neural network for additional processing of the results, residual connections, and a normalization layer.

Figure 2.12 depicts a full-fledged Transformer architecture, which includes the encoder, decoder and the attention layers.

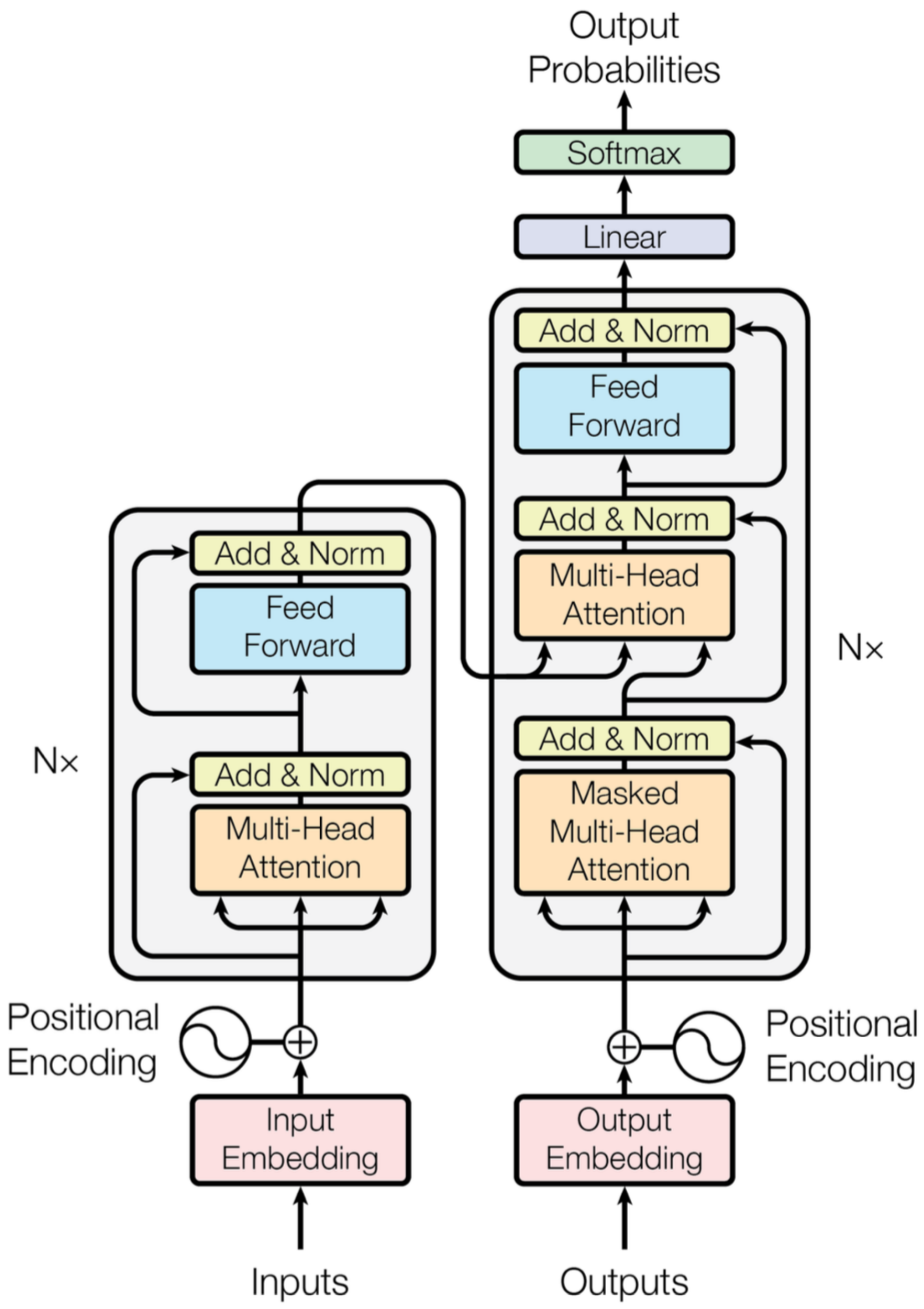


Figure 2.12: *Transformer Neural Network Architecture.*

Source : Adapted by the author from (VASWANI et al., 2017)

2.5.5 Multimodal Deep Learning

Modes refer to the channels that are used to deliver information. They include the methods that individuals use to learn. The data from different sources that are used in the learning process are semantically correlated. Some provide additional information that complements various sources of information. For example, when individuals want to detect an individual's emotion, they may use the information they gathered from the voice tone and the eye contact of the individual to establish the mood of the individual. Therefore, it easier to decode information by combining different sensory techniques.

Single modalities have been successfully used to facilitate supervised feature learning that uses DL networks. Some of the single modalities used for DL networks include audio, images, and text. The modalities focus on collecting information from different sources to improve the prediction abilities of the networks. In MDL (NGIAM et al., 2011), there should be a minimum of two information sources and an information processing model or network, used to combine the information sources.

In order to combine information, an early fusion layer can be defined to join the features before being processed by the network. The network is responsible to perform feature learning from the features already correlated using manual feature engineering. Another common technique in MDL is to define an intermediate fusion layer. The features should be learned from single sources of information by building models that suit the type of data used. Moreover, there should be a direct relationship of modalities to facilitate mapping information from the sources. Finally, a late fusion layer can also be defined to perform fusion of the high-level feature representations learned from each data modality. After learning the features, it is essential to combine them in a shared representation layer to be used by a classifier such as a DNN for making predictions.

2.6 Literature Review

There are several research papers on Android malware detection using traditional ML and unimodal DL methods (ALSMADI; ALQUDAH, 2021; LIU et al., 2020; QIU et al., 2020); However, taking into account that the general objective of this dissertation was to develop and evaluate new MDL Android malware detection method that leverages multiple data modalities extracted using HA, and combines both manual and automatic feature engineering with the aim of increasing Android malware detection rate, this literature review was performed focusing on semantic search rather than keywords search. Therefore, it was considered as the starting point the research article presented by (KIM et al., 2019), in which the use of MDL for Android malware detection using several data modalities was introduced for the first time. Next, three search approaches were implemented:

1. Cross-reference search: It was performed a cross-reference search in order to find all the research articles that cite (KIM et al., 2019).

2. Similarity search: The free online tool Connected Papers (EITAN et al., 2021) was used to generate a similarity graph of the connected papers, as depicted in Figure 2.13. The similarity graph built by Connected Papers is based on the concepts of Co-citation (SMALL, 1973) and Bibliographic Coupling (KESSLER, 1963) in which two papers that have highly overlapping citations and references are presumed to have a higher chance of treating a related subject matter, even if they do not cite each other directly, complementing the cross-reference search approach performed in 1). Moreover, the algorithm builds a graph in which similar papers are visually closer to each other.

3. Keyphrase search: It was performed a keyphrase search on Google Scholar (GOOGLE, 2021) and Microsoft Academic (MICROSOFT, 2021) for the following keyphrase: Android Malware Detection Multimodal Deep Learning.

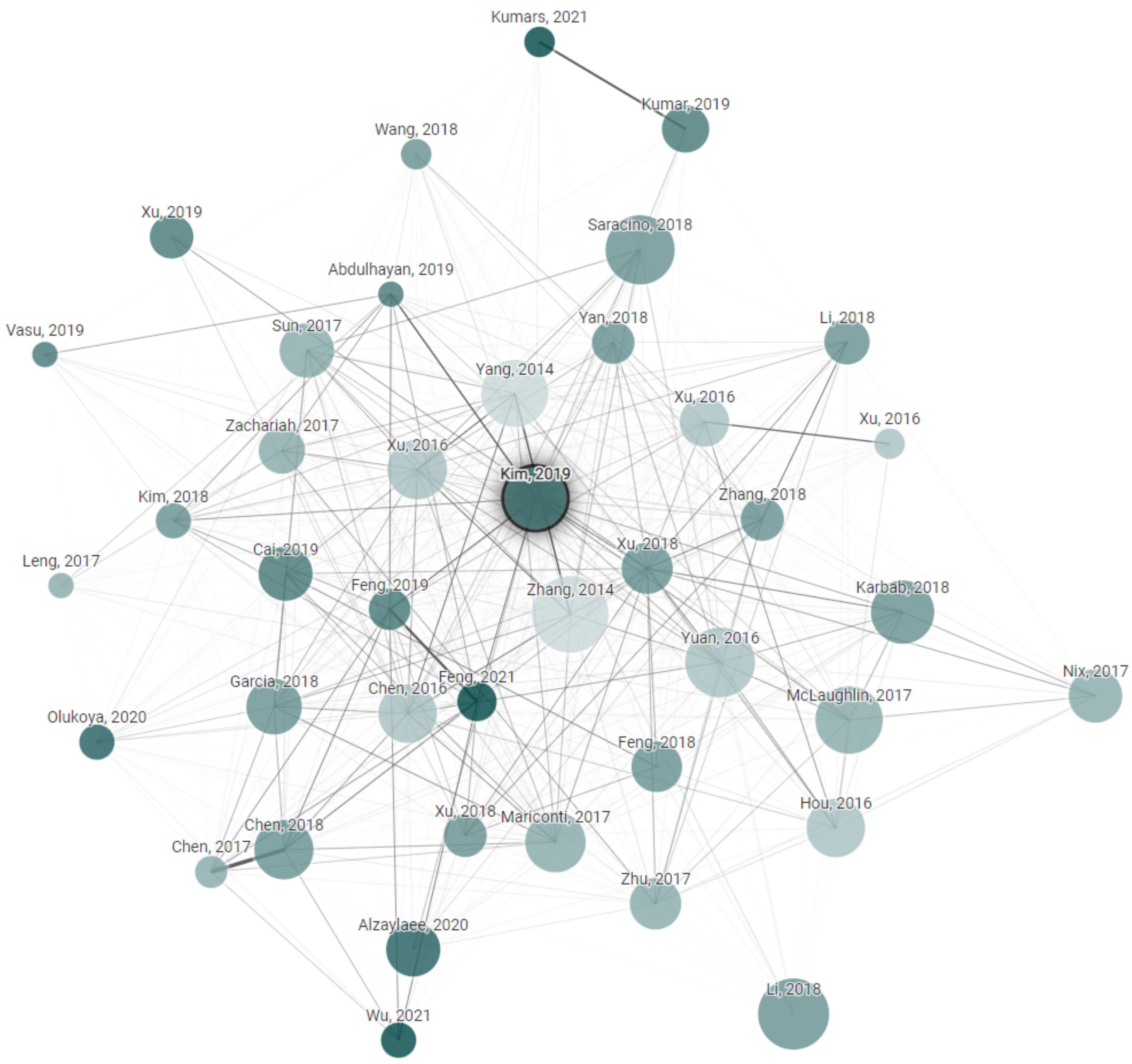


Figure 2.13: *Similarity graph for the paper (KIM et al., 2019). The size of the circles indicates the number of citations and the color of the circles indicates the publication year.*

Source : Adapted by the author from (EITAN et al., 2021)

Using the search approaches aforementioned combined with manual review of each paper, it was possible to find 10 research papers exploring different MDL architectures and data modalities for Android malware detection. The summarized information on the DL techniques and data modalities used in each article is presented in Table 2.1.

Article	Raw Data	DLT	SA Data	DLT	DA Data	DLT
(KIM et al., 2019)	N/A	N/A	S1, S2	T2	N/A	N/A
(VASU; PARI, 2019)	N/A	N/A	S1, S2	T2	N/A	N/A
(ZHU et al., 2019)	N/A	N/A	S1, S2	T1, T2	N/A	N/A
(MCGIFF et al., 2019)	N/A	N/A	S1	T2	N/A	N/A
(HWANG; CHUNG, 2020)	N/A	N/A	S1, S2	T1, T2	N/A	N/A
(AMRUTHA; BALAGOPAL, 2020)	N/A	N/A	S1, S2	T1, T2	D1, D2, D3	T1, T2
(NGUYEN et al., 2021)	N/A	N/A	S1, S2	T2	N/A	N/A
(DHALARIA; GANDOTRA, 2021)	N/A	N/A	S1, S2	T2	D1, D2, D3	T2
(MILLAR et al., 2021)	R1	T1	S1, S2	T1, T2	N/A	N/A
(OLIVEIRA; SASSI, 2021c)	**R1**	**T1**	**S1**	**T2**	**D1**	**T3**

Table 2.1: *Selection of articles on Android malware detection using Multimodal Deep Learning.*

DLT - Deep Learning Technique
R1 – DEX opcodes / grayscale images
S1 – Manisfest metadata
S2 – Decompiled / Disassembled DEX
D1 – System call sequences
D2 – Dynamic permissions
D3 – Information leaks
T1 – Convolutional Neural Networks
T2 – Feedforward Deep Neural Networks
T3 – Transformer Neural Networks

As we can see in Table 2.1, the only research paper found in the literature that considers the three data modalities and a different, specialized DL technique for each data modality is (OLIVEIRA; SASSI, 2021c), which is the published conference paper for this dissertation. All the other works developed bimodal DL methods and used at most two different DL techniques. Thus, this dissertation tackles that research gap by developing and evaluating Chimera. In addition, in order to improve repeatability and reproducibility, Chimera's performance is directly compared to traditional ML methods, ensemble ML methods, and unimodal DL methods using and extending the Omnidroid benchmark dataset.

Chapter 3

Materials and Methods

3.1 Methodology

The research methodology used in this dissertation is defined as applied research, as it aims to generate new knowledge for the purpose of solving a real world problem, thus having practical implications (GIL, 2008).

This dissertation also follows the quantitative research method, in which mathematical and statistical techniques are used to quantify and compare the results obtained (GERHARDT; SILVEIRA, 2009).

Regarding the technical procedures, this dissertation is classified as experimental research since it consists of determining an object of study and selecting the variables that can influence it (GIL, 2008). More concretely, the proposed method is the object of study and its architecture and hyperparameters are the variables. By changing the architecture or the hyperparameters, the method achieves different performance levels. To implement the experimental research, computational experiments were performed following the Supervised ML approach (ALPAYDIN, 2020).

Finally, in Section 2.6, a literature review was carried out considering MDL Android malware detection methods, and a comparison between those methods was established with the aim of identifying and understanding the current research gaps and how they could be exploited.

3.2 The Proposed Method

The method proposed in this dissertation follows the KDD process (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996) and Supervised ML (ALPAYDIN, 2020) in order to implement the necessary steps for data extraction, preprocessing, transformation, and mining. Figure 3.1 presents Chimera's and its subnetworks' architectures, Chimera-S (CHS), Chimera-R (CHR), and Chimera-D (CHD), responsible for feature learning from Static Analysis (SA) data, DEX grayscale images, and Dynamic Analysis (DA) data respectively.

