A thesis submitted in partial fulfillment of the requirements
for the industrial engineering degree, option:
Data Science & Artificial Intelligence

Automation in Cybersecurity: Deep Learning-Based Approaches for Malware Family Identification

Presented by:
Chaimaa ABI

Supervised by:
Dr. BERRANI Sid-Ahmed (ENP)
Dr. BOUDJELLAL Abdelouahab (EMP)

Publicly defended on June 22, 2023

Board of Examiners

ENP 2023

République Algérienne Démocratique Et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École Nationale Polytechnique

Département du Génie Industriel

A thesis submitted in partial fulfillment of the requirements
for the industrial engineering degree, option:
Data Science & Artificial Intelligence

Automation in Cybersecurity: Deep Learning-Based Approaches for Malware Family Identification

Presented by:
Chaimaa ABI

Supervised by:
Dr. BERRANI Sid-Ahmed (ENP)
Dr. BOUDJELLAL Abdelouahab (EMP)

Publicly defended on June 22, 2023

ENP 2023

République Algérienne Démocratique Et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École Nationale Polytechnique

Département du Génie Industriel

Mémoire de projet de fin d'étude en vue de l'obtention du diplôme d'ingénieur d'État en génie industriel, option : Data Science & Intelligence artificielle. Intelligence

---

Automatisation en Cybersécurité : Approches basées sur l'Apprentissage Profond pour l'Identification des Familles de Logiciels Malveillants.

Presenté par:
Chaimaa ABI

Encadré par:
Dr. BERRANI Sid-Ahmed (ENP)
Dr. BOUDJELLAL Abdelouahab (EMP)

Présenté et soutenu publiquement le 22 Juin, 2023

Composition du Jury

Président	ZOUAGHI Iskander	MCA	(ENP)
Examinateur	FOURAR-LAIDI Hakim	MCA	(ENP)
Promoteur	BERRANI Sid-Ahmed	MCA	(ENP)
Promoteur	BOUDJELLAL Abdelouahab	MCA	(EMP)

ENP 2023

Acknowledgements

I would like to express my deepest gratitude to my supervisors, Mr. Sid-Ahmed BERRANI and Mr. Abdelouahab BOUDJELLAL, for their invaluable guidance, support, and encouragement throughout the research process. Their expertise and insightful feedback have been instrumental in shaping this thesis.

My sincere thanks go to my family: my mother, my father and my beloved siblings for their unwavering support and understanding throughout this academic journey. Their love, encouragement, and belief in my abilities have been my constant source of motivation.

ملخص

يواجه العالم الرقمي تزايداً في انتشار البرامج الضارة، مما يشكل تهديداً كبيراً على أنظمة الكمبيوتر وأمن البيانات. تعتبر القدرة على اكتشاف وتصنيف البرامج الضارة بدقة أمراً حاسماًا لمكافحة التهديدات السيبرانية ومنع الأضرار المحتملة. إلا أن الأساليب التقليدية المستخدمة حالياًا تواجه تحديات في مواكبة التطور المستمر في مجال البرمجيات الضارة. لذا، يوجد حاجة ملحة لاستخدام النظم الذكية في هذا المجال. تهدف هذه الأطروحة إلى تطوير نظام تصنيف قوي ودقيق للبرامج الضارة باستخدام تقنيات التعلم الآلي والتعلم العميق، وتسهم في تعزيز الأمن السيبراني وحماية الأنظمة والبيانات من التهديدات المستمرة.
كلمات مفتاحية:تحليل البرامج الضارة؛ تصنيف البرامج الضارة؛ تصور البرامج الضارة؛ استخراج الميزات؛ التعلم العميق؛ متعدد الوسائط؛ الشبكات العصبية الالتفافية؛ التعلم الآلي

Résumé

La prolifération rapide des logiciels malveillants présente une menace importante pour les systèmes informatiques et la sécurité des données. La capacité à détecter et classer avec précision les logiciels malveillants est essentielle pour atténuer les menaces cybernétiques. Cependant, les méthodes traditionnelles de classification et d'analyse des logiciels malveillants ont montré certaines limites face à l'évolution constante des malwares. Dans cette thèse, nous proposons une approche qui exploite la puissance des techniques d'apprentissage automatique et d'apprentissage profond pour une classification efficace des logiciels malveillants et contribue à renforcer la cybersécurité et à protéger les systèmes et les données contre les menaces constantes.
Mots-clés : Analyse des logiciels malveillants ; classification des logiciels malveillants ; visualisation des logiciels malveillants ; extraction de caractéristiques ; apprentissage profond ; multimodal ; réseaux neuronaux convolutionnels ; apprentissage automatique.

Abstract

The rapid proliferation of malware presents a significant threat to computer systems and data security. The ability to detect and accurately classify malware is crucial for mitigating cyber threats and preventing potential damages. However, traditional methods for malware classification and analysis have shown some limitations in keeping pace with the with the ever-changing landscape of malware. In this thesis, we propose a novel approach that harnesses the power of machine and deep learning techniques for efficient malware classification and offers real-time and automated data-driven solution, enabling proactive measures to efficiently prevent and mitigate cyber threats.
Keywords: malware analysis; malware classification; malware visualization; feature extraction; deep learning; multimodal; convolutional neural networks; machine learning

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
ASM	Assembly Language
BiGRU	Bidirectional Gated Recurrent Unit
CNN	Convolutional Neural Networks
DDL	Dynamic Link Library
DL	Deep Learning
DPI	Deep Packet Inspection
FNN	Feed Forward Network
GRU	Gated Recurrent Unit
k-NN	k-Nearest Neighbors
LSTM	Long Short-Term Memory
Malware	Malicious Software
Opcodes	Operation Codes
RF	Random Forest
RNN	Recurrent Neural Network
RELU	Rectified Linear Unit
SVM	Support Vector Machine
TCN	Temporal Convolutional Network

Chapter 1

General Introduction

1.1 Problem Statement

Malware, short for malicious software, poses a serious threat to users, organizations, and computer systems. It is specifically designed to exploit system vulnerabilities and cause harm, ranging from stealing personal information to disrupting critical operations. Recent data from the AV-Test[1] Institute reveals a staggering increase in the number of malware and potentially unwanted applications. A total of 1,270 million instances have been recorded, with an alarming 39 million new cases identified in the first half of 2023 alone [1]. These figures demonstrate the urgent need for robust methodologies in discerning and classifying the ever-growing range of sophisticated and diverse malicious programs.

The current approaches for identifying malware are primarily signature-based systems and rule-based systems. Signature-based systems compare incoming traffic with a database of known threats to detect and categorize malware. However, they face limitations when encountering unknown threats not present in the database. Rule-based systems, on the other hand, rely on predefined rules or policies to define acceptable network behavior. When traffic violates these rules, an alert is triggered. However, assigning specific rules to every type of malware proves challenging for rule-based systems. For example, deep packet inspection (DPI), a commonly used rule-based system, analyzes data packets based on predefined rules but struggles to effectively detect emerging or unknown threats. Consequently, the constantly evolving cyber threat landscape surpasses the capabilities of signature and rule-based systems like DPI.