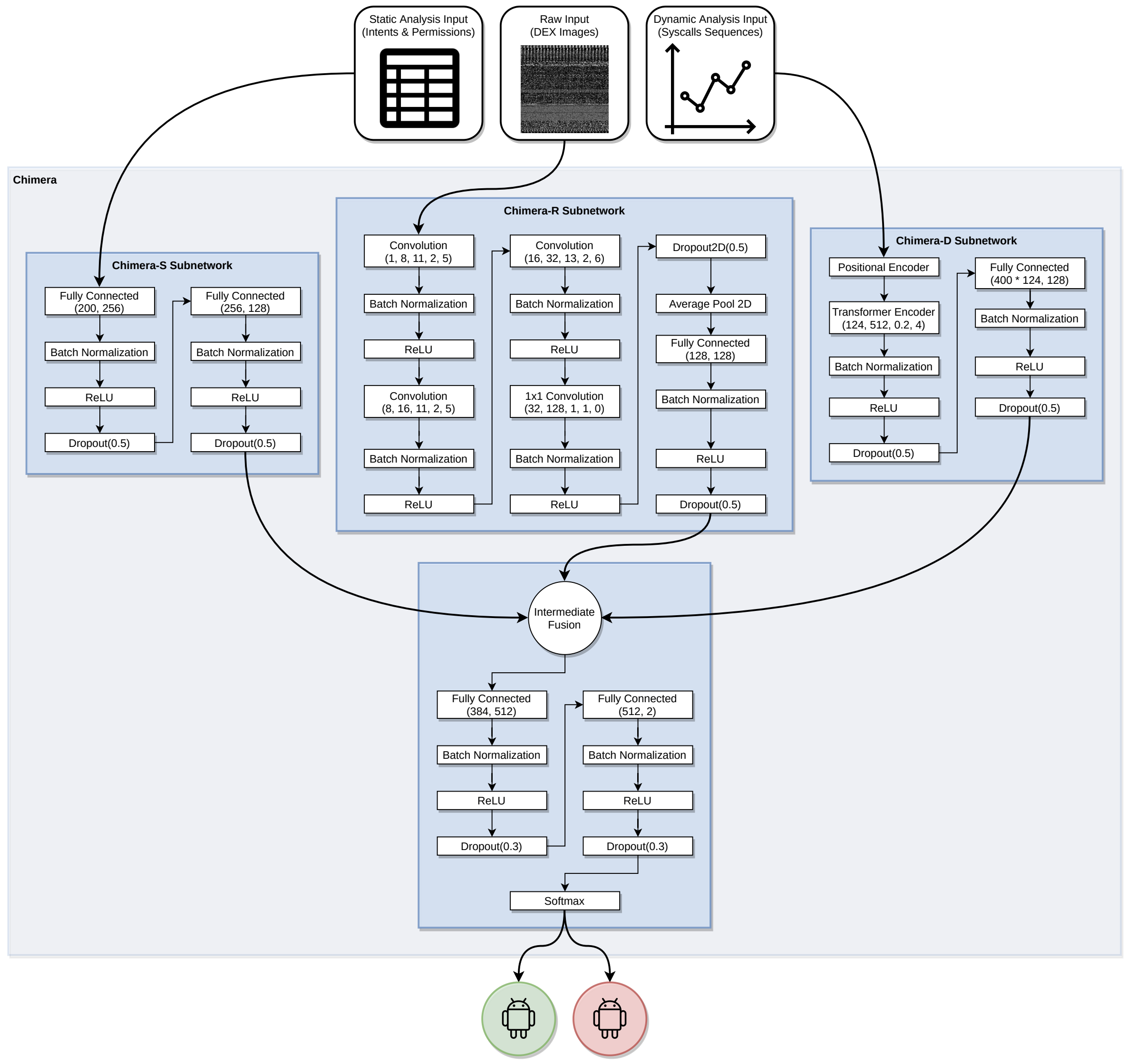


Figure 3.1: *Chimera multimodal deep learning Android malware detection method architecture.*

Figure 3.2 depicts the proposed 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 Supervised ML such as model selection, training, and evaluation. Since Chimera is a MDL method, each DL subnetwork, i.e, CHS, CHR, and CHD is responsible for feature learning from a different data modality. Therefore, the Selection, Preprocessing, Transformation, and Data Mining stages are performed independently for each DL subnetwork, suggesting the possibility of implementing parallelization in production environments. Moreover, an additional Data Mining implementation step is executed over the intermediate fusion layer to

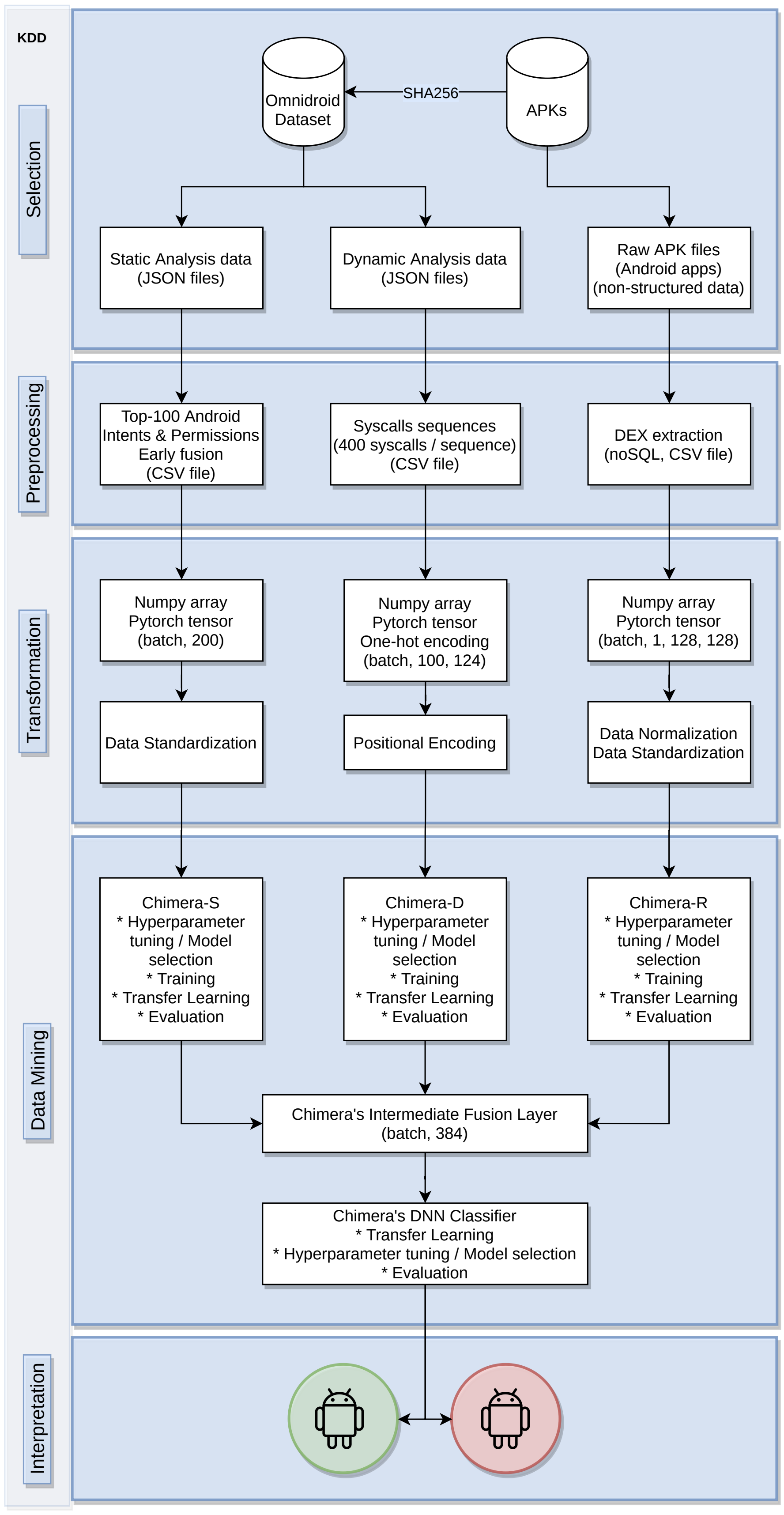

KDD
Selection
Preprocessing
Transformation
Data Mining
Interpretation
Omnidroid Dataset
SHA256
APKs
Static Analysis data (JSON files)
Dynamic Analysis data (JSON files)
Raw APK files (Android apps) (non-structured data)
Top-100 Android Intents & Permissions Early fusion (CSV file)
Syscalls sequences (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 3.2: *Chimera multimodal deep learning Android malware detection method KDD process stages. The blue boxes represent each KDD process stage. The white boxes represent the implementation steps for each stage.*

process 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 Chimera can be summarized by its generalization performance, i.e. the detection accuracy resulting from 10-fold cross-validation.

3.3 Datasets and Experimental Setup

This dissertation proposes and builds a dataset composed of a fusion of features from SA data, DA data, and raw data.

The subset of SA and DA features was obtained from the Android benchmark dataset Omnidroid, introduced by (MARTÍN; LARA-CABRERA; CAMACHO, 2019). Omnidroid is a balanced dataset composed of more than 10000 features representing pre-static, static, and dynamic analysis information, extracted from 22000 real malware and benign Android applications. After applying several data extraction, preparation, cleaning and consolidation techniques, the Omnidroid dataset presented the information in the structured CSV and JSON formats ready to be transformed and used for knowledge extraction.

Due to legal restrictions, the Omnidroid dataset does not include the APK files used to extract the information. Since Chimera and CHR use data from the DEX files, which are part of the APK files, it was necessary to search and download the APK files from the online malware repositories Androzoo (ANDROZOO, 2021) using its free account access, and from Koodous (KOODOUS, 2021) using its premium account access. The joining between the Omnidroid records and the APK files was established using the SHA256 hash (Rachmawati; Tarigan; Ginting, 2018) of each application, which is uniquely associated to each APK. It is important to notice that Koodous is an APK repository and terms of use of each application is bound to the application itself, and the extraction and analysis of APKs raw bytes does not violate any licenses whatsoever.

Table 3.1 presents the structure of the resulting dataset produced as a part of this dissertation. The dataset is composed of a primary key defined as the SHA256 of the APK file.

As shown in Table 3.1, the dataset is composed of 16384 (128 x 128) integer features ranging from 0 to 255 representing the raw features interpreted as pixel values of the DEX grayscale images. 100 binary SA data features representing Android Intents. 100 binary SA data features representing Android Permissions. 400 integer DA data features representing system calls sequences, ranging from 1 to 124, and a binary class field representing the instance as either benign (0) or malware (1).

Field Name	**Field Type**	**Input Domain**
SHA256	String	32 bytes
DEX_PIXEL_0, ..., DEX_PIXEL_16383	Integer	{0, 1, ..., 255}
INTENT_0, ..., INTENT_99	Integer	{0, 1}
PERMISSION_0, ..., PERMISSION_99	Integer	{0, 1}
SYSCALL_0, ..., SYSCALL_399	Integer	{0, 1, ..., 123}
CLASS	Integer	{0 = Goodware, 1 = Malware}

Table 3.1: *Chimera multimodal deep learning Android malware detection method multimodal dataset structure*

To favor reproducibility and new research works, the dataset was published in the IEEE DataPort Dataset Storage and Dataset Search Platform (OLIVEIRA; SASSI, 2021d) under the Creative Commons Attribution 4.0 International (CC BY 4.0) licensing scheme (COMMONS, 2021), in which the the dataset can be copied, redistributed, remixed, transformed, and build upon, under the condition of its source being cited. As of October 2021, the author received several emails from researchers across the world interested in using the dataset for their research work.

The experimental platform used for data extraction, preprocessing, and transformation, as well as data mining processing, including models training, optimization, and evaluation, was based an Intel (R) Core (R) i7-8750H CPU @ 2.20GHz, 16 cores, 64 GB memory, and four Nvidia GeForce GTX 1080 Ti graphics cards. The GPU units were used to train, optimize, and evaluate the methods based on DL architectures, whereas the CPU units were used to train, optimize, and evaluate the traditional and Ensemble ML methods. Numpy (HARRIS et al., 2020), PyTorch (PASZKE et al., 2019), Pandas (TEAM, 2020), and Skorch (TIETZ et al., 2017) were used for the implementations. To favor reproducibility and repeatability, the source code implemented for training, optimization, and evaluation of all the methods will be available at (OLIVEIRA; SASSI, 2021b).

3.4 Data Selection, Preprocessing, and Transformation

The following sections present the implementation of the KDD process phases for data selection, preprocessing, and transformation for each one of the Chimera's subnetworks.

3.4.1 Chimera-S

Inspired by the work presented in (FEIZOLLAH et al., 2017; IDREES; RAJARAJAN, 2014; IDREES et al., 2017), Chimera-S (CHS) and Chimera makes use of manual feature engineering to implement an early fusion layer that combines 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 (FEIZOLLAH et al., 2017), 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.

As a baseline, it was extracted the top-100 Android Intents and the top-100 Android Permissions from Omnidroid's JSON files. Next, the 100-dimensional feature vectors from each modality were concatenated in the feature direction into a 200-dimensional feature vector to establish the early fusion layer. The SHA256 field was used to join the registers. The resulting dataset was saved 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 (ALPAYDIN, 2020) 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. Table 3.2 shows the Top-10 Android Intents as well as their presence among the dataset's instances.

Android Intent	**Presence (%)**
android.intent.action.main	99.847
android.intent.action.boot_completed	28.790
android.intent.action.view	19.400
android.intent.action.user_present	16.262
android.intent.action.package_added	14.466
android.intent.action.package_removed	10.436
android.intent.action.phone_state	4.451
android.intent.action.search	3.689
android.intent.action.package_replaced	3.240
android.intent.action.create_shortcut	3.223

Table 3.2: *Top-10 Android Intents and their presence in the dataset's instances.*

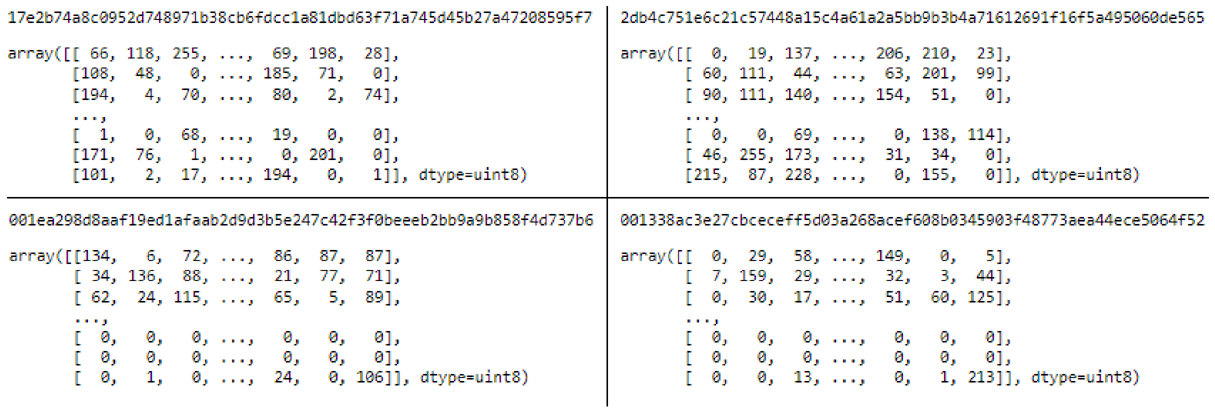
Table 3.3 shows the Top-10 Android Permissions as well as their presence among the dataset's instances. The leftmost column of Figure 3.2 summarizes the implementation steps of the KDD process stages Selection, Preprocessing, and Transformation for CHS and Chimera.

Android Permission	**Presence (%)**
android.permission.internet	95.606
android.permission.access_network_state	79.310
android.permission.write_external_storage	71.551
android.permission.read_phone_state	58.802
android.permission.access_wifi_state	49.679
android.permission.wake_lock	45.329
android.permission.access_coarse_location	34.957
android.permission.vibrate	34.906
android.permission.access_fine_location	32.683
android.permission.receive_boot_completed	27.812

Table 3.3: *Top-10 Android Permissions and their presence in the dataset's instances.*

3.4.2 CHIMERA-R

DEX files contain the executable code in Dalvik format. Similar to the Java Virtual Machine (JVM), the Dalvik VM translates DEX opcodes (bytecodes) into native CPU instructions. The DEX format provides a compact and optimized executable module (ENCK et al., 2011). The methods proposed by (HUANG; KAO, 2018; DING et al., 2020) make use of DEX bytecodes as images for malware detection and classification tasks using CNNs. Figure 3.3 depicts the raw bytes of DEX grayscale images representing two benign Android applications and two Android malware instances. Figure 3.4 depicts the DEX grayscale images representing the same instances.



```
17e2b74a8c0952d748971b38cb6fdcc1a81dbd63f71a745d45b27a47208595f7

array([[ 66, 118, 255, ...,  69, 198,  28],
       [108,  48,   0, ..., 185,  71,   0],
       [194,   4,  70, ...,  80,   2,  74],
       ...,
       [  1,   0,  68, ...,  19,   0,   0],
       [171,  76,   1, ...,   0, 201,   0],
       [101,   2,  17, ..., 194,   0,   1]], dtype=uint8)
```

```
2db4c751e6c21c57448a15c4a61a2a5bb9b3b4a71612691f16f5a495060de565

array([[  0,  19, 137, ..., 206, 210,  23],
       [ 60, 111,  44, ...,  63, 201,  99],
       [ 90, 111, 140, ..., 154,  51,   0],
       ...,
       [  0,   0,  69, ...,   0, 138, 114],
       [ 46, 255, 173, ...,  31,  34,   0],
       [215,  87, 228, ...,   0, 155,   0]], dtype=uint8)
```

```
001ea298d8aaf19ed1afaab2d9d3b5e247c42f3f0beeeb2bb9a9b858f4d737b6

array([[134,   6,  72, ...,  86,  87,  87],
       [ 34, 136,  88, ...,  21,  77,  71],
       [ 62,  24, 115, ...,  65,   5,  89],
       ...,
       [  0,   0,   0, ...,   0,   0,   0],
       [  0,   0,   0, ...,   0,   0,   0],
       [  0,   1,   0, ...,  24,   0, 106]], dtype=uint8)
```

```
001338ac3e27cbceceff5d03a268acef608b0345903f48773aea44ece5064f52

array([[  0,  29,  58, ..., 149,   0,   5],
       [  7, 159,  29, ...,  32,   3,  44],
       [  0,  30,  17, ...,  51,  60, 125],
       ...,
       [  0,   0,   0, ...,   0,   0,   0],
       [  0,   0,   0, ...,   0,   0,   0],
       [  0,   0,  13, ...,   0,   1, 213]], dtype=uint8)
```

Figure 3.3: *Raw bytes of 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.*

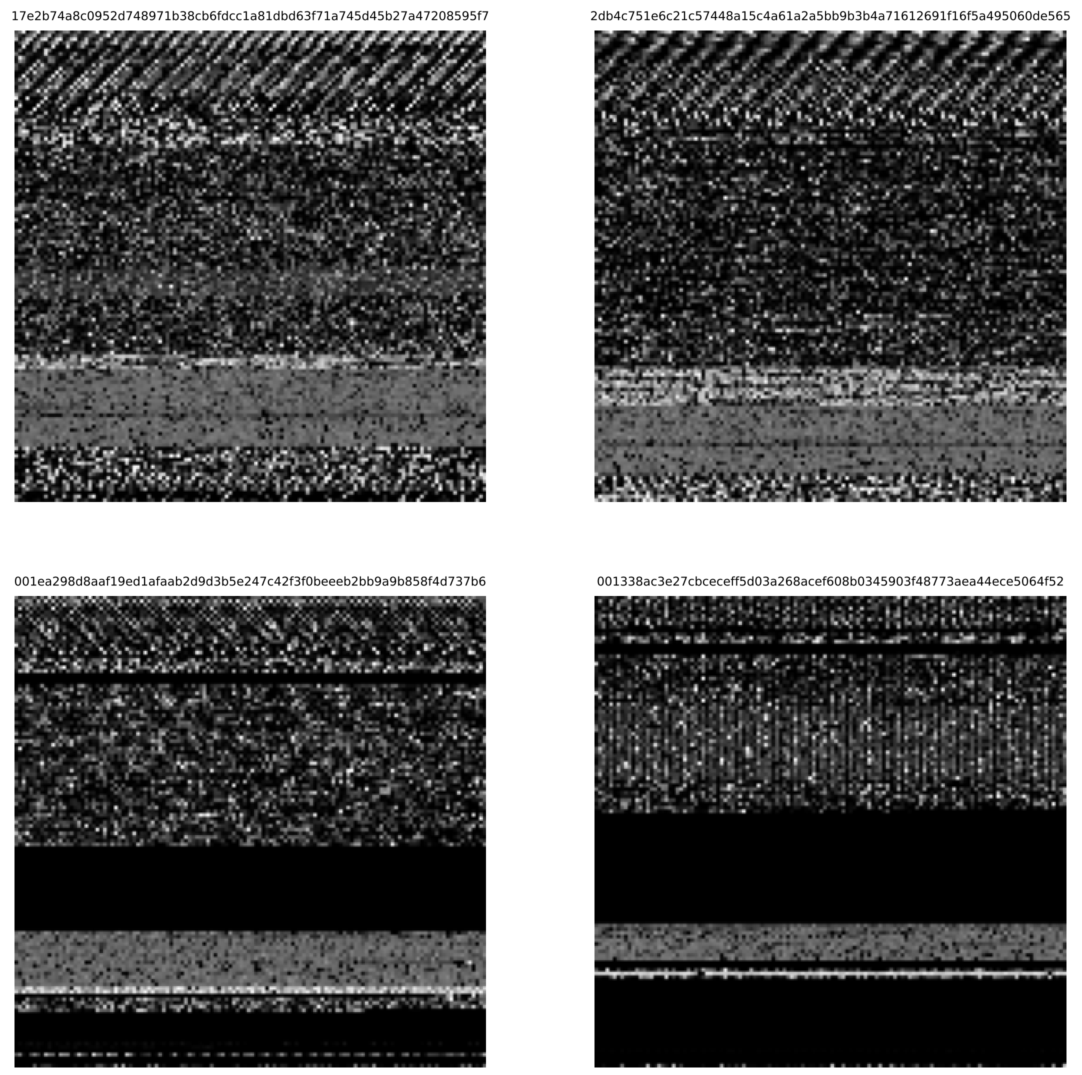

17e2b74a8c0952d748971b38cb6fdcc1a81dbd63f71a745d45b27a47208595f7
2db4c751e6c21c57448a15c4a61a2a5bb9b3b4a71612691f16f5a495060de565
001ea298d8aaf19ed1afaab2d9d3b5e247c42f3f0beeeb2bb9a9b858f4d737b6
001338ac3e27cbceceff5d03a268acef608b0345903f48773aea44ece5064f52

Figure 3.4: *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.*

Motivated by their work, Chimera-R (CHR) 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 unstructured data, their content was saved 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 (TURKOWSKI, 1990), and by applying a Scaling procedure (ALPAYDIN, 2020) to set the values of the features to the same scale, i.e. between 0 and 1, and a Standardization procedure (ALPAYDIN, 2020) 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. The image dimensions was chosen to be 1x128x128 since preliminary experiments indicated that smaller images promoted underfitting, whereas 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 (TURKOWSKI, 1990).

The rightmost column of Figure 3.2 summarizes the implementation steps of the KDD process stages Selection, Preprocessing, and Transformation for CHR and Chimera.

3.4.3 Chimera-D

Based on the work introduced by (XIAO et al., 2019), Chimera-D (CHD) and Chimera also depend on data collected using dynamic analysis (DA), particularly, the 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 3.5 shows the system call sequences of two benign Android applications and two Android malware instances, each containing 100-time steps.

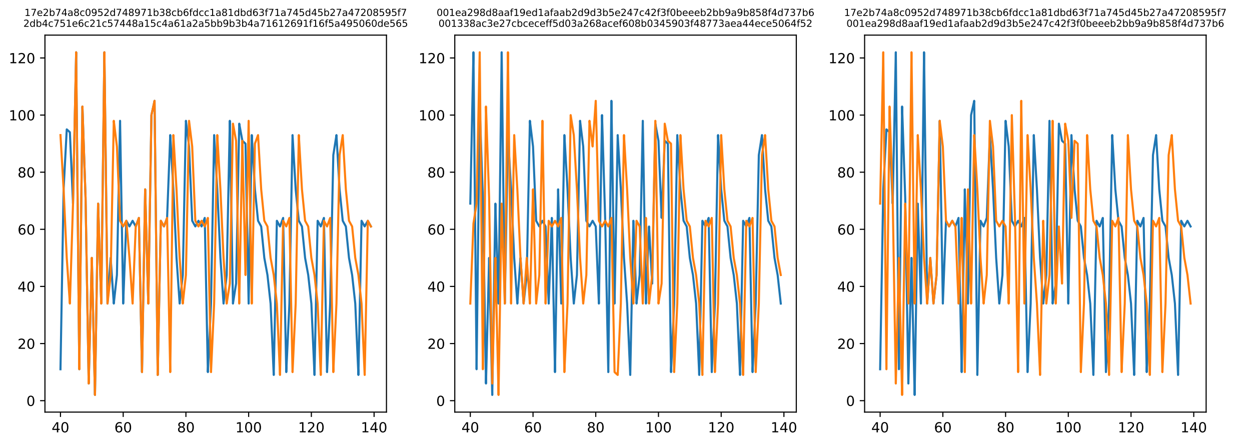


Figure 3.5: *System call sequences of two benign applications in the first column, two Trojan malware in the second column, and one benign application overlapped with one Trojan malware in the third column. The titles include the SHA256 hash of each instance. The x-axis represents the time step. The y-axis represents the system call number.*

Omnidroid includes the system calls sequences logged by the strace tool and saved into CSV files. To reduce noise and avoid loops, all the consecutive repeating system calls where removed. Then, sequences were trimmed 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 was performed by converting the integer encoded feature to its one-hot encoding representation 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 CHD, which is based on a TNN encoder.

Table 3.4 presents a list of ten system calls and their associated numbers in alphabetical order. The central column of Figure 3.2 summarizes the implementation steps of the KDD process stages Selection, Preprocessing, and Transformation for CHD and Chimera.

System Call Number	System Call Function Name	One-hot Enconding Representation
1	accept	(0, ..., 0, 0, 0, 0, 0, 0, 0, 0, 0, 1)
2	access	(0, ..., 0, 0, 0, 0, 0, 0, 0, 0, 1, 0)
3	bind	(0, ..., 0, 0, 0, 0, 0, 0, 0, 1, 0, 0)
4	brk	(0, ..., 0, 0, 0, 0, 0, 0, 1, 0, 0, 0)
5	cacheflush	(0, ..., 0, 0, 0, 0, 0, 1, 0, 0, 0, 0)
6	capset	(0, ..., 0, 0, 0, 0, 1, 0, 0, 0, 0, 0)
7	chdir	(0, ..., 0, 0, 0, 1, 0, 0, 0, 0, 0, 0)
8	chmod	(0, ..., 0, 0, 1, 0, 0, 0, 0, 0, 0, 0)
9	clock_gettime	(0, ..., 0, 1, 0, 0, 0, 0, 0, 0, 0, 0)
10	clone	(0, ..., 1, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Table 3.4: *System call number, System call function name in alphabetical order, and the 124-dimensional one-hot encoding representation of the System call number.*

3.5 Data Mining and Interpretation

Supervised deep learning is composed of several techniques for data mining of structured and unstructured data and can be used for classification and regression tasks (GOODFELLOW; BENGIO; COURVILLE, 2016). For the purpose of implementing the final Chimera MDL architecture for data mining presented in Figure 3.1, first it was necessary to break that down into its subnetworks to perform training, optimization, and evaluation independently. Each subnetwork architecture was implemented as a stand-alone unimodal DL method for Android malware detection, each one specialized in feature learning from a different data modality. CHS was designed to perform feature learning from relational data, represented as Android Intents & Permissions, extracted using SA. CHR was designed to perform feature learning from raw data, represented as DEX grayscale images, and CHD was designed to perform feature learning from temporal data, represented as system call sequences, extracted using DA. Figure 3.6 depicts each one the the Chimera's subnetworks implemented as stand-alone supervised DL methods.

As presented in Figure 3.6, all the subnetworks make use of the ReLU activation function to introduce nonlinearity, help mitigate the vanishing gradient problem, and speed up training convergence. Moreover, taking into account that CHS, CHR, CHD, and Chimera are binary classifiers that could be extended to multiclass classifiers in future work, the Softmax activation function was included after the output layer to encode the high-level feature representations into a probability distribution that can be used for binary classification. Consequently, the Cross-Entropy Loss function was introduced to quantify the training and evaluation errors during the learning process.

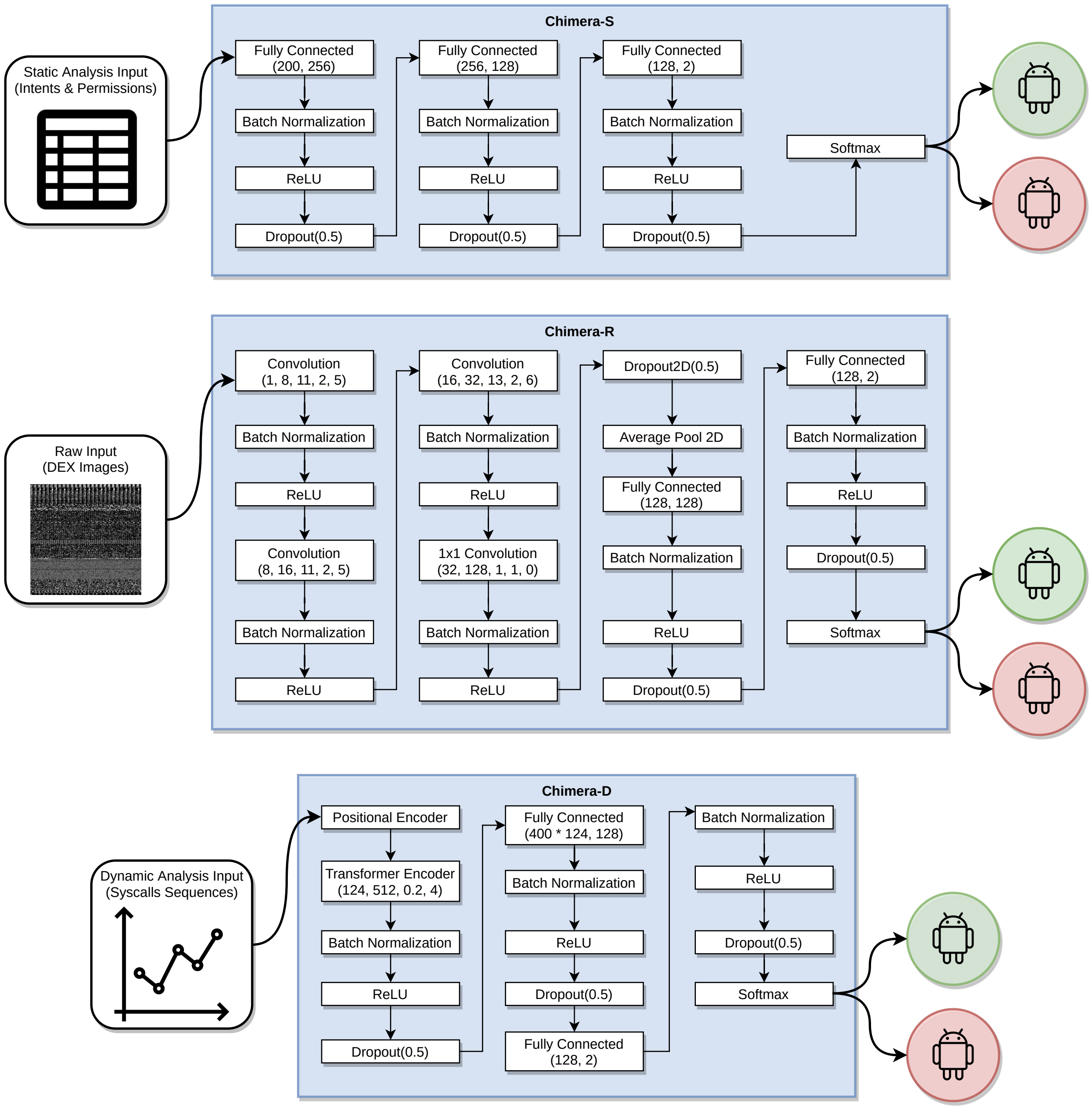


Figure 3.6: *Chimera multimodal deep learning Android malware detection method subnetworks for Static Analysis data (Chimera-S, CHS), raw data (DEX grayscale images) (Chimera-R, CHR), and Dynamic Analysis data (Chimera-D, CHD).*

Preliminary experiments indicated that both Batch Normalization and Dropout layers play an important role in mitigating overfitting and the training stability in CHS, CHR, CHD, and Chimera fully connected layers, and Batch Normalization plays a similar role for the convolutional layers of CHR and Chimera.