Moreover, analyzing and understanding malware presents additional challenges. Attackers continually employ anti-analysis techniques [2] such as obfuscation[2], polymorphism[3], metamorphism[4], and packing to evade detection [3, 4, 5]. These techniques obscure the true nature of malware, making it difficult to analyze and detect malicious intent. Moreover, the manual analysis process is time-consuming and struggles to keep up with the

1. An independent organization that conducts antivirus software testing and research. More information can be found at www.av-test.org.
2. Practice of making code obscure and harder to understand using techniques like encryption and control flow manipulation.
3. Generation of different variations of code to evade signature-based detection.
4. Active modification of code during execution to change its appearance and behavior.

rapidly growing number of malware instances. In an experimental study [6], a malware classification game has been conducted, involving 110 participants with different levels of expertise. The results show that experts took an average of 29 minutes, while novices required around 44.5 minutes in average to classify unknown samples based on detailed sandbox reports.

In addition to these challenges, attackers are now leveraging AI algorithms to automate various malicious activities, contributing to the mass spread of malware. By automating the generation of convincing phishing emails, messages, or websites, AI amplifies the potential impact of malware. Exploiting AI models like ChatGPT allows attackers to automate the creation of obfuscated and polymorphic code, enabling them to generate a large quantity of malware variants with a drastically reduced effort.

To combat AI-driven threats, a response beyond human expertise and manual defense techniques is necessary. Traditional defense methods are no longer sufficient in the face of sophisticated attacks powered by AI. Intelligent systems, powered by machine and deep learning algorithms, provide valuable solutions to address these challenges. By leveraging large datasets of malware, these data-driven systems can learn patterns associated with anti-analysis techniques, hence, they can identify and classify malware even when it is obfuscated or disguised. Furthermore, Intelligent systems automate the analysis process, enabling real-time processing of a large volume of malware samples based on learned patterns.

The remainder of the introductory chapter is organized as follows: In Section 1.2, we discuss relevant published work in the area of malware classification, exploring various approaches and techniques employed by researchers. We then highlight the gaps and limitations of the existing approaches in Section 1.3.
Section 1.4 outlines the objectives of this study, clearly defining the goals and desired outcomes. Finally, in Section 1.5, we describe the organization of the remaining chapters in this thesis, providing a clear roadmap to navigate through the document.

1.2 Related Published Work

Researchers have explored various approaches within the literature, including traditional machine learning and deep learning techniques applied to malware analysis.

Traditional machine learning approaches have been widely utilized to distinguish malware programs. Santos et al. [7] proposed a method that utilized opcode sequences as features for malware detection. By calculating the frequency of opcode sequences and applying mutual information, the authors identified the most relevant opcodes for accurate malware classification. Masud et al. [8] trained a classifier using in addition to n-grams: assembly instruction sequences and Dynamic Link Library (DLL) function calls as features extracted from the executables.

API call[5] sequences have also been leveraged as features in malware analysis. Sato

5. Refer to the specific function calls made by a malware program to application programming interfaces.

and Tran [9] employed techniques from Natural Language Processing, such as n-grams, doc2vec[6], and TF-IDF[7], to convert API sequences into numeric vectors. These vectors are then used as inputs to malware classifiers. Similarly, the authors of [10] have utilized deep learning techniques, specifically Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) models, to analyze and classify malware programs based on API call sequences.

In the paper [11], word embeddings, such as HMM2Vec[8] and Word2Vec[9], have been used to represent opcode sequences extracted from assembly files. These embeddings captured semantic relationships between opcodes and served as features for classification algorithms, including support vector machine (SVM), k-nearest neighbor (k-NN), and random forest (RF) in malware analysis.

In another study done by Xie et al. [12], a similar approach has been employed, utilizing word embeddings to represent opcode sequences. However, instead of traditional classifiers, LSTM layers were used for classification. Additionally, the authors have incorporated a self-attention mechanism to enhance the model's ability to capture relevant features and dependencies within the malware data.

Several studies have explored the utilization of image-based techniques for malware classification. Nataraj et al. [13] proposed converting byte files into grayscale images and observed visual similarities among malware samples from the same family. Vasan et al. [14] extended this concept by converting raw malware binaries into color images and employing convolutional neural network (CNN) techniques for classification. Furthermore, Narayanan and Davuluru [15] introduced an ensemble[10] classification system combining CNNs and LSTM networks to distinguish different malware programs.

Building upon the ensemble approach of Narayanan and Davuluru, Wu et al. [16] have introduced a sequential architecture for malware classification. Their method integrated the temporal convolutional network (TCN) and bidirectional gated recurrent unit (BiGRU) models. The TCN network extracted temporal features through one-dimensional causal convolution, followed by the BiGRU capturing sequence information bidirectionally.

1.3 Limitations in Existing Research

Existing research in the field of malware classification has made significant progress, but there are still several gaps and limitations that need to be addressed.

6. An NLP technique that generates numerical representations for documents, capturing their semantic meaning and contextual information
7. A numerical statistic that quantifies the importance of a term in a document corpus
8. An NLP technique that combines Hidden Markov Models (HMMs) with the Word2Vec algorithm to learn word embeddings
9. An NLP technique that learns vector representations for words based on contextual usage
10. A machine learning technique that combines multiple individual classifiers to make predictions

One major gap is the limited integration of multiple data modalities in malware classification. Currently, many existing models either rely solely on tabular extracted features from malware executables or employ deep learning approaches that utilize byte images converted from the raw malware binaries. While these approaches have shown promising results, they often focus on analyzing either the structural information of the malware or the visual patterns captured in the byte images. There is a need for more comprehensive approaches that combine different data modalities. Integrating diverse data sources can provide a more holistic understanding of malware behavior and enable the identification of unique patterns and characteristics.

Furthermore, there is a lack of comparative analysis in existing research. While individual approaches have been explored, comparative studies that evaluate the performance of different algorithms, feature extraction methods, or data representations on a common benchmark dataset are valuable. Comparative analyses can provide insights into the strengths and weaknesses of different methodologies and guide future research directions.

1.4 Objectives and Scope of the Thesis

The objectives of this thesis are as follows:

- To leverage both byte and assembly formats of malware executables to extract valuable information, by analyzing malware at different levels of abstraction, a more comprehensive understanding of their behavior can be obtained. This approach enables the identification of unique characteristics and patterns that may not be evident when analyzing each modality seperately.

- To leverage the power of deep learning and feature engineering techniques in malware analysis, deep learning algorithms can automatically learn intricate patterns and relationships from large datasets, while feature engineering allows for the extraction of meaningful features that capture the essence of malware behavior. By combining these approaches, the analysis can benefit from the strengths of both methods.

- To compare the performance of the proposed model to a baseline model in terms of accuracy, precision, recall, and other relevant evaluation metrics. This comparison aims to assess the improved performance of the proposed approach in accurately classifying malware family compared to existing methods.

- To achieve high performance and develop a robust model capable of accurately classifying malware families and generalizing to new, unseen data, the effectiveness of the approach will be evaluated through testing and validation on real-world malware samples.

In this master thesis, we present a comprehensive approach for malware classification that integrates two distinct data modalities: images and tabular meta-data extracted from assembly files. Assembly features allow us to gain valuable insights into the nature of malware samples. While, byte files are best processed when converted into images, as images reveal clear visual patterns specific to each malware family. The extracted features and byte images are then fed into a deep learning model with two branches, enabling simultaneous processing of the data modalities for malware classification. To provide a comparative analysis, we also evaluate the performance of our approach against a Convolutional Neural Network (CNN) model that is trained solely on byte images.

1.5 Thesis Outline

The rest of this thesis is organized into several chapters, each addressing specific aspects of the study. Chapter 2 focuses on explaining the employed machine and deep learning methods in this study. Chapter 3 explains the suggested approach and describes the implementation steps, starting from the initial data preparation stage and progressing through the various stages, including feature extraction and model training. Chapter 4 discusses the performance of the proposed approach, evaluating its effectiveness. Finally, the last chapter concludes with key findings, draws conclusions, and outlines suggestions for future work.