Finally, the Adam optimizer was chosen 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.

3.5.1 Evaluation Metrics

Since the Omnidroid dataset is a balanced dataset (MARTÍN; LARA-CABRERA; CAMACHO, 2019), it contains the same number of positive and negative instances, i.e., approximately the same number of malware and goodware instances. Thus, according to (THARWAT, 2020), the following performance metrics were chosen for evaluations and comparisons:

$$Precision = \frac{TP}{TP + FP} \tag{3.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.3}$$

$$AUCROC = \int_0^1 TPR \, \mathrm{d}(FPR) \tag{3.4}$$

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

TP is the number of correctly predicted positive instances, i.e., the value of actual class is positive and the value of predicted class is also positive.

TN is the number of correctly predicted negative instances, i.e., the value of actual class is negative and the value of predicted class is also negative.

FP is the number of incorrectly predicted positive instances, i.e., the value of actual class is negative and the value of predicted class is positive.

FN is the number of incorrectly predicted negative instances, i.e., the value of actual class is positive and the value of predicted class is negative.

Precision represents the ratio of correctly predicted positive instances to the total number of predicted positive instances.

Recall represents the the ratio of correctly predicted positive instances to the total of positive instances.

Accuracy represents the ratio of correctly predicted instances to the total of instances.

AUC ROC represents the capability of the model in distinguishing between the two classes. The higher the AUC ROC, the better the model is at prediction.

Figure 3.7 presents a visualization of the metrics Precision, Recall, and Accuracy for two classes, red class and blue class. The black circle represents a binary classifier predictions. The green region indicates correctly classified instances, whereas the red regions indicate misclassified instances. These results can be summarized by the Confusion Matrix.

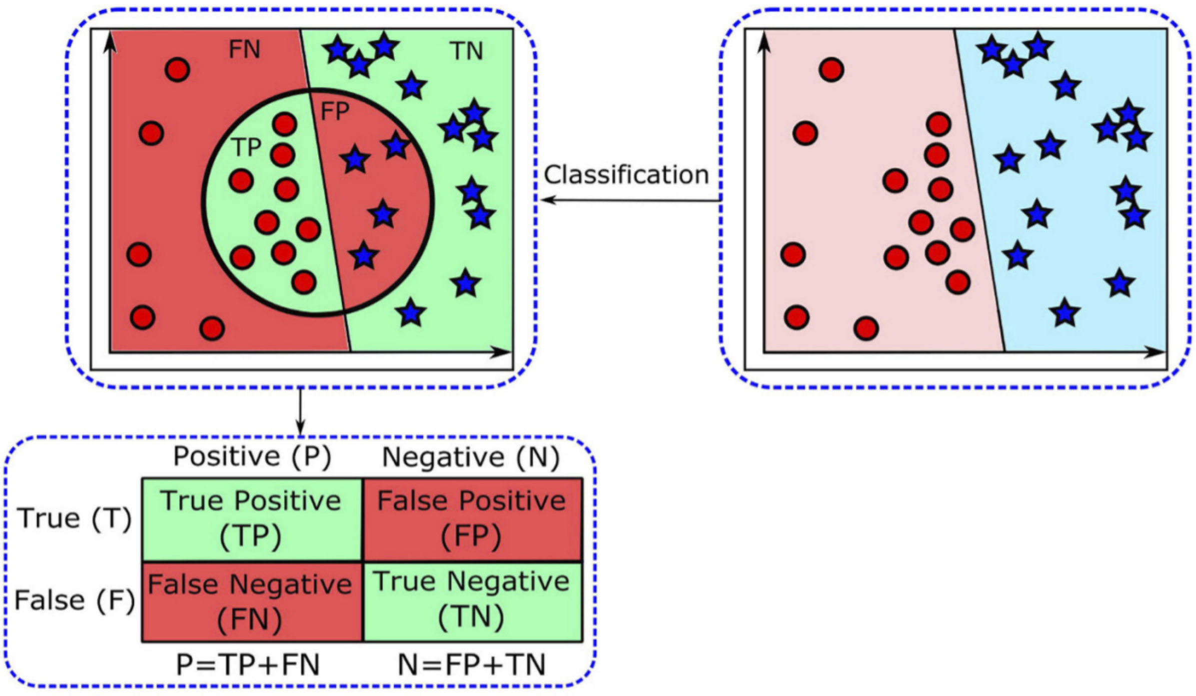


Figure 3.7: *Visualization of the metrics Precision, Recall, and Accuracy for two classes and the associated Confusion Matrix.*

Source : Adapted by the author from (THARWAT, 2020)

3.5.2 Model Selection

In order to choose the best architectures for Chimera-S (CHS), Chimera-R (CHR), Chimera-D (CHD), and Chimera, model selection (or hyperparameter tuning) was performed using grid search cross-validation with 10-fold cross-validation to estimate the generalization error (ARLOT; CELISSE et al., 2010). 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 dissertation, the Accuracy metric was chosen 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. Figure 3.8 illustrates the 10-fold cross-validation procedure and Figure 3.9 depicts the model selection process.

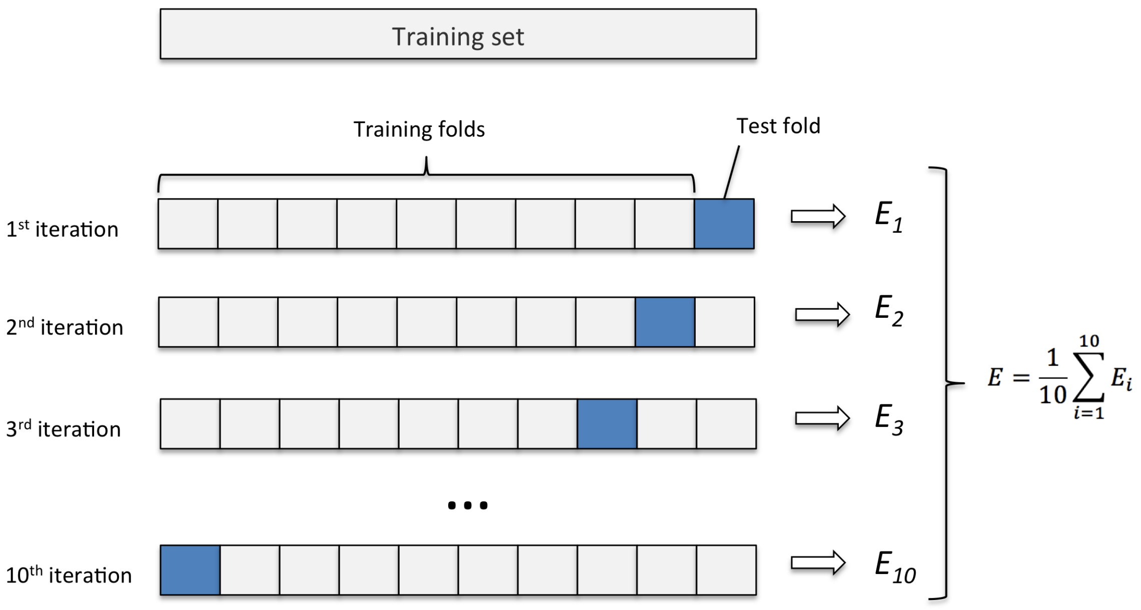


Figure 3.8: *10-fold cross-validation procedure used to estimate the models' performance.*

Source : Adapted by the author from (NIU et al., 2018)

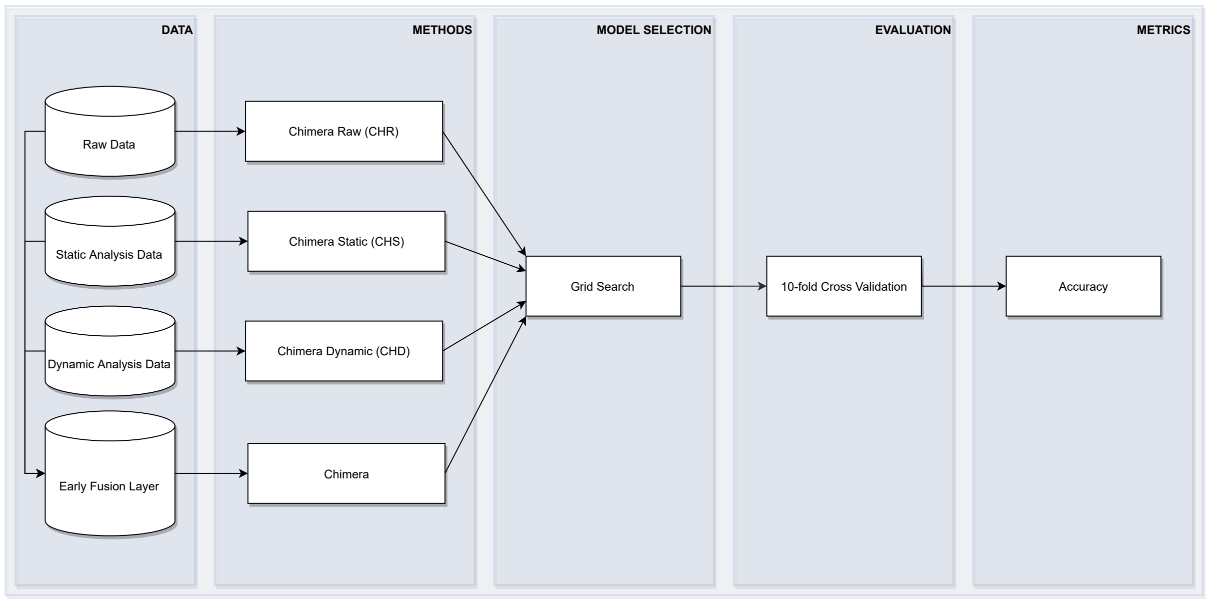


Figure 3.9: *Chimera multimodal deep learning Android malware detection method and its subnetworks model selection process.*

3.5.3 Chimera-S

As depicted in Figure 3.6, CHS introduces a DNN architecture composed of 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. Dropout and Batch Normalization layers were included 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

Figure 3.6 presents the optimized CHS architecture. Figure 3.10 depicts the training loss as a function of the number of epochs of the optimized CHS model.

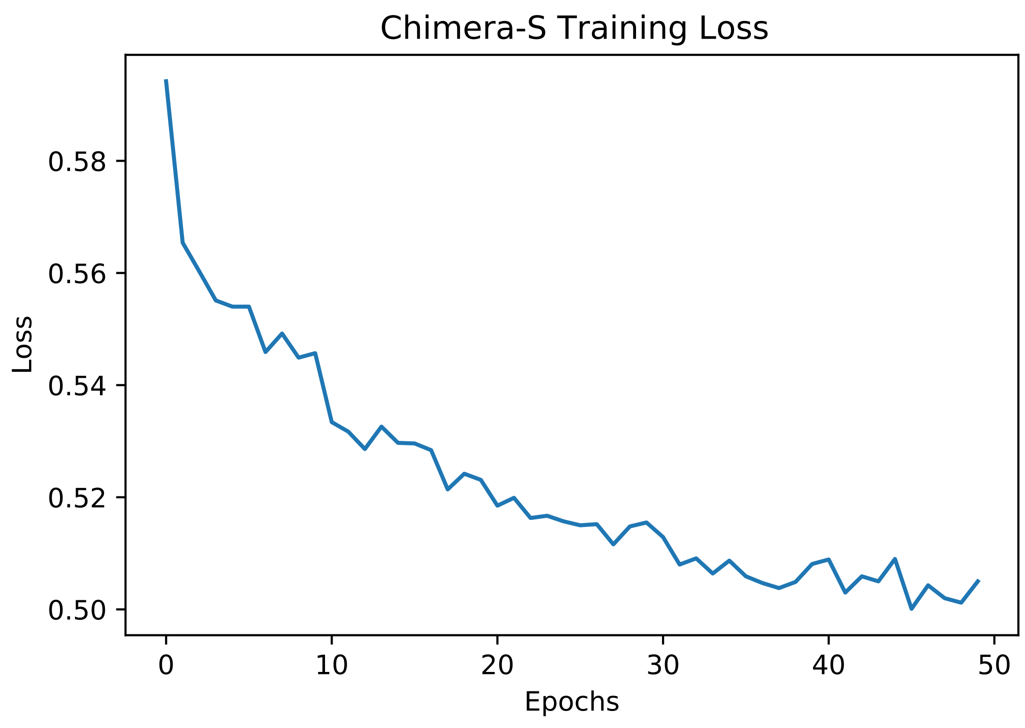


Figure 3.10: *Chimera-S training loss as a function of the number of epochs.*

3.5.4 CHIMERA-R

As depicted in Figure 3.4, 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, (HUANG; KAO, 2018) proposed a CNN architecture for Android malware classification using DEX grayscale images, and (DING et al., 2020) 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 (LECUN et al., 1998) 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 (HUANG; KAO, 2018; DING et al., 2020), where CNNs were used for feature learning. The main novelty is the introduction of a new CNN architecture inspired by the Residual Networks (ResNet) architecture (HE et al., 2016). As shown in Figure 3.6, CHR 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 (GOODFELLOW; BENGIO; COURVILLE, 2016). 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 (LIN; CHEN; YAN, 2013). 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 (GOODFELLOW; BENGIO; COURVILLE, 2016) 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. Figure 3.11 presents a simplified, out of scale representation of the Chimera-R CNN architecture.

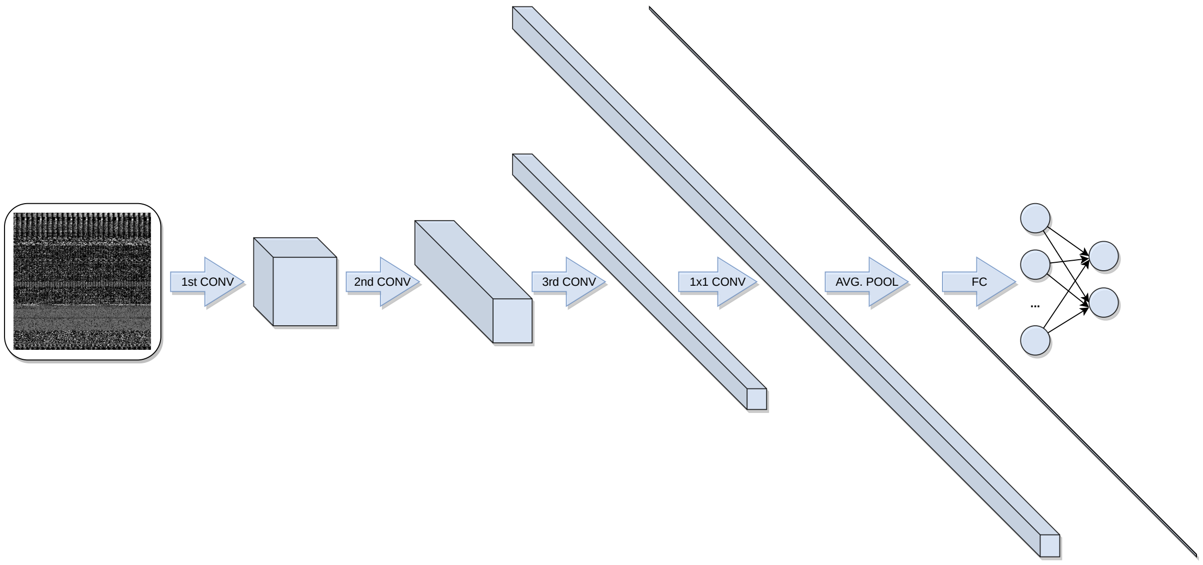


Figure 3.11: *Chimera-R Convolutional Neural Network architecture (simplified and out of scale).*

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 nonlinsearity and prevent the vanishing gradient problem.

Similar to CHS, to mitigate overfitting, Dropout and Batch Normalization layers were included between the fully connected layers; However, contrary to what was verified for CHS, preliminary 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 CHS, 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

Figure 3.6 presents the optimized CHR architecture. Figure 3.12 depicts the training loss as a function of the number of epochs of the optimized CHR model.

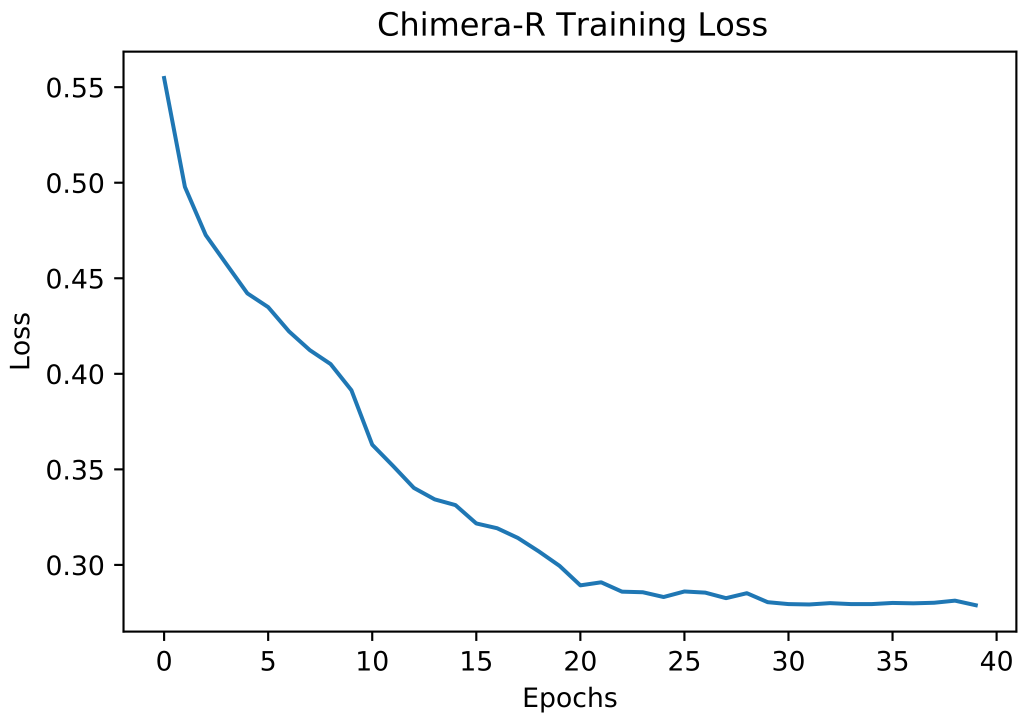


Figure 3.12: *Chimera-R training loss as a function of the number of epochs.*

3.5.5 Chimera-D

As presented in Figure 3.5, 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, (XIAO et al., 2019) proposed an LSTM architecture to implement a neural probabilistic language model for Android malware detection using system call sequences.

This dissertation is based on a different architecture for sequence learning, the TNN (VASWANI et al., 2017). TNN is a state-of-the-art encoder-decoder DL architecture designed to handle sequential data, such as natural language. Unlike LSTMs, TNNs do not need to process sequential data in order. Due to this feature, TNNs facilitate parallelization during training time. Also, TNNs 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 presented in Figure 3.6, CHD 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 TNN encoder layer for sequence learning and temporal feature extraction. Finally, a DNN is used for Android malware detection.

Its important to notice that CHD and Chimera only make use of the encoder part of the TNN. The TNN 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 TNN encoder. The second layer is composed of 128 neurons, and the output layer contains two neurons, followed by a Softmax activation function.

Similar to CHS and CHR, Dropout and Batch Normalization layers were included after the TNN encoder and the fully connected layers to mitigate overfitting and increase training stability. The ReLU activation function was used in the TNN encoder and in CHD to introduce nonlinearity. To train CHD, the learning rate warm-up strategy was chosen, which increases 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 TNNs (VASWANI et al., 2017).

The following hyperparameters were considered for model selection:

- Number of neurons in the TNN feedforward layer: {128, 256, 512, 1024}
- Number of neurons in the DNN hidden layer: {128, 256, 512}
- TNN 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 TNN feedforward layer: 512
- Number of neurons in the DNN hidden layer: 128

- TNN 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

Figure 3.6 presents the optimized CHD architecture. Figure 3.13 depicts the training loss as a function of the number of epochs of the optimized CHD model.

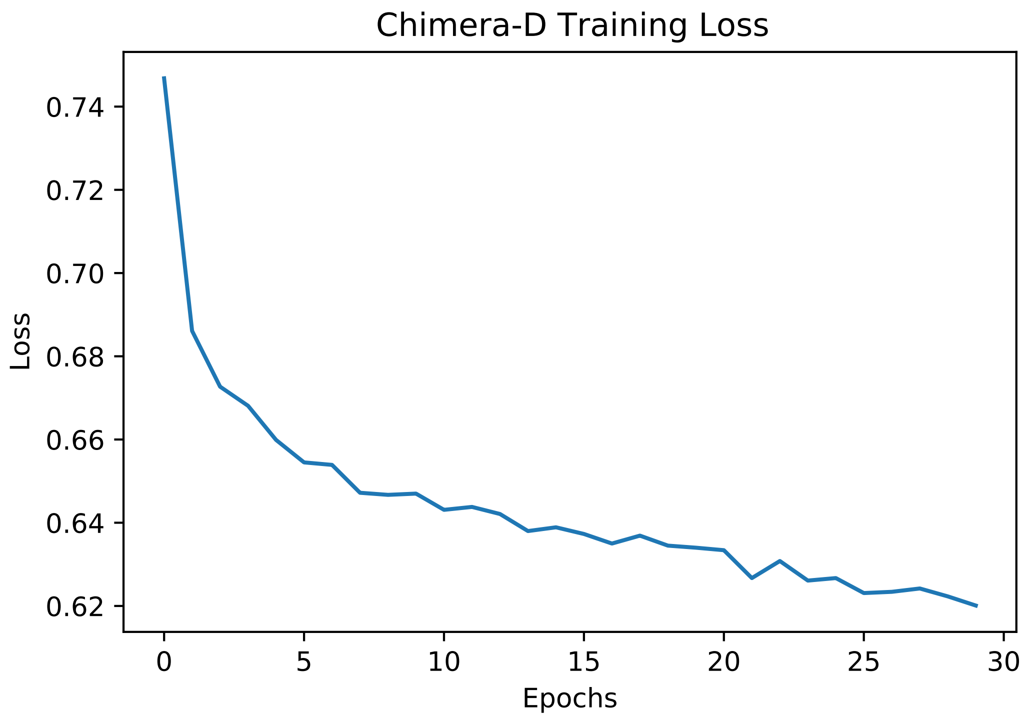


Figure 3.13: *Chimera-D training loss as a function of the number of epochs.*

3.5.6 Chimera

As presented in Figure 3.1 and in Algorithm 1, once the subnetworks CHS, CHR, and CHD have forward propagated their inputs, a shared representation layer is implemented by concatenating their results in the feature direction and passed to the last Chimera's DNN classifier for Android malware detection.

Similar to what was verified by (KIM et al., 2019; GIBERT; MATEU; PLANES, 2020), it was found out that training Chimera as a single model resulted in underfitting one subnetwork

Algorithm 1: Chimera Multimodal Deep Learning Method

```
1 $X \leftarrow LoadMultimodalData()$
2 $X_{raw}, X_{intents}, X_{permissions}, X_{dynamic} \leftarrow Split(X)$
3 $X_{static} \leftarrow X_{intents} \oplus X_{permissions}$
4 $CHR_{subnetwork}, CHS_{subnetwork}, CHD_{subnetwork} \leftarrow LoadPretrainedWeights()$
5 $\hat{Y}_{raw} \leftarrow CHR_{subnetwork}(X_{raw})$
6 $\hat{Y}_{static} \leftarrow CHS_{subnetwork}(X_{static})$
7 $\hat{Y}_{dynamic} \leftarrow CHD_{subnetwork}(X_{dynamic})$
8 $\hat{Y}_{multimodal} \leftarrow \hat{Y}_{raw} \oplus \hat{Y}_{static} \oplus \hat{Y}_{dynamic}$
9 $\hat{Y} \leftarrow Softmax(Chimera_{DNN}(\hat{Y}_{multimodal}))$
```

and overfitting of the other subnetworks. Taking that into account, CHS, CHR, and CHD were implemented as fully-fledged, independent unimodal DL methods, as shown in Figure 3.6, and trained separately using the optimized hyperparameters found in the Sections 3.5.3, 3.5.4, and 3.5.5 respectively. Then, Transfer Learning (LONG; SHELHAMER; DARRELL, 2015) was used to combine the trained models into the final Chimera model.

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 loaded and frozen, i.e. kept constant, 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. Figure 3.14 illustrates the Chimera architecture with the frozen subnetworks CHR, CHS, and CHD adapted for Transfer Learning followed by a DNN composed of trainable weights to be used in the classifier. Algorithm 1 describes the implementation steps of Transfer Learning.

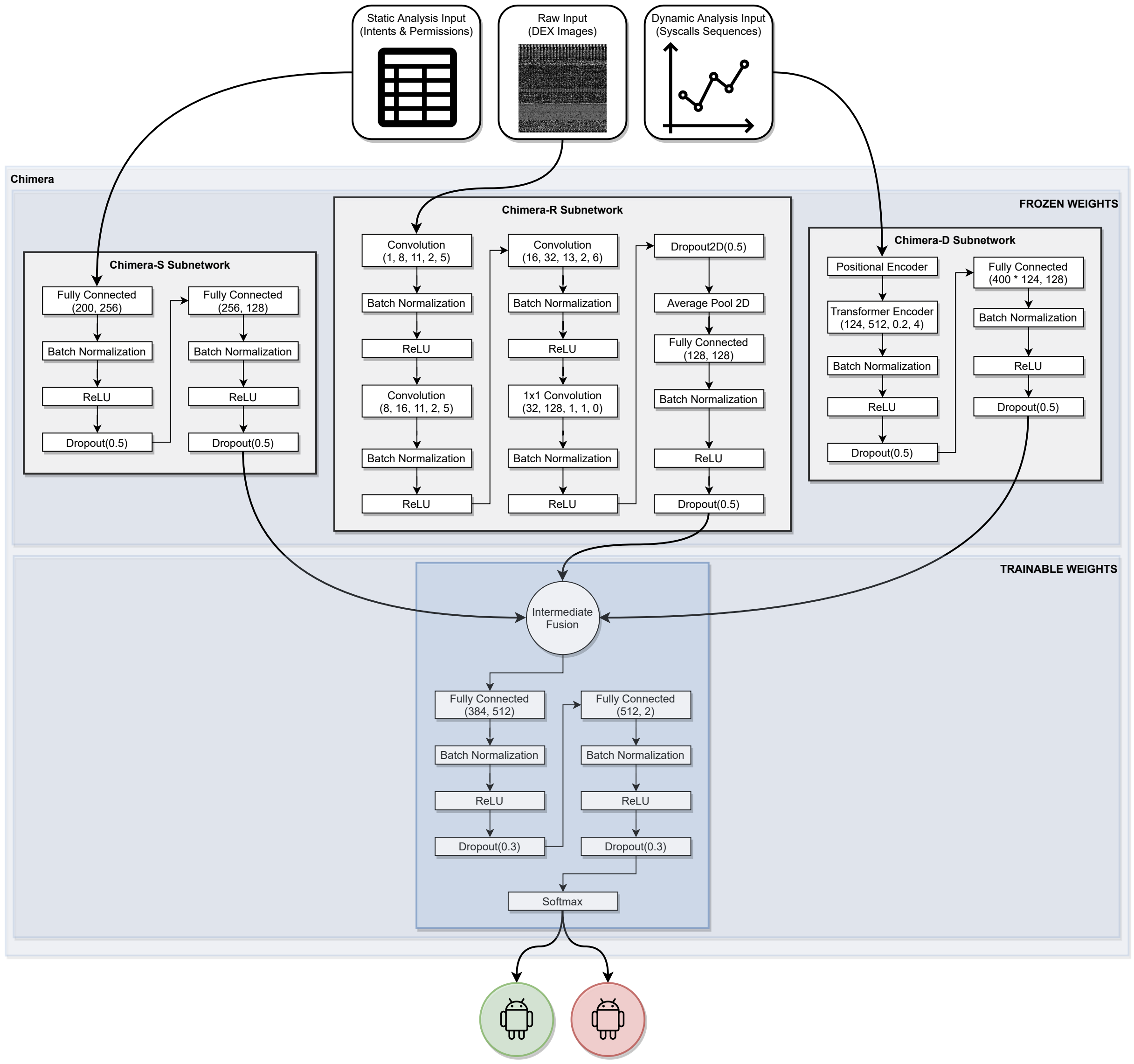


Figure 3.14: *Chimera multimodal deep learning Android malware detection method transfer learning technique for model training.*

Finally, as ilustrated in Figure 3.1, 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 nonlinearity. Dropout and Batch Normalization layers were included between the fully connected layers to mitigate overfitting. Moreover, similar to CHS and CHR, 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

Figure 3.1 presents the optimized Chimera architecture. Figure 3.15 depicts the training loss as a function of the number of epochs of the optimized Chimera model.

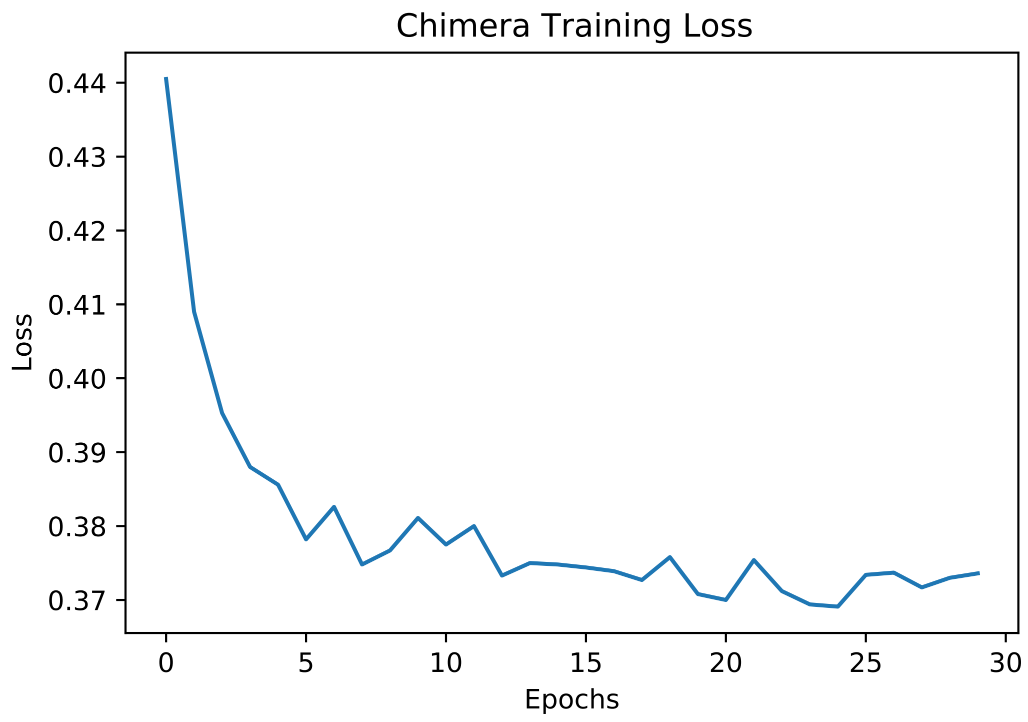


Figure 3.15: *Chimera training loss as a function of the number of epochs.*

3.6 Performance Evaluation

The evaluation process was broken down into the following phases:

1. Evaluation phase 1:
   (a) Chimera
   (b) Chimera-R
   (c) Extra Trees
   (d) Random Forest
   (e) Logistic Regression
   (f) Support Vector Machines
   (g) Decision Tree
   (h) K-Nearest Neighbors
   (i) Naive Bayes
2. Evaluation phase 2:
   (a) Chimera
   (b) Chimera-S
   (c) Extra Trees
   (d) Random Forest
   (e) Logistic Regression
   (f) Support Vector Machines
   (g) Decision Tree
   (h) K-Nearest Neighbors
   (i) Naive Bayes
3. Evaluation phase 3:
   (a) Chimera
   (b) Chimera-D

   (c) Extra Trees
   (d) Random Forest
   (e) Logistic Regression
   (f) Support Vector Machines
   (g) Decision Tree
   (h) K-Nearest Neighbors
   (i) Naive Bayes

4. Evaluation phase 4:
   (a) Chimera
   (b) Extra Trees
   (c) Random Forest
   (d) Logistic Regression
   (e) Support Vector Machines
   (f) Decision Tree
   (g) K-Nearest Neighbors
   (h) Naive Bayes

As depicted in Figure 3.16, evaluation phase 1, represented by the blue and black arrows, takes into consideration the classical ML methods, the Ensemble ML methods, and the independent, full-fledged Android malware detection method CHR, using raw data - DEX grayscale images, and Chimera, using multimodal data. The objective of phase 1 evaluation is to compare the performance of traditional ML methods, Ensemble ML methods, and the unimodal DL method CHR using only one data modality, i.e. raw data. Chimera's performance evaluation results were included for comparison purporses.