Chapter 2

Theoretical Backgorund

The purpose of this chapter is to provide the necessary background knowledge and contextual understanding required to comprehend the proposed approach for malware classification presented in Chapter 3. It aims to present a comprehensive overview of the field and the employed machine and deep learning methods for malware classification.

The chapter theoretical background is structured as follows: Section 2.1 provides a comprehensive understanding of malware and its associated risks. In Section 2.2, the concept of malware classification is explained, highlighting the advantages of employing machine learning techniques in this particular domain. Sections 2.3 and 2.4 delve into the methods utilized for feature extraction and selection, while Section 2.5 focuses specifically on the exploration of deep learning methods employed in our study. Finally, Section 2.6 offers an in-depth explanation of the employed evaluation metrics and the loss function that has been utilized for training the deep learning classifiers.

2.1 Malware Overview

Malware, short for "malicious software", refers to any software or code designed to disrupt, damage, or gain unauthorized access to computer systems, networks, or devices. It is created with malicious intent and aims to compromise the confidentiality, integrity, or availability of digital information. Malware can take various forms, such as viruses[1], worms[2], trojans[3], ransomware[4], spyware[5], adware[6], or bots[7], and it often spreads through infected files, websites, email attachments, or software vulnerabilities.

Malware can execute a wide range of malicious activities on infected systems, including:

— **Data theft:** refers to the malicious activity conducted by certain types of malware, where sensitive information, such as login credentials and personal details, is stolen from compromised systems and is transmitted to the attacker [17].

— **System disruption:** malware can disrupt the normal functioning of a system by modifying or deleting files, corrupting data, or altering system configurations. This can lead to system crashes, loss of important files, or rendering the infected system unusable.

— **Remote control:** some malware are designed to provide unauthorised access to the machine it infects. This allows attackers to control the compromised system, use it for criminal activities, or exploit its resources to launch further attacks on other networks or systems [18].

— **Ransomware:** encrypts the victim's files and demands a ransom payment in exchange for the decryption key [19].

— **Adware:** displays unwanted advertisements or redirects users to malicious websites. It can lead to a degraded browsing experience, privacy invasion, and exposure to further malware infections [20].

These examples highlight the diverse range of malicious activities that malware can undertake, and the significant risks it poses to individuals, businesses, and organizations.

2.2 Malware Classification

Malware classification refers to the process of categorizing malicious software into distinct groups based on their characteristics, behavior, and functionalities. Malware variants belonging to the same family share similarities in their code, propagation methods, and malicious intent. By grouping malware into families, security practitioners can gain insights into the nature of threats and develop effective defense strategies.

1. Malicious programs that rely on human actions, such as opening an infected file or executing a malicious code, to spread.
2. Self-replicating malicious software that spreads across networks without user intervention.
3. Malware disguised as legitimate software or files, tricking users into executing them. Once activated, they can perform unauthorized actions
4. Malware that encrypts a victim's files or locks their system, demanding a ransom in exchange for restoring access.
5. Collects data from the victims device and send it to a third party without their consent
6. Software that displays unwanted advertisements, often in the form of pop-ups
7. Automated programs designed to exploit the resources of the victim's computer or network to perform specific tasks.

Utilizing machine and deep learning techniques for malware classification offers the following advantages over traditional employed approaches:

— **Automation of the process:** the complexity and diversity of modern malware strains, coupled with the anti-analysis techniques employed to conceal their true nature, present a challenge in manually defining and updating rules or signatures for each variant. Machine learning techniques address this challenge by automatically learning patterns and features from large-scale datasets. By training models on labeled malware samples, machine learning algorithms can identify distinguishing characteristics that differentiate malware families. Automating the classification process not only saves valuable time and effort but also allows for the efficient analysis of a vast number of malware samples.

— **Real-time response:** In today's rapidly evolving threat landscape, timely detection and response to malware attacks are critical. Manual analysis of samples can be time-consuming, averaging 30 to 45 minutes per sample. Moreover, the sheer volume of malware samples being generated daily makes it impractical to analyze and respond to each instance individually. By utilizing machine learning classification techniques, security practitioners can quickly classify malware into distinct categories, which enables real-time decision-making and response.

2.3 Feature Engineering

Feature engineering is the process of transforming raw data into informative features that accurately capture the relevant aspects of the underlying problem.

2.3.1 N-gram Method

N-gram method is a statistical technique used to extract sequences of N items, such as words or characters from sequential data.

Sequential Data

Sequential data refers to data in which the order of the data points holds significance and plays a pivotal role in comprehending the context. For instance, consider the following sentence:

"The algorithm sorts the array in ascending order"

In this sentence, the specific arrangement of the words is essential for interpreting its meaning. Any alteration to the word order would result in a change of semantics. For instance, if we were to reorganize the words as *"In ascending order the array sorts the algorithm"* the sentence would convey a different message. Similarly, if the word order is changed to *"Algorithm the in array the order sorts ascending"* the sentence would no longer make sense.

In the upcoming methodology chapter, we will be extracting opcode sequences from assembly executables. Opcodes are the fundamental instructions executed by a computer's processor. Consider the following assembly instructions:

```
MOV AX, 5
ADD BX, AX
SUB CX, BX
```

Where MOV, ADD, and SUB represent opcodes, and AX, BX, CX are registers.

When the previous instructions are executed:

— The opcode MOV assigns the value 5 to the AX register.

— The opcode ADD adds the value of AX (5) to BX, resulting in $BX = 5$.

— The opcode SUB subtracts the value of BX (5) from CX, resulting in $CX = -5$.

Thus, the output of the original opcode sequence is $AX = 5$, $BX = 5$, and $CX = -5$. If we were to change the order of the opcodes, the program's behavior and outcome would be completely different. For instance, if we rearrange the opcodes as follows:

```
ADD BX, AX
SUB CX, BX
MOV AX, 5
```

The resulting output of the rearranged opcode sequence is $AX = 5$, $BX = 0$, and $CX = 0$. The arrangement of opcodes influences the behavior and final state of a program. Thus, we conclude the sequential nature of opcode sequences, as a result, the application of N-grams can extract informative features from opcode sequences.

Words Sequence to N-gram Counts

To extract n-gram counts from opcode sequences, a sliding window technique is employed. A sliding window of size n is applied to the sequence, where n represents the desired length of the n-grams. The window moves one opcode at a time, capturing the current n-gram. As the window slides through the sequence, each encountered n-gram is recorded and its frequency is counted. This process continues until the window reaches the end of the sequence. The n-gram counts are then calculated by tallying the frequencies of each unique n-gram. The frequencies indicate how often a particular n-gram appears in the sequence.

Let us now outline the process of extracting n-gram counts step-by-step, using the sentence "The system logs record the system events" as an illustrative example.

0. **Sliding Window:** To begin, we employ a sliding window technique with a window size of n, representing the desired length of the n-grams. In this case, we consider bigrams (n=2). The sliding window moves through the sentence, capturing groups of two consecutive words resulting in the following bigrams:
   — "The system"
   — "system logs"
   — "logs record"
   — "record the"
   — "the system"
   — "system events"

0. **Counting Frequencies:** Next, we count the frequencies of each unique n-gram encountered in the sentence. In other words, we count how many times each n-gram appears.