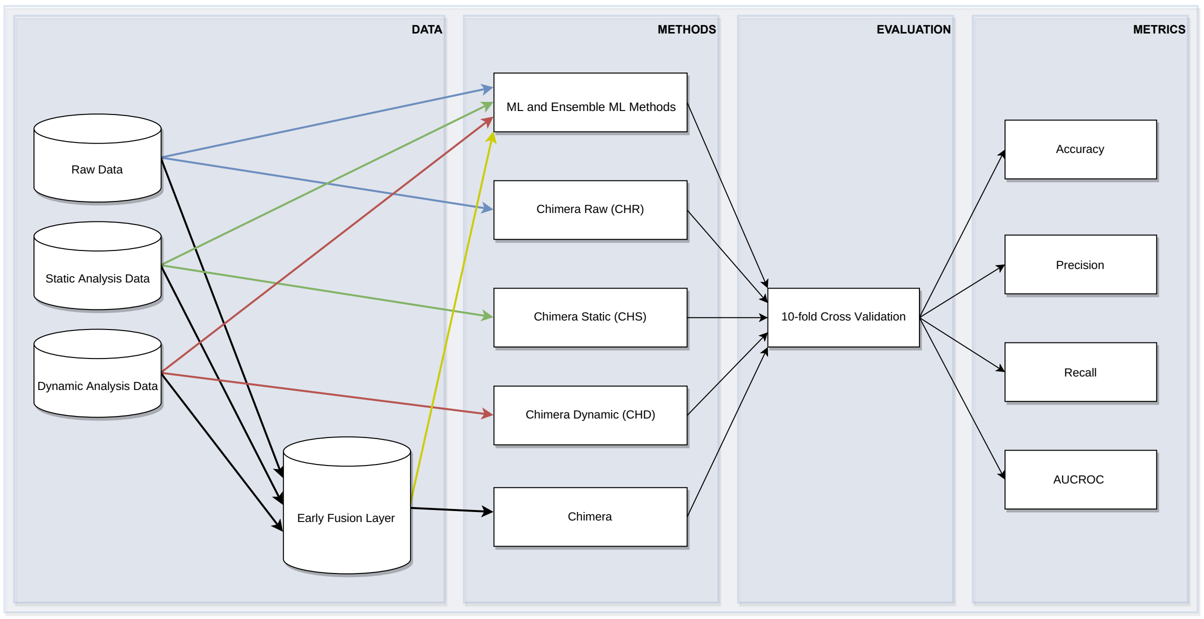


Figure 3.16: *Chimera multimodal deep learning Android malware detection method, its subnetworks, classical ML methods, and Ensemble ML methods performance evaluation process.*

As presented in Figure 3.16, evaluation phase 2, represented by the green and black arrows, takes into consideration the classical ML methods, the Ensemble ML methods, and the independent, full-fledged Android malware detection method CHS, using static analysis data, and Chimera, using multimodal data. The objective of phase 2 evaluation is to compare the performance of traditional ML methods, Ensemble ML methods, and the unimodal DL method CHS using only one data modality, i.e. SA data. Chimera's performance evaluation results were included for comparison purposes.

As shown in Figure 3.16, evaluation phase 3, represented by the red arrows and black arrows, takes into consideration the classical ML methods, the Ensemble ML methods, and the independent, full-fledged Android malware detection method CHD, using dynamic analysis data. The objective of phase 3 evaluation is to compare the performance of traditional ML methods, Ensemble ML methods, and the unimodal CHD using only one data modality, i.e. DA data. Chimera's performance evaluation results were included for comparison purposes.

As illustrated in Figure 3.16, evaluation phase 4, represented by the black and yellow arrows, takes into consideration the classical ML methods, the Ensemble ML methods, and the multimodal DL method Chimera, using multimodal data. The objective of phase 4 evaluation is to compare the performance of traditional ML methods, Ensemble ML methods, and the MDL method Chimera using multiples data modalities at once, i.e., raw data, SA data, and DA data.

Intending to carry out the evaluations, 10-fold cross-validation was performed on the optimized models found in Sections 3.5.3, 3.5.4, 3.5.5, and 3.5.6. Moreover, the following metrics were considered: Accuracy, Precision, Recall, and AUC ROC, as defined in Section 3.5.1.

CHAPTER 4

RESULTS AND DISCUSSION

In the context of this dissertation, the Accuracy metric defined by the Equation 3.3 represents the total number of correct detections over the total number of instances. Since the Omnidroid dataset is a balanced dataset, 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 achieves the highest possible Accuracy. Therefore, the higher the Accuracy, the better is the overall method's performance. The Precision metric (See Equation 3.1) represents the total number of correct malware detections over the total number of malware detections. According to Equation 3.1, if the number of false positives is equal to zero, then the method achieves the highest possible Precision. The Recall metric, defined by the Equation 3.2), represents the total number of correct malware detections over the total number of malware instances. If the number of false negatives is equal to zero, then the method achieves the highest possible Recall. It is 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 and can be solved by just white listing the application; on the other hand, a false negative means that a malware was detected as a goodware, thus, bypassing the detection method. The AUC ROC metric, defined by the Equation 3.4, is used to summarize a binary classifier's performance as its discrimination threshold is varied. A high AUC ROC indicates that the method can achieve high values for Precision or Recall by varying the classification threshold accordingly. Finally, the Fit Time refers to the amount of time (in seconds) necessary to train each method for a given dataset.

As depicted in Tables 4.1, 4.2, 4.3, 4.4, the results of 10-fold cross-validation using static analysis data (Android Intents & Permissions), raw data (DEX grayscale images), dynamic analysis data (system call sequences), and multimodal data show that Chimera achieved the best performance for all the considered metrics except the Fit Time.

Classifier	Accuracy	Precision	Recall	AUC ROC	Fit Time (s)
Chimera	**0.909 ± (0.001)**	**0.948 ± (0.003)**	**0.863 ± (0.004)**	**0.972 ± (0.000)**	237.439
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	0.831 ± (0.002)	**0.889 ± (0.003)**	0.750 ± (0.004)	0.908 ± (0.002)	47.694
Multi-layer Perceptron	0.823 ± (0.003)	0.842 ± (0.005)	0.789 ± (0.004)	0.883 ± (0.002)	37.947
K-Nearest Neighbors	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**
Logistic Regression	0.792 ± (0.002)	0.803 ± (0.002)	0.764 ± (0.003)	0.866 ± (0.002)	2.535
Support Vector Machines	0.788 ± (0.002)	0.799 ± (0.003)	0.763 ± (0.003)	0.860 ± (0.003)	36.207
Naive Bayes	0.585 ± (0.002)	0.859 ± (0.008)	0.189 ± (0.003)	0.783 ± (0.003)	**0.325**

Table 4.1: *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.*

Classifier	Accuracy	Precision	Recall	AUC ROC	Fit Time (s)
Chimera	**0.909 ± (0.001)**	**0.948 ± (0.003)**	**0.863 ± (0.004)**	**0.972 ± (0.000)**	237.439
Chimera-R	**0.801 ± (0.002)**	**0.816 ± (0.004)**	0.777 ± (0.004)	**0.885 ± (0.001)**	177.506
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
Multi-layer Perceptron	0.758 ± (0.003)	0.762 ± (0.003)	0.748 ± (0.006)	0.841 ± (0.003)	798.400
Logistic Regression	0.701 ± (0.002)	0.699 ± (0.003)	0.707 ± (0.005)	0.776 ± (0.002)	**50.412**
Support Vector Machines	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-Nearest Neighbors	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 4.2: *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.*

Classifier	Accuracy	Precision	Recall	AUC ROC	Fit Time (s)
Chimera	**0.909 ± (0.001)**	**0.948 ± (0.003)**	**0.863 ± (0.004)**	**0.972 ± (0.000)**	237.439
Chimera-D	**0.590 ± (0.003)**	**0.586 ± (0.004)**	0.573 ± (0.006)	0.625 ± (0.004)	203.709
Multi-layer Perceptron	0.588 ± (0.004)	0.568 ± (0.005)	0.644 ± (0.006)	**0.726 ± (0.004)**	669.931
Logistic Regression	0.583 ± (0.005)	0.582 ± (0.005)	0.543 ± (0.007)	0.620 ± (0.005)	28.229
Random Forest	0.569 ± (0.003)	0.566 ± (0.003)	0.539 ± (0.004)	0.597 ± (0.004)	39.282
Extra Trees	0.564 ± (0.003)	0.560 ± (0.003)	0.530 ± (0.004)	0.586 ± (0.004)	59.573
K-Nearest Neighbors	0.543 ± (0.005)	0.536 ± (0.005)	0.538 ± (0.006)	0.559 ± (0.006)	14.688
Support Vector Machines	0.543 ± (0.006)	0.538 ± (0.007)	0.524 ± (0.006)	0.565 ± (0.007)	262.466
Decision Tree	0.527 ± (0.004)	0.521 ± (0.004)	0.504 ± (0.004)	0.526 ± (0.004)	**14.229**
Naive Bayes	0.507 ± (0.008)	0.506 ± (0.009)	**0.915 ± (0.057)**	0.522 ± (0.013)	**4.034**

Table 4.3: *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.*

Classifier	Accuracy	Precision	Recall	AUC ROC	Fit Time (s)
Chimera	**0.909 ± (0.001)**	**0.948 ± (0.003)**	**0.863 ± (0.004)**	**0.972 ± (0.000)**	237.439
Extra Trees	**0.804 ± (0.002)**	**0.811 ± (0.002)**	0.785 ± (0.003)	**0.892 ± (0.002)**	121.260
Multi-layer Perceptron	0.799 ± (0.005)	0.803 ± (0.009)	0.787 ± (0.003)	0.880 ± (0.004)	295.086
Random Forest	0.789 ± (0.002)	0.793 ± (0.003)	0.772 ± (0.004)	0.877 ± (0.002)	138.766
Logistic Regression	0.787 ± (0.003)	0.782 ± (0.003)	0.786 ± (0.005)	0.873 ± (0.002)	57.985
Support Vector Machines	0.772 ± (0.003)	0.768 ± (0.003)	0.771 ± (0.005)	0.852 ± (0.003)	695.069
Decision Tree	0.739 ± (0.003)	0.735 ± (0.003)	0.734 ± (0.004)	0.739 ± (0.003)	439.701
K-Nearest Neighbors	0.672 ± (0.002)	0.627 ± (0.002)	**0.824 ± (0.004)**	0.766 ± (0.003)	**17.126**
Naive Bayes	0.527 ± (0.001)	0.804 ± (0.010)	0.053 ± (0.002)	0.520 ± (0.001)	**20.711**

Table 4.4: *10-fold cross-validation results of different methods on multimodal data. The text in bold indicates the best mean values for each metric. The dark gray row indicates the best performing method.*

Chimera implements a shared representation layer for Android malware detection; thus, it takes advantage of the features learned from 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 modeling of Chimera-D, and (3) The DEX images resampling of Chimera-R. By combining manual feature engineering and automatic feature engineering, DL models can extract more useful features to be used in the detection task; therefore achieving higher performances.

Regarding the Fit Time metric, it is worth noting that in the evaluations presented in Tables 4.1, 4.2, and 4.3, Chimera used a dataset composed of multiple data modalities, whereas the other methods used a dataset composed of a single modality each, thus, Chimera deals with a larger volume of data, consequently increasing the Fit Time. In order to perform a direct comparison between the Fit Time of Chimera and the other methods, as presented is Figure 3.16, the multimodal dataset was also used to train and evaluate the other methods. Table 4.4 presents the evaluation results considering the multimodal dataset.

The results presented in Table 4.4 show that although Chimera is MDL method, its Fit Time is significantly lower than the Multi-layer perceptron, Support Vector Machines, and Decision Tree methods. One reason for that the dimensionality reduction of the input vectors promoted by Chimera's subnetworks. On the one hand, the input dimension for the ML methods is numerically equivalent to the number of input features of the multimodal dataset presented in Table 3.1, i.e. 16986. On the other hand, as described in 3.5.6, Chimera's input dimension is defined by the number of features in the intermediate fusion layer, which is equal to 512. The subnetworks CHS, CHR, and CHD were able to learn useful features for Android malware detection and the intermediate fusion layer combined those features in a lower dimensional input vector for Chimera, thus reducing the input dimensionality and the Fit Time. It is also worth noting that the ML methods' Accuracy, Precision, Recall, and AUC ROC are significantly lower when considering the multimodal dataset. This result indicates that, although more input information was used, the ML methods were not able to extract more knowledge from that information if compared to Chimera.

The results presented in Table 4.1 show that Chimera-S achieves the 4th best Accuracy, the 2nd best Recall, and the 2nd best AUC ROC. The results show that the Ensemble ML methods Random Forest and Extra Trees were very efficient in learning the patterns from the Static Analysis data, which is structured by nature. Also, their Fit Times were lower than Chimera-S'. The data modality used in this evaluation was subjected to manual feature engineering by choosing the most significant features for Android malware detection (See Section 3.4.1), nevertheless, Chimera-S, based on a DNN was able to extract useful features for Android malware detection, thus achieving higher Accuracy, Precision, and AUC ROC than the Multi-layer Perceptron and the other ML methods.

The results presented in Table 4.2 show that Chimera-R outperforms all the ML methods for almost all the performance metrics, except the Fit Time. Regarding the Recall metric, a more detailed analysis shows that, although the K-Nearest Neighbors method achieved the highest Recall, it also achieved low Accuracy, Precision, and AUC ROC, which indicates that the method was biased towards the malicious samples and were not able to generalize well, therefore, the Recall metric is not a reliable. Chimera-R is based on CNNs, which are specialized DL architectures for image processing. Since the input data for Chimera-R are DEX grayscale images, the CNN architecture was able to extract useful features for Android malware detection, thus significantly increasing the method's performance. As opposed to DL methods, classical ML methods rely on manual feature engineering to achieve higher performances. Since the data modality used for this evaluation was subjected to mininal feature engineering (See Section 3.4.2), the ML methods achieved lower performance. Moreover, a manual feature engineering step to increase the ML methods' performance would require additional research since in general its not clear how to map image patterns to malicious patterns in malware instances. The results presented by this evaluation clearly show the advantage of DL models over traditional ML models, i.e., the capacity of automatic feature learning from unstructured data.

The results presented in Table 4.3 show that Chimera-D outperforms all the ML methods for all the performance metrics except for the Recall, AUC ROC, and the Fit Time. Regarding the Recall metric, a more detailed analysis shows that, although the Naive-Bayes method achieved the highest Recall, it also achieved low Accuracy, Precision, and AUC ROC - close to a dummy classifier indeed - which indicates that the method was strongly biased towards the malicious samples and were not able to generalize well, therefore, the Recall metric is not a reliable. Concerning the AUC ROC metric, the Multi-layer Perceptron achieve the 2nd highest value, which indicates that the method has a higher capability in distinguishing between the two classes, and probably there is a classifier threshold in which the methods achieves better Precision or Recall. The results of this evaluation indicates that the TNN architecture was able to extract useful features for Android malware detection, although in a lower degree than Chimera-R. A possible reason for that is that the sequences of system calls do not present a high discriminative power compared to the DEX. Another reason can be related to the size of the sequences used during manual feature engineering (See Section 3.4.2). The results presented by this evaluation clearly show the advantage of using a MDL method such as Chimera, which does not depend only on a single data modality, on the contrary, it combines multiple data modalities to learn the best features to accomplish the task at hand.

It is important to notice that Chimera achieved higher performance than its subnetworks evaluated independently, as shown in Tables 4.1, 4.2, and 4.3. The reason for that is because Chimera learned to correlate the features learned by its subnetworks from multiple data modalities, consequently increasing the number of true positives and the true negatives, and decreasing the number of false positives and false negatives, thus increasing its Accuracy, Precision, Recall, and AUC ROC.

Finally, as we can see in Figure 4.1, the ROC curves of all the considered methods for the multimodal dataset are depicted in the same plot for easier comparison. Clearly, Chimera has the overall best ROC and ROC AUC, which indicates that the method has the highest capability in distinguishing between the two classes, consequently, the classification threshold could be tuned to achieve better Precision - or a lower number of false positives, or better Recall - or a lower number of false negatives, depending on the desired outcome.

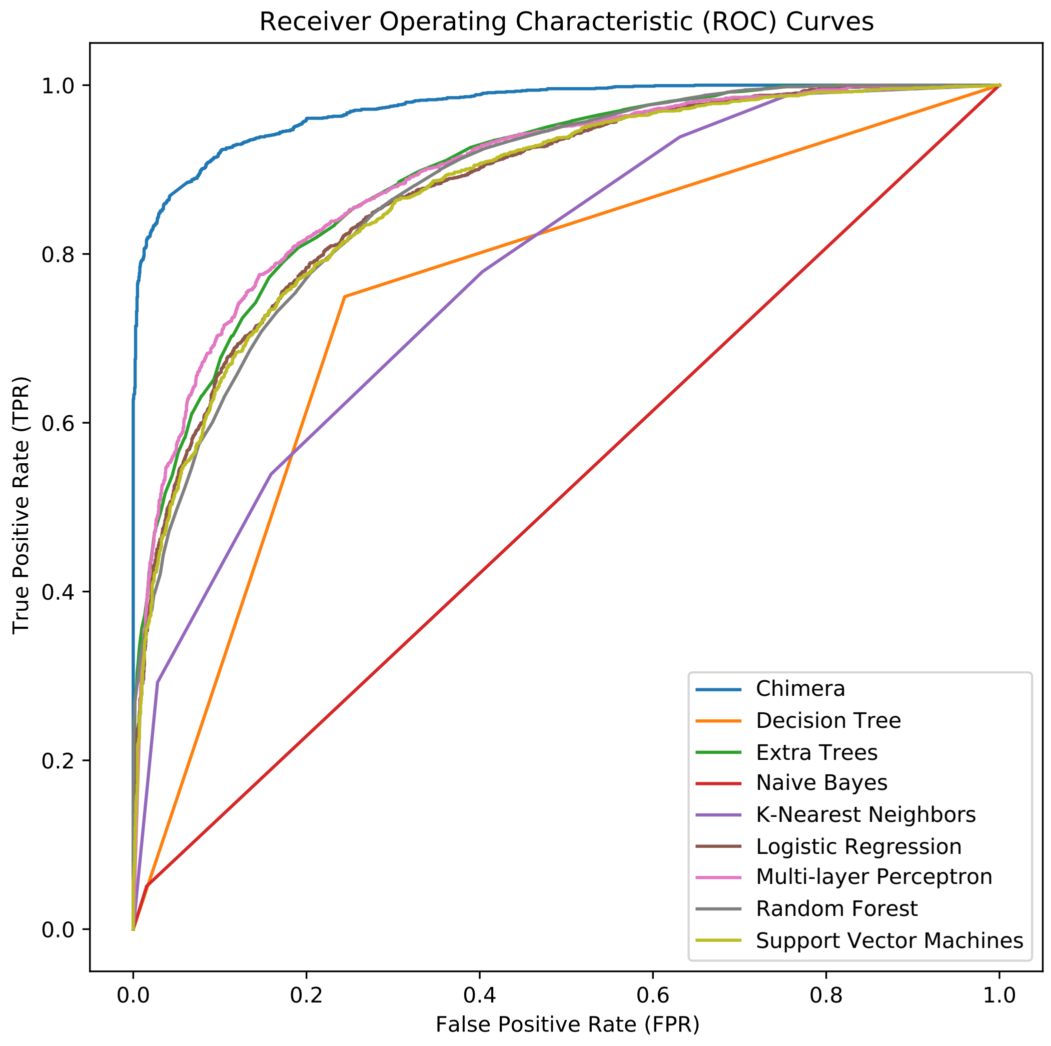


Figure 4.1: *Receiver Operating Characteristic (ROC) curves of Chimera, Machine Learning, and Ensemble Machine Learning methods.*

CHAPTER 5

CONCLUSION

The general objective of this dissertation was to develop and evaluate a new Android malware detection method, named Chimera, based on Multimodal Deep Learning (MDL) and Hybrid Analysis (HA), using different data modalities and combining both manual and automatic feature engineering in order to increase Android malware detection rate, thus answering the proposed research question: How the development and evaluation of a new Android malware detection method, based on MDL and HA, using different data modalities and combining both manual and automatic feature engineering, can increase Android malware detection rate? With the aim of answering the research question, a set of specific objectives were implemented: 1) To build a multimodal dataset containing data extracted from multiple data sources. 2) To implement the KDD process for feature selection, data preprocessing, and data transformation. 3) To tune the models by choosing the best set o hyperparameters using model selection strategies. 4) To determine the best training strategy for the MDL method Chimera, and 5) To evaluate and compare the method with classical ML methods, Ensemble ML methods, Chimera's subnetworks Chimera-R, Chimera-S, and Chimera-D.

Chimera combines 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 results of the computational experiments showed that Chimera's Accuracy, Precision, Recall, and AUC ROC reached 0.909 ± (0.001), 0.948 ± (0.003), 0.863 ± (0.004), and 0.972 ± (0.000) respectively, outperforming the classical ML methods, Ensemble ML methods, Chimera's DL subnetworks Chimera-S (CHS), Chimera-R (CHR), and Chimera-D (CHD), thus answering the research question posed by this dissertation.

In summary, this dissertation has the following contributions:

- Contributions to the academia:
  - The development of a new MDL method using specialized DL architectures for feature learning from different security-related data modalities.
  - The performance evaluation results showing that the developed method's performance outperforms several classical ML, Ensemble ML, and DL methods, which by themselves justify the method development.
  - The creation of a new multimodal dataset made publicly available at the IEEE DataPort Dataset Storage and Dataset Search Platform (OLIVEIRA; SASSI, 2021d).
- Contributions to the corporations: This dissertation is a research work in which a new method was proposed, developed, and evaluated. In order to the developed method become a production-ready software that can be used by corporations to protect their perimeters and users, it is necessary to invest in the product development. Therefore, the main contribution of this dissertation to the corporations are evidences that the proposed method is scientifically sound and might be considered for product development.
- Contributions to the end users and to the society: As aforementioned, once the research work becomes a production-ready software, not only corporations could use it to protect their perimeters but also end users could take advantage of its high accuracy. For example, the product could be offered in cloud platform used to validate Android applications before they are installed on the user's devices, thus protecting the end users against Android malware.

The method developed in this dissertation has the following limitations that could be tackled in a future work:

- All the models developed in this dissertation followed minimal architectures. More complex architectures could have been proposed, resulting in better performance, but it would require much more computational resources that were not available. This situation had a direct impact on the range of hyperparameters used during model selection and tuning.
- Chimera requires that the data of all the data modalities are available for each instance. So, given an instance, if there is a problem gathering dynamic analysis data but static data was gathered correctly, the model would ignore that instance as a whole. The solution

to this problem would make the method more resilient and could possibly be used as a multimodal regularization method to reduce overfitting.

- Chimera is a black-box multimodal deep learning method. The development and inclusion of interpretable DL layers to Chimera would provide information on why a sample was classified as malware, which would be useful for incident responders and defenders.
- Chimera was trained and evaluated using the Omnidroid dataset, which is a balanced dataset. It is important to understand the methods' performance on unbalanced data taking into consideration each data modality and the multimodal data modality.

Android malware poses a significant threat to the modern society and its complexity and variety keep rising as organizations try to defend their perimeters using new tools and techniques. It is a cat-and-mouse game in which defenders and malicious agents are constantly improving their detection and anti-detection tactics and techniques, one trying to overcome the other.

The research carried out in this dissertation do not intend to exhaust the subject, on the contrary, it was sought to make a contribution to the development of Android malware detection methods based on multimodal deep learning and hybrid analysis. It is known that there is a clear demand for systematic studies that can establish other application domains even more suitable for the proposed method. This scenario therefore offers ample space for future work.

CHAPTER 6

APPENDIX I : PUBLICATIONS

(OLIVEIRA; SASSI, 2021c) . **Hunting Android Malware Using Multimodal Deep Learning and Hybrid Analysis Data**. XV Brazilian Congress on Computational Intelligence (CBIC 2021), Joinville - SC.

(OLIVEIRA; SASSI, 2021a) . **Behavioral Malware Detection using Deep Graph Convolutional Neural Networks**. International Journal of Computer Applications. April 2021.

(OLIVEIRA; SASSI, 2020) . **Chimera: An Android Malware Detection Method Based on Multimodal Deep Learning and Hybrid Analysis**. TechRxiv. December 2020.

CHAPTER 7

APPENDIX II : PUBLISHED DATASETS

(OLIVEIRA; SASSI, 2021d). **Malware Analysis Datasets: Chimera Multimodal Deep Learning Android Malware Detection Method** - IEEE Dataport. 2021.

(OLIVEIRA; SASSI, 2019a). **Malware Analysis Datasets: API Call Sequences** - IEEE Dataport. 2019.

(OLIVEIRA; SASSI, 2019d). **Malware Analysis Datasets: Top-1000 PE Imports** - IEEE Dataport. 2019.

(OLIVEIRA; SASSI, 2019c). **Malware Analysis Datasets: Raw PE as Image** - IEEE Dataport. 2019.

(OLIVEIRA; SASSI, 2019b). **Malware Analysis Datasets: PE Section Headers** - IEEE Dataport. 2019.

CHAPTER 8

APPENDIX III : MACHINE LEARNING AND ENSEMBLE MACHINE LEARNING METHODS' SCIKIT-LEARN HYPERPARAMETERS

ExtraTreesClassifier(random_state=42)

{'bootstrap': False, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': 'auto', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': 42, 'verbose': 0, 'warm_start': False}

RandomForestClassifier(random_state=42)

{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': 'auto', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': 42, 'verbose': 0, 'warm_start': False}

LogisticRegression(random_state=42)

{'C': 1.0, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercept_scaling': 1, 'l1_ratio': None, 'max_iter': 100, 'multi_class': 'auto', 'n_jobs': None, 'penalty': 'l2', 'random_state': 42, 'solver': 'lbfgs', 'tol': 0.0001, 'verbose': 0, 'warm_start': False}

GaussianNB()

{'priors': None, 'var_smoothing': 1e-09}

KNeighborsClassifier()

{'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski', 'metric_params': None, 'n_jobs': None, 'n_neighbors': 5, 'p': 2, 'weights': 'uniform'}

LinearSVC(random_state=42)

{'C': 1.0, 'class_weight': None, 'dual': True, 'fit_intercept': True, 'intercept_scaling': 1, 'loss': 'squared_hinge', 'max_iter': 1000, 'multi_class': 'ovr', 'penalty': 'l2', 'random_state': 42, 'tol': 0.0001, 'verbose': 0}

DecisionTreeClassifier(random_state=42)

{'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'random_state': 42, 'splitter': 'best'}

MLPClassifier(random_state=42)

{'activation': 'relu', 'alpha': 0.0001, 'batch_size': 'auto', 'beta_1': 0.9, 'beta_2': 0.999, 'early_stopping': False, 'epsilon': 1e-08, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'learning_rate_init': 0.001, 'max_fun': 15000, 'max_iter': 200, 'momentum': 0.9, 'n_iter_no_change': 10, 'nesterovs_momentum': True, 'power_t': 0.5, 'random_state': 42, 'shuffle': True, 'solver': 'adam', 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose': False, 'warm_start': False}

Bibliography

ALPAYDIN, E. *Introduction to machine learning*. [S.l.]: MIT press, 2020. Citado na pág. 20, 46, 51, 54.

ALSMADI, T.; ALQUDAH, N. A survey on malware detection techniques. In: *2021 International Conference on Information Technology (ICIT)*. [S.l.: s.n.], 2021. p. 371–376. Citado na pág. 42.

AMRUTHA, N.; BALAGOPAL, N. Multimodal deep learning method for detection of malware in android using static and dynamic features. *CSI Journal of*, p. 13, 2020. Citado na pág. 16, 45.

ANDROZOO. *Androzoo Ransomware*. 2021. Disponível em: <https://androzoo.uni.lu/>. Citado na pág. 19, 20, 49.

ANI, U. P. D.; HE, H.; TIWARI, A. Review of cybersecurity issues in industrial critical infrastructure: manufacturing in perspective. *Journal of Cyber Security Technology*, Taylor & Francis, v. 1, n. 1, p. 32–74, 2017. Citado na pág. 15.

ARLOT, S.; CELISSE, A. et al. A survey of cross-validation procedures for model selection. *Statistics surveys*, The author, under a Creative Commons Attribution License, v. 4, p. 40–79, 2010. Citado na pág. 60.

CANALTECH. *Renner Ransomware*. 2021. Disponível em: <https://canaltech.com.br/seguranca/lojas-renner-esta-fora-do-ar-e-confirma-ataque-de-sequestro-digital-193292/>. Citado na pág. 17.

COMMONS, C. *Creative Commons Attribution 4.0 International (CC BY 4.0)*. 2021. Disponível em: <https://creativecommons.org/licenses/by/4.0/>. Citado na pág. 50.

DAWSON, J.; THOMSON, R. The future cybersecurity workforce: going beyond technical skills for successful cyber performance. *Frontiers in psychology*, Frontiers, v. 9, p. 744, 2018. Citado na pág. 22, 23.

DHALARIA, M.; GANDOTRA, E. A hybrid approach for android malware detection and family classification. *International Journal of Interactive Multimedia & Artificial Intelligence*, v. 6, n. 6, 2021. Citado na pág. 45.

DING, Y.; ZHANG, X.; HU, J.; XU, W. Android malware detection method based on bytecode image. *Journal of Ambient Intelligence and Humanized Computing*, Springer, p. 1–10, 2020. Citado na pág. 53, 64.

EGELE, M.; SCHOLTE, T.; KIRDA, E.; KRUEGEL, C. A survey on automated dynamic malware-analysis techniques and tools. *ACM computing surveys (CSUR)*, ACM New York, NY, USA, v. 44, n. 2, p. 1–42, 2008. Citado na pág. 15.

EITAN, A. T.; SMOLYANSKY, E.; HARPAZ, I. K.; PERETS, S. 2021. Disponível em: <https://www.connectedpapers.com/>. Citado na pág. 43, 44.

ENCK, W.; OCTEAU, D.; MCDANIEL, P. D.; CHAUDHURI, S. A study of android application security. In: *USENIX security symposium*. [S.l.: s.n.], 2011. v. 2, p. 2. Citado na pág. 53.

FAYYAD, U.; PIATETSKY-SHAPIRO, G.; SMYTH, P. From data mining to knowledge discovery in databases. *AI magazine*, v. 17, n. 3, p. 37–37, 1996. Citado na pág. 18, 25, 27, 46.