   — "The system": 2

   — "system logs": 1

   — "logs record": 1

   — "record the": 1

   — "system events": 1

In this example, the bigram "The system" occurs twice in the sentence, indicating the repetition of this word pair.

By following this process, we can extract the n-gram counts and then utilize them as input features for further analysis or model training.

2.4 Feature Selection

To enhance the performance of our machine learning model, we employ feature selection techniques using random forests. Feature selection reduce dimensionality, improve model efficiency, and identify the most relevant features for prediction. Random forests, with their ability to measure feature importance, provide a suitable approach for this purpose.

2.4.1 Random Forest

Random forests are an ensemble learning method that combines multiple decision trees to make predictions [21]. Each decision tree in the forest is constructed using a bootstrap sample of the training data and a random subset of features at each split. The final prediction is obtained through aggregation, such as voting or averaging, across all the individual trees. A key advantage of random forests is their ability to assess feature importance, which gives information about the relevance of variables in the prediction process. Feature importances are calculated as follows:

Step 1: Gini Impurity Calculation at Each Split

Gini impurity is a measure of impurity or disorder within a set of samples. It quantifies the probability of incorrectly classifying a randomly selected sample based on the distribution of class labels within the set. For a given node in a decision tree, the Gini impurity (I) is calculated as follows:

$$I = 1 - \sum_i (p_i)^2 \tag{2.1}$$

In a random forest, each decision tree undergoes recursive splits to partition the data. At each split, the Gini impurity is calculated for the parent node and the resulting child nodes.

For a specific feature used for splitting, the Gini impurity of the parent node (I_{parent}) is calculated using the class label distribution. Then, the Gini impurity of each resulting child node (I_{child}) is calculated in the same way.

Step 2: Calculating Impurity Decrease

To assess the impact of a feature on reducing impurity, the impurity decrease is calculated. It represents the difference between the Gini impurity of the parent node and the weighted average of the child node impurities. This can be expressed as:

$$\Delta I = I_{\text{parent}} - \sum \left(\frac{n_{\text{child}}}{n_{\text{parent}}} \right) \cdot I_{\text{child}} \tag{2.2}$$

where n_{child} represents the number of samples in the child node and n_{parent} represents the number of samples in the parent node.

Step 3: Average Impurity Decrease across Trees

To obtain the feature importance score, the average impurity decrease across all decision trees in the random forest is calculated. This provides an overall measure of how much each feature contributes to reducing impurity and improving predictive accuracy.

The feature importance score for a specific feature is calculated by summing up the impurity decreases for that feature across all trees and dividing it by the number of trees.

Step 4: Normalizing Feature Importance Scores

The feature importance scores obtained from the average impurity decrease can be normalized to facilitate easier interpretation and comparison. This step is optional but can be helpful in presenting the relative importance of features.

2.4.2 Justification and Discussion

For the selection of the best features from the rest of the features we used Random Forest due to its reliability, computational efficiency, and ease of interpretation.

Firstly, Random Forest has proven to be a highly reliable method for feature selection. It leverages the power of an ensemble of decision trees to reduce the risk of overfitting and provide robust results. By considering multiple trees and averaging their predictions, it captures the collective wisdom of the forest, a concept we refer to as *the wisdom of the crowd.*

Secondly, Random Forest is less computationally expensive compared to other methods used for feature selection like wrapper-based methods.

Lastly, Random Forest offers a straightforward interpretation of feature importance. It calculates the average decrease in node impurity (e.g., Gini impurity or entropy) caused by a feature, providing a measure of its relevance in the classification or regression task. This intuitive ranking of feature importance allows to identify the most influential features easily, aiding in the selection of the best features.

2.5 Deep Learning

In our study (Chapter 3), we have employed different deep learning models, namely Feed-Forward Network, CNNs, and Multimodal. These networks are trained using an optimizer on a loss function, and the weights are updated through the backpropagation process. The final predictions are obtained through forward propagation of inputs toward the output. In this section, we will delve into the working mechanisms of these concepts.

2.5.1 Feed-Forward Network

A neural network is a computational model composed of interconnected layers of artificial neurons that maps the input features X to the target variable Y. It employs activation functions within each neuron to facilitate complex transformations of the input data and introduce non-linearity.

A Feed-Forward network is a fundamental type of artificial neural network. It consists of an input layer, hidden layers, and an output layer. Information flows through the hidden layers from the input layer to the output layer without any feedback loops. In other words, the information moves through the network in a one-way manner, passing through the layers sequentially without any backward connections. Figure 2.1 shows an example feed-forward network architecture.

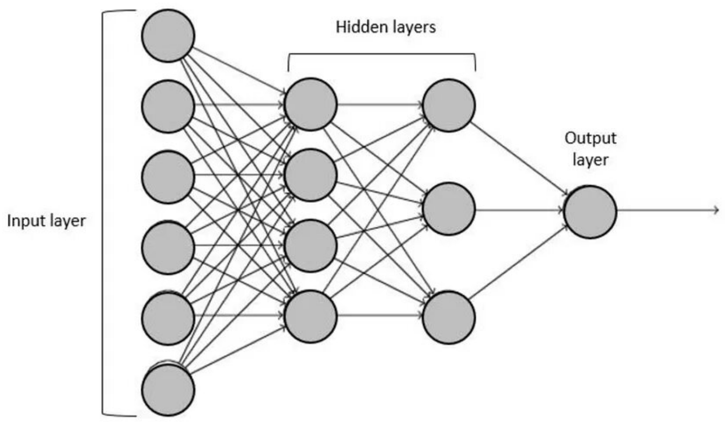


Figure 2.1 – Feed-forward neural network.

Sequential networks like Recurrent Neural Networks (RNNs) are examples of networks that are not feed-forward because they contain feedback connections, which enable them to incorporate information from previous time steps and capture temporal dependencies in sequential data. Figure 2.2 illustrates the difference between feed-forward networks and RNNs.

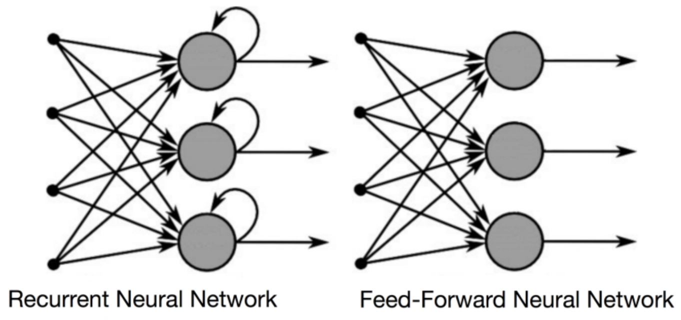


Figure 2.2 – RNN vs FNN networks.

Forward Propagation

During feed-forward, the network applies a series of mathematical operations, such as matrix multiplications and activation functions, to transform the input data. As the data passes through the network layers, it undergoes a hierarchical feature extraction process, where lower layers capture simple features and higher layers capture more complex features. By propagating input data through the network, the feed-forward process generates output predictions in the final layer.

Let's consider a simple Feed-Forward network with one hidden layer:

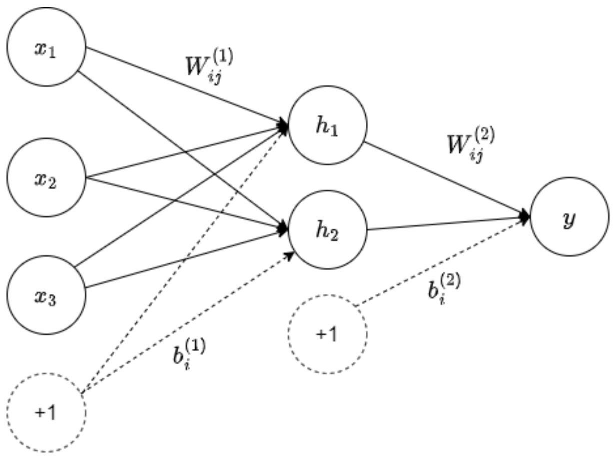


Figure 2.3 – FNN with one hidden layer.

The symbols in Figure 2.3 refer to:

— x: Input vector
— h: Hidden layer
— y: Output layer
— $W^{(1)}$: Weight matrix connecting input layer to hidden layer
— $b^{(1)}$: Bias vector of the hidden layer
— $W^{(2)}$: Weight matrix connecting hidden layer to output layer
— $b^{(2)}$: Bias vector of the output layer

— σ: Activation function

The hidden layer performs calculations to transform the input. It takes the input vector, x, and applies a weight matrix, $W^{(1)}$, which represents the connections between the input layer and the hidden layer. This multiplication is followed by the addition of a bias vector, $b^{(1)}$, specific to the hidden layer. The result is passed through an activation function, σ, to introduce non-linearity and generate the activations of the hidden layer neurons.

$$h = \sigma(W1 \cdot x + b1) \tag{2.3}$$

The activations of the hidden layer neurons are further processed in the output layer. The hidden layer activations, h, are multiplied by a weight matrix, $W^{(2)}$, representing the connections between the hidden layer and the output layer. Additionally, a bias vector, $b^{(2)}$, specific to the output layer, is added. The resulting values are passed through an activation function, σ, to obtain the final output activations, y.

$$y = \sigma(W2 \cdot h + b2) \tag{2.4}$$

Backpropagation

Backpropagation is an algorithm used for training neural networks by updating the model's parameters to enhance the accuracy of the network's predictions. It consists of the following steps:

0. **Forward Pass:** The network generates predictions in the output layer through forward propagation, where input data flows through the network and activations are computed layer by layer.
0. **Loss Calculation:** The loss function L is then utilized to measure the discrepancy between the predicted outputs and the true labels.
0. **Backward Pass:** In the backward pass, the gradients of the weights with respect to the loss are computed. These gradients indicate how the loss changes with respect to each weight in the network.
0. **Weight Update:** With the gradients calculated, an optimizer algorithm, such as stochastic gradient descent (SGD) or Adam, is employed to update the weights. The weights are updated according to the following equation:

   $$W_{\text{new}} = W - \eta \cdot \nabla W \tag{2.5}$$

   Here, W_{new} represents the updated weight, W is the current weight, η is the learning rate, and ∇W represents the gradient of the weight.
0. **Iterative Process:** Steps 1-4 are repeated for multiple iterations or epochs until the network's performance converges or reaches a satisfactory level. Each iteration involves another forward pass, loss calculation, backward pass, and weight update.

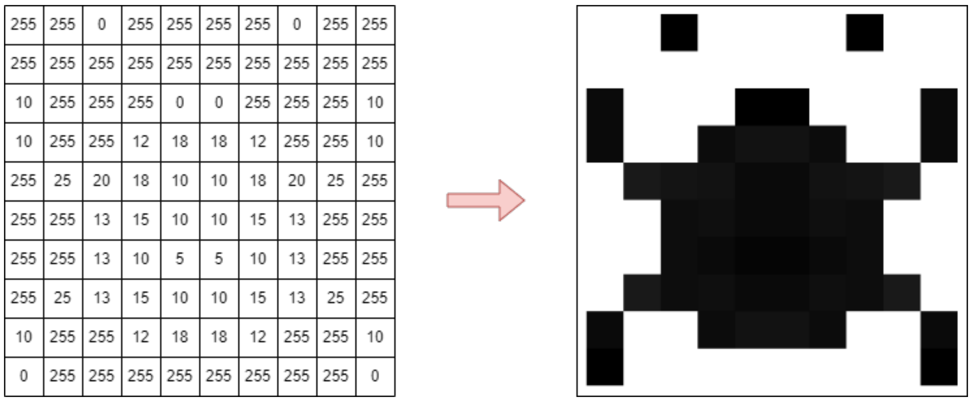
   By iteratively adjusting the weights based on the computed gradients, the network gradually improves its predictions and minimizes the loss function.

2.5.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are deep learning algorithms widely used for analyzing and processing visual data, particularly images [22, 23]. In the realm of digital images, visual content is often represented as matrices of pixels. Grayscale images, which possess varying intensities of black and white, can be represented as 2D matrices, with each pixel corresponding to a specific location and holding a value between 0 and 255.

Color images, on the other hand, are represented as 3D matrices. Each pixel in a color image is composed of three color channels: red, green, and blue (RGB). The combination of these channels determines the color of the pixel. For instance, a specific pixel may have values of (255, 0, 0) in the RGB channels, representing pure red.

In Figure 2.4, we can observe a 2D matrix representation of a malware shape alongside its corresponding grayscale image.



255	255	0	255	255	255	255	0	255	255
255	255	255	255	255	255	255	255	255	255
10	255	255	255	0	0	255	255	255	10
10	255	255	12	18	18	12	255	255	10
255	25	20	18	10	10	18	20	25	255
255	255	13	15	10	10	15	13	255	255
255	255	13	10	5	5	10	13	255	255
255	25	13	15	10	10	15	13	25	255
10	255	255	12	18	18	12	255	255	10
0	255	255	255	255	255	255	255	255	0

Figure 2.4 – An example 10 $\times$ 10 2D matrix (left) and its image representation.

CNNs consist of two main parts: the convolutional part and the fully connected layers. The convolutional part comprises mainly two types of layers: convolutional layers and pooling layers.

Convolutional Layers

Convolutional layers apply filters, also known as kernels, to the input images. These filters capture local patterns and features through a mathematical operation called convolution. Mathematically, the fully connected layers can be represented as a sequence of matrix multiplications and nonlinear activation functions:

$$(f * g)(i, j) = \sum_m \sum_n f(m, n) \cdot g(i - m, j - n) \tag{2.6}$$

where f represents the input image, g denotes the filter, and $(f * g)$ denotes the resulting feature map. The operation involves element-wise multiplication between the filter weights $g(i - m, j - n)$ and the corresponding input pixel values $f(m, n)$, followed by summation over the filter's spatial dimensions m and n as shown in Figure 2.5.

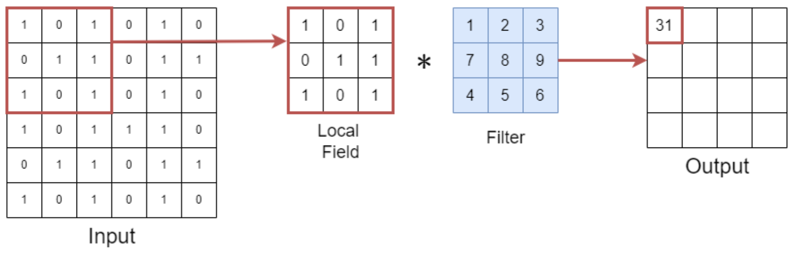


Figure 2.5 – Convolution operation.

Pooling layers

Pooling layers, typically placed between convolutional layers, applied to reduce the spatial dimensionality of the convolved feature maps. They achieve this by subsampling the feature maps using operations like max pooling, or average pooling.