FEIZOLLAH, A.; ANUAR, N. B.; SALLEH, R.; SUAREZ-TANGIL, G.; FURNELL, S. Androdialysis: Analysis of android intent effectiveness in malware detection. *computers & security*, Elsevier, v. 65, p. 121–134, 2017. Citado na pág. 51.

GERHARDT, T. E.; SILVEIRA, D. T. *Métodos de pesquisa*. [S.l.]: Plageder, 2009. Citado na pág. 46.

GIBERT, D.; MATEU, C.; PLANES, J. Hydra: A multimodal deep learning framework for malware classification. *Computers & Security*, Elsevier, p. 101873, 2020. Citado na pág. 69.

GIL, A. C. *Métodos e técnicas de pesquisa social*. [S.l.]: 6. ed. Ediitora Atlas SA, 2008. Citado na pág. 46.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>. Citado na pág. 27, 28, 29, 30, 31, 32, 33, 36, 56, 64, 65.

GOOGLE. 2021. <https://scholar.google.com.br/scholar?hl=en&as_sdt=0%2C5&q=Android+Malware+Detection+Multimodal+Deep+Learning&btnG=>. Accessed: 2021-10-22. Citado na pág. 43.

HARRIS, C. R.; MILLMAN, K. J.; WALT, S. J. van der; GOMMERS, R.; VIRTANEN, P.; COURNAPEAU, D.; WIESER, E.; TAYLOR, J.; BERG, S.; SMITH, N. J.; KERN, R.; PICUS, M.; HOYER, S.; KERKWIJK, M. H. van; BRETT, M.; HALDANE, A.; R'ıO, J. F. del; WIEBE, M.; PETERSON, P.; G'eRARD-MARCHANT, P.; SHEPPARD, K.; REDDY, T.; WECKESSER, W.; ABBASI, H.; GOHLKE, C.; OLIPHANT, T. E. Array programming with NumPy. *Nature*, Springer Science and Business Media LLC, v. 585, n. 7825, p. 357–362, set. 2020. Disponível em: <https://doi.org/10.1038/s41586-020-2649-2>. Citado na pág. 50.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 770–778. Citado na pág. 64.

HUANG, T. H.-D.; KAO, H.-Y. R2-d2: color-inspired convolutional neural network (cnn)-based android malware detections. In: IEEE. *2018 IEEE International Conference on Big Data (Big Data)*. [S.l.], 2018. p. 2633–2642. Citado na pág. 53, 64.

HUMAYUN, M.; JHANJHI, N.; ALSAYAT, A.; PONNUSAMY, V. Internet of things and ransomware: Evolution, mitigation and prevention. *Egyptian Informatics Journal*, Elsevier, v. 22, n. 1, p. 105–117, 2021. Citado na pág. 17.

HWANG, S.-J.; CHUNG, H. An android malware detector using deep learning hybrid model. In: *The CICET 2020 Technical Program includes 2 invited speakers and 19 oral presentations. We are beholden to all of the authors and speakers for their contributions to CICET 2020. On behalf of the program committee, we would like to welcome the delegates and their guests to CICET 2020. We hope that the delegates and guests will enjoy the conference.* [S.l.: s.n.], 2020. p. 3. Citado na pág. 45.

IDIKA, N.; MATHUR, A. P. A survey of malware detection techniques. *Purdue University*, v. 48, p. 2007–2, 2007. Citado na pág. 15, 16.

IDREES, F.; RAJARAJAN, M. Investigating the android intents and permissions for malware detection. In: IEEE. *2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. [S.l.], 2014. p. 354–358. Citado na pág. 51.

IDREES, F.; RAJARAJAN, M.; CHEN, T. M.; RAHULAMATHAVAN, Y.; NAUREEN, A. Andropin: Correlating android permissions and intents for malware detection. In: IEEE. *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. [S.l.], 2017. p. 394–399. Citado na pág. 51.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. Citado na pág. 37.

JIMÉNEZ, J. M. H.; GOSEVA-POPSTOJANOVA, K. Using four modalities for malware detection based on feature level and decision level fusion. In: SPRINGER. *International Conference on Advanced Information Networking and Applications*. [S.l.], 2020. p. 1383–1396. Citado na pág. 19.

KESSLER, M. M. Bibliographic coupling between scientific papers. *American documentation*, Wiley Online Library, v. 14, n. 1, p. 10–25, 1963. Citado na pág. 43.

KHARIWAL, K.; SINGH, J.; ARORA, A. Ipdroid: Android malware detection using intents and permissions. In: *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*. [S.l.: s.n.], 2020. p. 197–202. Citado na pág. 24.

KIM, T.; KANG, B.; RHO, M.; SEZER, S.; IM, E. G. A multimodal deep learning method for android malware detection using various features. *IEEE Transactions on Information Forensics and Security*, v. 14, n. 3, p. 773–788, 2019. Citado na pág. 13, 16, 19, 42, 43, 44, 45, 69.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. Citado na pág. 35, 36.

KOODOUS. *Koodous*. 2021. Disponível em: <https://koodous.com/>. Citado na pág. 19, 20, 49.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *nature*, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015. Citado na pág. 16.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, Ieee, v. 86, n. 11, p. 2278–2324, 1998. Citado na pág. 38, 64.

LI, X.; LOH, P. K.; TAN, F. Mechanisms of polymorphic and metamorphic viruses. In: IEEE. *2011 European intelligence and security informatics conference*. [S.l.], 2011. p. 149–154. Citado na pág. 16.

LIN, M.; CHEN, Q.; YAN, S. Network in network. *arXiv preprint arXiv:1312.4400*, 2013. Citado na pág. 64.

LIU, K.; XU, S.; XU, G.; ZHANG, M.; SUN, D.; LIU, H. A review of android malware detection approaches based on machine learning. *IEEE Access*, v. 8, p. 124579–124607, 2020. Citado na pág. 42.

LONG, J.; SHELHAMER, E.; DARRELL, T. Fully convolutional networks for semantic segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2015. p. 3431–3440. Citado na pág. 70.

MARTÍN, A.; LARA-CABRERA, R.; CAMACHO, D. Android malware detection through hybrid features fusion and ensemble classifiers: the andropytool framework and the omnidroid dataset. *Information Fusion*, Elsevier, v. 52, p. 128–142, 2019. Citado na pág. 19, 49, 59.

MAYRHOFER, R.; STOEP, J. V.; BRUBAKER, C.; KRALEVICH, N. The android platform security model. *ACM Transactions on Privacy and Security (TOPS)*, ACM New York, NY, USA, v. 24, n. 3, p. 1–35, 2021. Citado na pág. 23.

MCGIFF, J.; HATCHER, W. G.; NGUYEN, J.; YU, W.; BLASCH, E.; LU, C. Towards multimodal learning for android malware detection. In: IEEE. *2019 International Conference on Computing, Networking and Communications (ICNC)*. [S.l.], 2019. p. 432–436. Citado na pág. 16, 45.

MICROSOFT. 2021. <https://academic.microsoft.com/search?q=Android%20Malware%20Detection%20Multimodal%20Deep%20Learning&f=&orderBy=0&skip=0&take=10>. Accessed: 2021-10-22. Citado na pág. 43.

MILLAR, S.; MCLAUGHLIN, N.; RINCON, J. M. del; MILLER, P. Multi-view deep learning for zero-day android malware detection. *Journal of Information Security and Applications*, Elsevier, v. 58, p. 102718, 2021. Citado na pág. 45.

NARUDIN, F. A.; FEIZOLLAH, A.; ANUAR, N. B.; GANI, A. Evaluation of machine learning classifiers for mobile malware detection. *Soft Computing*, Springer, v. 20, n. 1, p. 343–357, 2016. Citado na pág. 16.

NGIAM, J.; KHOSLA, A.; KIM, M.; NAM, J.; LEE, H.; NG, A. Y. Multimodal deep learning. In: *ICML*. [S.l.: s.n.], 2011. Citado na pág. 16, 18, 42.

NGUYEN, D. V.; NGUYEN, G. L.; NGUYEN, T. T.; NGO, A. H.; PHAM, G. T. Minad: Multi-inputs neural network based on application structure for android malware detection. *Peer-to-Peer Networking and Applications*, Springer, p. 1–15, 2021. Citado na pág. 45.

NIU, M.; LI, Y.; WANG, C.; HAN, K. Rfamyloid: a web server for predicting amyloid proteins. *International journal of molecular sciences*, Multidisciplinary Digital Publishing Institute, v. 19, n. 7, p. 2071, 2018. Citado na pág. 61.

OLIVEIRA, A.; SASSI, R. *Malware Analysis Datasets: API Call Sequences*. IEEE Dataport, 2019. Disponível em: <https://dx.doi.org/10.21227/tqqm-aq14>. Citado na pág. 88.

OLIVEIRA, A.; SASSI, R. *Malware Analysis Datasets: PE Section Headers*. IEEE Dataport, 2019. Disponível em: <https://dx.doi.org/10.21227/2czh-es14>. Citado na pág. 88.

OLIVEIRA, A.; SASSI, R. *Malware Analysis Datasets: Raw PE as Image*. IEEE Dataport, 2019. Disponível em: <https://dx.doi.org/10.21227/8brp-j220>. Citado na pág. 88.

OLIVEIRA, A.; SASSI, R. *Malware Analysis Datasets: Top-1000 PE Imports*. IEEE Dataport, 2019. Disponível em: <https://dx.doi.org/10.21227/004e-v304>. Citado na pág. 88.

OLIVEIRA, A.; SASSI, R. *Chimera: An Android Malware Detection Method Based on Multimodal Deep Learning and Hybrid Analysis*. TechRxiv, 2020. Disponível em: <https://www.techrxiv.org/articles/preprint/Chimera_An_Android_Malware_Detection_Method_Based_on_Multimodal_Deep_Learning_and_Hybrid_Analysis/13359767/1>. Citado na pág. 87.

OLIVEIRA, A.; SASSI, R. Behavioral malware detection using deep graph convolutional neural networks. *International Journal of Computer Applications*, Foundation of Computer Science (FCS), NY, USA, New York, USA, v. 174, n. 29, p. 1–8, Apr 2021. ISSN 0975-8887. Disponível em: <http://www.ijcaonline.org/archives/volume174/number29/31858-2021921218>. Citado na pág. 87.

OLIVEIRA, A.; SASSI, R. *Chimera Code*. 2021. Disponível em: <https://github.com/angelo5d0>. Citado na pág. 50.

OLIVEIRA, A.; SASSI, R. Hunting android malware using multimodal deep learning and hybrid analysis data. In: FILHO, C. J. A. B.; SIQUEIRA, H. V.; FERREIRA, D. D.; BERTOL, D. W.; OLIVEIRA, R. C. L. ao de (Ed.). *Anais do 15 Congresso Brasileiro de Inteligência Computacional*. Joinville, SC: SBIC, 2021. p. 1–10. Disponível em: <https://sbic.org.br/eventos/cbic_2021/cbic2021-32/>. Citado na pág. 45, 87.

OLIVEIRA, A.; SASSI, R. *Malware Analysis Datasets: Chimera Multimodal Deep Learning Android Malware Detection Method*. IEEE Dataport, 2021. Disponível em: <https://dx.doi.org/10.21227/1wdz-2d93>. Citado na pág. 50, 85, 88.

PASZKE, A.; GROSS, S.; MASSA, F.; LERER, A.; BRADBURY, J.; CHANAN, G.; KILLEEN, T.; LIN, Z.; GIMELSHEIN, N.; ANTIGA, L.; DESMAISON, A.; KOPF, A.; YANG, E.; DEVITO, Z.; RAISON, M.; TEJANI, A.; CHILAMKURTHY, S.; STEINER, B.; FANG, L.; BAI, J.; CHINTALA, S. Pytorch: An imperative style, high-performance deep learning library. In: WALLACH, H.; LAROCHELLE, H.; BEYGELZIMER, A.; ALCHé-BUC, F. d'; FOX, E.; GARNETT, R. (Ed.). *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019. p. 8024–8035. Disponível em: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>. Citado na pág. 50.

PATEL, K. towardsdatascience.com, 2019. Disponível em: <https://towardsdatascience.com/overfitting-vs-underfitting-ddc80c2fc00d>. Citado na pág. 37.

QIU, J.; ZHANG, J.; LUO, W.; PAN, L.; NEPAL, S.; XIANG, Y. A survey of android malware detection with deep neural models. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 53, n. 6, dez. 2020. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3417978>. Citado na pág. 42.

Rachmawati, D.; Tarigan, J. T.; Ginting, A. B. C. A comparative study of message digest 5(md5) and sha256 algorithm. *Journal of Physics: Conference Series*, v. 978, n. 1, p. 12116, 2018. Citado na pág. 49.

ROŠKOT, M.; WANASIKA, I.; KROUPOVA, Z. K. Cybercrime in europe: surprising results of an expensive lapse. *Journal of Business Strategy*, Emerald Publishing Limited, 2020. Citado na pág. 17.

SIKORSKI, M.; HONIG, A. *Practical malware analysis: the hands-on guide to dissecting malicious software*. [S.l.]: no starch press, 2012. Citado na pág. 15, 24, 25.

SMALL, H. Co-citation in the scientific literature: A new measure of the relationship between two documents. *Journal of the American Society for information Science*, Wiley Online Library, v. 24, n. 4, p. 265–269, 1973. Citado na pág. 43.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, JMLR. org, v. 15, n. 1, p. 1929–1958, 2014. Citado na pág. 37, 38, 39.

STRANG, G.; HERMAN, E. J. *Calculus Volume 3*. [S.l.]: OpenStax, 2016. Citado na pág. 33, 34.

SUAREZ-TANGIL, G.; STRINGHINI, G. Eight years of rider measurement in the android malware ecosystem. *IEEE Transactions on Dependable and Secure Computing*, IEEE, 2020. Citado na pág. 17.

TEAM, T. pandas development. *pandas-dev/pandas: Pandas*. [S.l.], 2020. Disponível em: <https://doi.org/10.5281/zenodo.3509134>. Citado na pág. 50.

TENSORFLOW. 2021. <https://playground.tensorflow.org/>. Accessed: 2021-10-22. Citado na pág. 31.

THARWAT, A. Classification assessment methods. *Applied Computing and Informatics*, Emerald Publishing Limited, 2020. Citado na pág. 59, 60.

TIETZ, M.; FAN, T. J.; NOURI, D.; BOSSAN, B.; skorch Developers. *skorch: A scikit-learn compatible neural network library that wraps PyTorch*. [S.l.], 2017. Disponível em: <https://skorch.readthedocs.io/en/stable/>. Citado na pág. 50.

TURKOWSKI, K. Filters for common resampling tasks. In: ACADEMIC PRESS PROFESSIONAL, INC. *Graphics gems*. [S.l.], 1990. p. 147–165. Citado na pág. 54, 55.

VASU, B.; PARI, N. Combining multimodal dnn and sigpid technique for detecting malicious android apps. In: IEEE. *2019 11th International Conference on Advanced Computing (ICoAC)*. [S.l.], 2019. p. 289–294. Citado na pág. 16, 19, 45.

VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, Ł.; POLOSUKHIN, I. Attention is all you need. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2017. p. 5998–6008. Citado na pág. 40, 41, 67, 68.

VENTURES, C. *Cybercrime Cost*. 2021. Disponível em: <https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/>. Citado na pág. 17.

WANG, Z.; LIU, Q.; CHI, Y. Review of android malware detection based on deep learning. *IEEE Access*, IEEE, 2020. Citado na pág. 16.

XIAO, X.; ZHANG, S.; MERCALDO, F.; HU, G.; SANGAIAH, A. K. Android malware detection based on system call sequences and lstm. *Multimedia Tools and Applications*, Springer, v. 78, n. 4, p. 3979–3999, 2019. Citado na pág. 55, 67.

ZHOU, Z.-H. *Ensemble methods: foundations and algorithms*. [S.l.]: CRC press, 2012. Citado na pág. 20.

ZHU, D.; XI, T.; JING, P.; WU, D.; XIA, Q.; ZHANG, Y. 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*. [S.l.: s.n.], 2019. p. 51–60. Citado na pág. 16, 19, 45.