In max pooling, the maximum value within each pooling region is selected, capturing the most prominent features. On the other hand, average pooling calculates the average value within each region, summarizing the information across the pooling region. These pooling operations help reduce computational complexity while retaining essential information about the detected features. Figure 2.6 illustrates an example of calculations of max and average pooling.

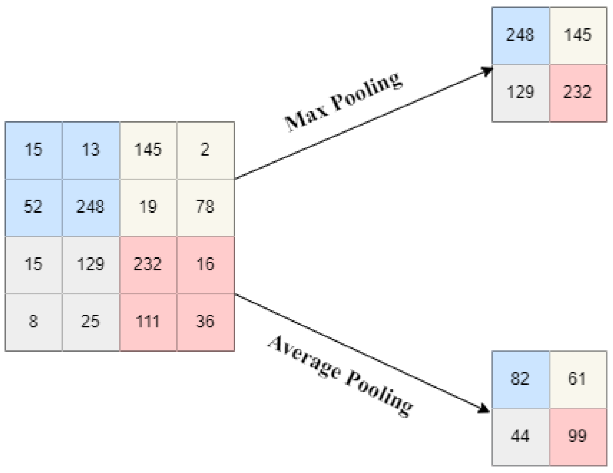


Figure 2.6 – Example of max and average pooling.

The output of the convolutional layers is then flattened, converting the high-dimensional feature maps into a vector form. This vector is then fed into fully connected layers, also known as dense layers. These layers make predictions based on the extracted information.

The general process of a CNN can be summarized as follows:

0. Apply filters to the input image, capturing local patterns and features.
0. Subsample the resulting feature maps using pooling operations in order to reduce dimensionality.
0. Repeat the convolution and pooling steps, stacking multiple layers, to extract increasingly higher-level and abstract features from the input data.

0. Flatten the extracted features and pass them through fully connected layers to predict the output.

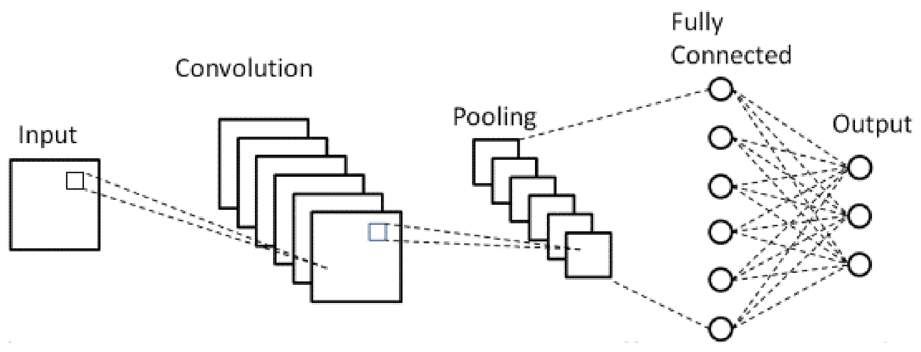


Figure 2.7 – Basic CNN architecture.

2.5.3 Multimodal Networks

Unimodal deep learning approaches have their limitations, particularly when it comes to tackling complex tasks. In the field of image classification, for instance, traditional deep learning models solely trained on images may struggle to accurately predict the content in intricate cases. This is due to the models' inherent limitations in capturing sufficient context or information from images alone.

To address these limitations, multimodal neural networks have been developed [24, 25]. These networks are specifically designed to process and integrate information from multiple modalities, such as visual, textual, and auditory data. By combining these different modalities, multimodal deep learning can capture a more comprehensive representation of the environment and gain access to a richer set of information, thereby enhancing performance across various tasks.

A typical multimodal architecture consists of three essential components: unimodal encoders, a fusion network, and a classifier.

0. **Unimodal encoders:** encode each input modality separately, extracting relevant features.
0. **Fusion network:**combines the extracted features from each modality to create a unified representation of the data.
0. **Classifier:** utilizes the fused data to make predictions for the given task.

An example that demonstrates the effectiveness of multimodal deep learning is the diagnosis of COVID-19 pneumonia. In a study conducted by Hilmizen et al. [26], they proposed a multimodal approach that combines CT scans and chest X-ray images as illustrated in Figure 2.8. The multimodal model achieved a classification accuracy of 99.87%, outperforming models trained solely on CT scans (98% accuracy) or X-rays (98.93% accuracy). This study highlights the significant improvement offered by multimodal deep learning compared to relying on a single modality.

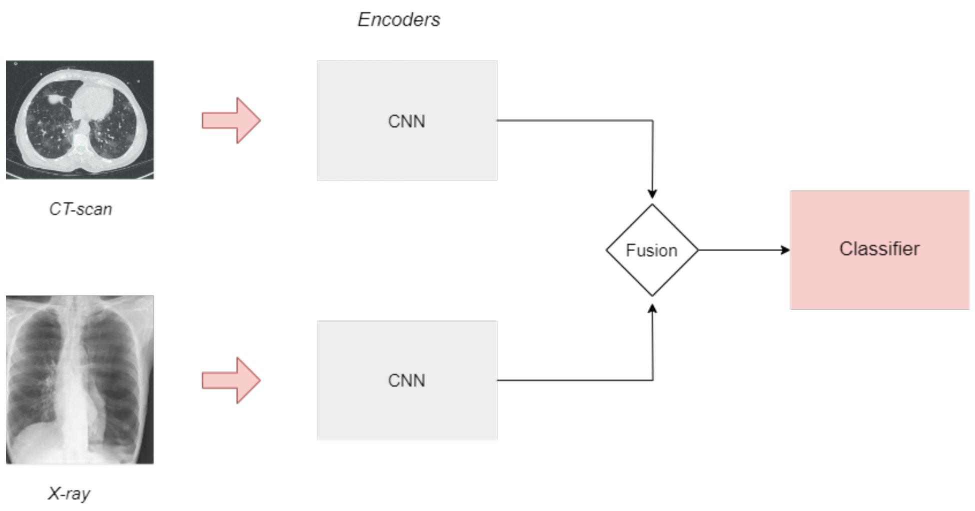


Figure 2.8 – Sample multimodal for COVID-19 diagnosis.

2.6 Loss Function and Evaluation Metrics

2.6.1 Loss Function: Categorical Cross-Entropy

To achieve accurate predictions of the target variable, deep learning neural networks learn the optimal weights through an optimization process. The selected optimization algorithm iteratively updates the weights to minimize the loss function, allowing the neural network to continually refine its predictions and improve overall performance.

In the context of malware classification, we employed the cross-entropy loss function [27]. We specifically utilized the categorical cross-entropy variant of the loss function, which is tailored for handling multiple classes. The categorical cross entropy loss function is defined as follows:

$$\text{Loss} = -\sum_{i=1}^{\text{output size}} (y^{(i)} \cdot \log(p^{(i)})) \tag{2.7}$$

where y represents the true class labels and p represents the predicted class probabilities.

The cross-entropy loss function heavily penalizes large differences between the predicted probabilities and the true class labels. This behavior is due to the nature of the logarithm function used in the formulation of the loss.

As shown in Figure 2.9, when the input to the logarithm function becomes smaller and approaches 0 from the positive side, the natural logarithm (ln(x)) becomes progressively more negatively larger. Conversely, as the input approaches 1, the value of logarithms approaches 0.

This implies that when the model confidently assigns a high probability to the true class, the loss is minimized. Conversely, the loss value increases significantly as the assigned probability to the true class decreases. This means that the model is penalized more for incorrect or uncertain predictions, encouraging it to strive for higher confidence and accuracy.

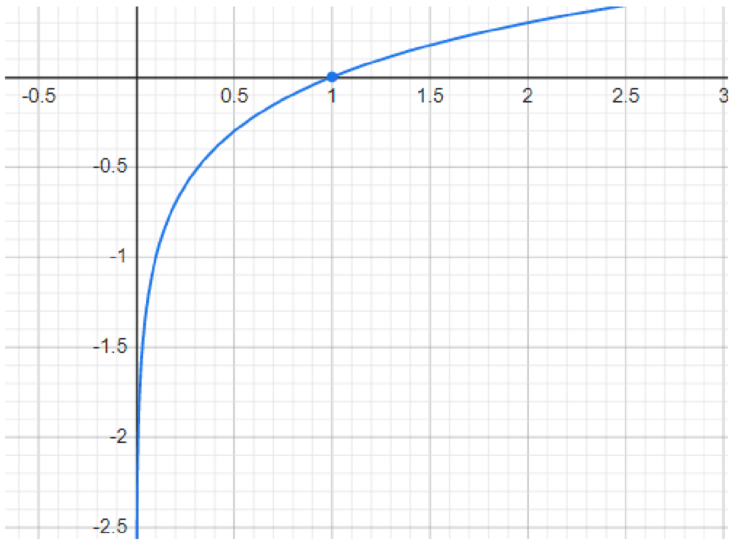


Figure 2.9 – Natural logarithm function.

To further illustrate the behavior of the cross-entropy loss function, let's consider a classification problem with three classes: A, B, and C. Suppose we have a sample instance that belongs to class A, and the true label is denoted as [1, 0, 0], indicating the ground truth probability distribution across the classes.

let's assume the model confidently assigns a probability of 0.9 to class A, 0.05 to class B, and 0.05 to class C, which means that the model's prediction is accurate. The log loss using the cross-entropy formula can be computed as follows:

$$\begin{aligned} \text{Loss} &= -\sum_{i=1}^{\text{output size}} (y^{(i)} \cdot \log(p^{(i)})) \\ &= -(1 \cdot \log(0.9) + 0 \cdot \log(0.05) + 0 \cdot \log(0.05)) = -\log(0.9) = 0.105 \end{aligned} \tag{2.8}$$

In the given equation, ”y” represents the target class or true class label, while ”p” denotes the probability of the class predicted by the model.

In another scenario, suppose the model assigns probabilities of 0.15 to class A, 0.4 to class B, and 0.45 to class C. The log loss can be calculated as:

$$\begin{aligned} \text{Loss} &= -\sum_{i=1}^{\text{output size}} (y^{(i)} \cdot \log(p^{(i)})) \\ &= -(1 \cdot \log(0.15) + 0 \cdot \log(0.4) + 0 \cdot \log(0.45)) = -\log(0.9) = 1.897 \end{aligned} \tag{2.9}$$

The log loss is lower (0.105) when the model's prediction aligns accurately with the true class, but higher (1.897) when the prediction deviates from the true label, indicating the significance of the loss in reflecting the model's accuracy.

2.6.2 Evaluation Metrics

When evaluating the proposed models and for meaningful comparisons, several metrics were utilized. These metrics include:

Accuracy

Accuracy measures the overall correctness of the model's predictions. It calculates the ratio of correctly classified instances to the total number of instances. The formula for accuracy is:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{2.10}$$

Accuracy provides a general overview of the model's performance in terms of correct classification. However, it may not be the ideal metric when dealing with imbalanced datasets [28].

Recall

Also known as sensitivity or true positive rate, measures the model's ability to correctly identify positive instances. It calculates the ratio of correctly identified positive instances to the total number of actual positive instances [29]. The formula for recall is:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{2.11}$$

Precision

Precision quantifies the model's ability to accurately classify positive instances. It calculates the ratio of correctly identified positive instances to the total number of instances predicted as positive [29]. The formula for precision is:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{2.12}$$

F1-Score

Also referred to as F-measure. The F1 score is calculated using the following formula:

$$\text{F1 Score} = 2\frac{\text{Precision}\text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.13}$$

The F1 score is particularly useful when the dataset is imbalanced which is the case for the task at hand. It provides a single value that represents the trade-off between precision and recall.

2.7 Conclusion

This chapter offers an overview of the essential concepts necessary for understanding the proposed methods for malware classification. It covers the employed feature engineering and selection algorithms, evaluation metrics, and deep learning techniques. Additionally, it provides the necessary context for understanding the problem of malware categorization. In the next chapter, we will delve into the detailed explanation of our proposed methods for classifying malware, outlining the specific approaches, algorithms, and techniques utilized to address the challenge effectively.

Chapter 3

Proposed Methodology for Malware Classification

In this chapter, we present our approach for malware classification using deep learning techniques. We propose two different methods for malware classification. The first method involves training a CNN on byte images extracted from malware's byte file. For the second method, we extract relevant features from the malware's assembly file. These features capture important characteristics and patterns within the code. A multimodal is then trained using two inputs: the byte images and the extracted features from the assembly file. By combining both the visual representations and the assembly-extracted features, we aim to leverage the complementary information and enhance the classification accuracy. To ensure optimal performance, both models undergo fine-tuning, including iterative adjustments of their parameters and architectures.

Implementing these two distinct methods for malware classification allows to compare the CNN, which is a single-modality model, with the multimodal that incorporates two different data modalities: images and tabular data. This comparison enables the examination of the performance differences between these approaches and assess the benefits of integrating multiple data sources for malware classification.

Throughout this chapter, we aim to provide a thorough understanding of the proposed methods for malware family classification. We discuss the techniques and methodologies employed in detail. Section 3.1 focuses on describing the data utilized for this classification task. Moving forward, in Section 3.2, we provide a comprehensive overview of the first proposed method, while in Section 3.3, we present the second method employed for malware classification.

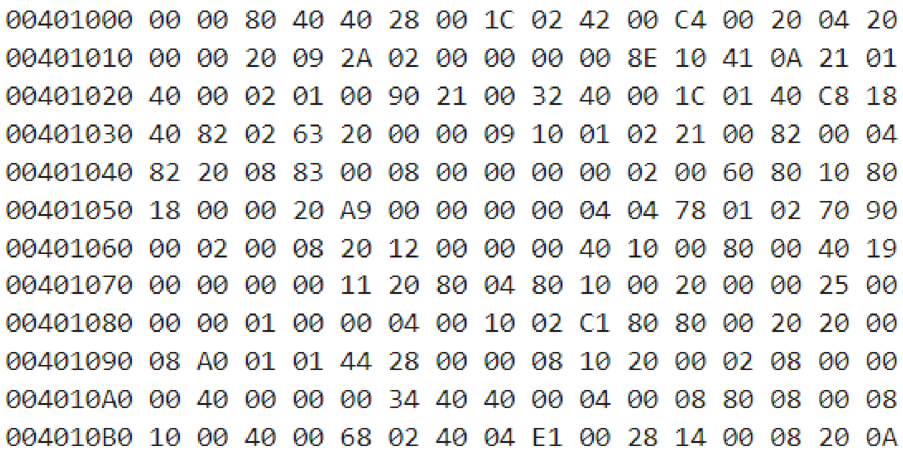
3.1 Data Description

The present work utilizes a dataset provided by Microsoft [30] and acquired through the Kaggle Platform [31], consisting of 10,868 executable malware records. Each record is represented by two files: a binary file containing the raw bytes of the file, and an assembly file containing the corresponding disassembly code.

Additionally, for testing the trained model, a separate dataset from Kaggle was provided, consisting of 10,873 executables. This test dataset does not contain the corresponding labels, as it is intended solely for evaluating the performance of the model.

Byte Files

Initially, malware is written in high-level programming languages such as C and Java and is then compiled. The compilation process transforms the source code into machine code. Machine code represents a low-level binary format that can be executed by the CPU. In this context, the resulting machine code is often referred to as a byte file. As depicted in Figure 3.1, a snippet of a sample malware machine code is illustrated in hexadecimal format.

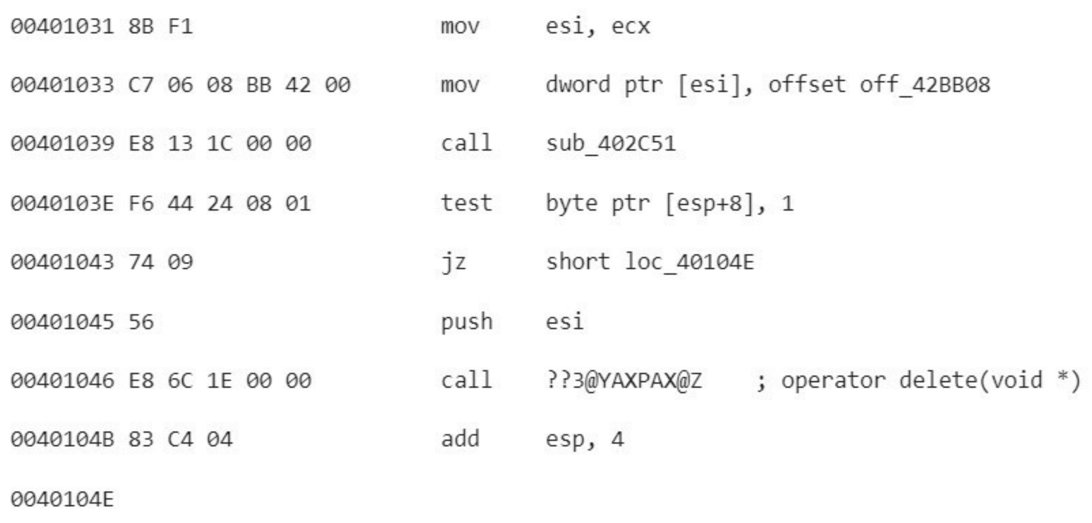


```
00401000 00 00 80 40 40 28 00 1C 02 42 00 C4 00 20 04 20
00401010 00 00 20 09 2A 02 00 00 00 00 8E 10 41 0A 21 01
00401020 40 00 02 01 00 90 21 00 32 40 00 1C 01 40 C8 18
00401030 40 82 02 63 20 00 00 09 10 01 02 21 00 82 00 04
00401040 82 20 08 83 00 08 00 00 00 00 02 00 60 80 10 80
00401050 18 00 00 20 A9 00 00 00 00 04 04 78 01 02 70 90
00401060 00 02 00 08 20 12 00 00 00 40 10 00 80 00 40 19
00401070 00 00 00 00 11 20 80 04 80 10 00 20 00 00 25 00
00401080 00 00 01 00 00 04 00 10 02 C1 80 80 00 20 20 00
00401090 08 A0 01 01 44 28 00 00 08 10 20 00 02 08 00 00
004010A0 00 40 00 00 00 34 40 40 00 04 00 08 80 08 00 08
004010B0 10 00 40 00 68 02 40 04 E1 00 28 14 00 08 20 0A
```

Figure 3.1 – Byte file.

Assembly Files

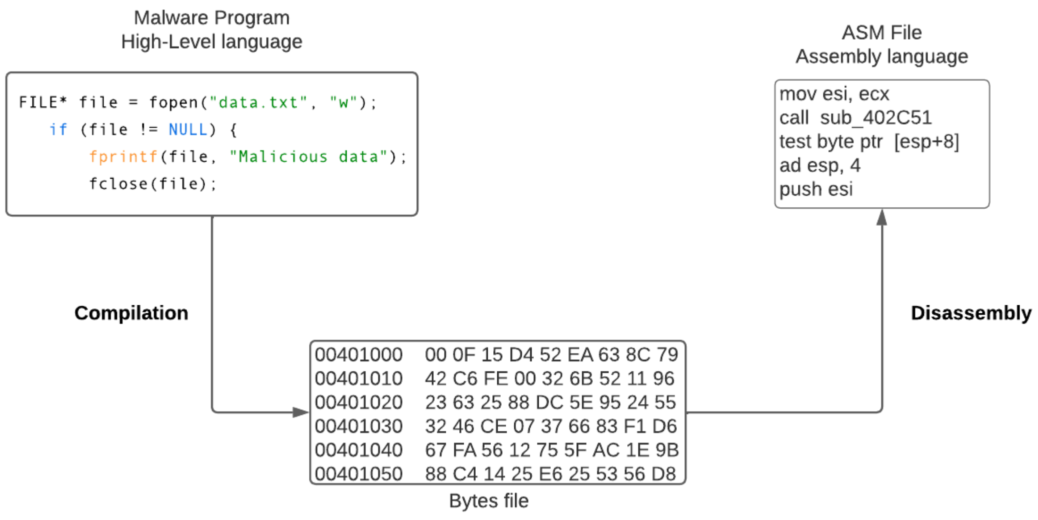
To facilitate the comprehension of the machine code, malware analysts employ disassembler tools like IDA Pro [32, 33] to generate assembly files from the byte files. Since machine code is not easily readable by humans, converting it into assembly language becomes essential. Figure 3.2 showcases a representative example of an assembly file snippet.



```
00401031 8B F1                mov    esi, ecx
00401033 C7 06 08 BB 42 00    mov    dword ptr [esi], offset off_42BB08
00401039 E8 13 1C 00 00       call   sub_402C51
0040103E F6 44 24 08 01       test   byte ptr [esp+8], 1
00401043 74 09                jz     short loc_40104E
00401045 56                   push   esi
00401046 E8 6C 1E 00 00       call   ??3@YAXPAX@Z    ; operator delete(void *)
0040104B 83 C4 04             add    esp, 4
0040104E
```

Figure 3.2 – Assembly file.

Assembly language represents a more human-readable format that preserves the underlying information about the malware's behavior. The assembly file contains instructions and other information about the program's execution. This conversion enables analysts to comprehend the intricate workings of the malware. Figure 3.3 provides a visual representation of the relationship between the byte file and its corresponding assembly file.



```
FILE* file = fopen("data.txt", "w");
   if (file != NULL) {
       fprintf(file, "Malicious data");
       fclose(file);
```

```
mov esi, ecx
call  sub_402C51
test byte ptr  [esp+8]
ad esp, 4
push esi
```

```
00401000    00 0F 15 D4 52 EA 63 8C 79
00401010    42 C6 FE 00 32 6B 52 11 96
00401020    23 63 25 88 DC 5E 95 24 55
00401030    32 46 CE 07 37 66 83 F1 D6
00401040    67 FA 56 12 75 5F AC 1E 9B
00401050    88 C4 14 25 E6 25 53 56 D8
```

Figure 3.3 – Byte to assembly conversion.

The dataset is structured as a multiclass classification problem, with each sample belonging to one of nine malware families: Ramnit, Lollipop, Kelihos-ver3, Vundo, Simda, Tracur, Kelihos-ver1, Obfuscator.ACY, and Gatak. Table 3.1 summarizes the characteristics of each of the nine malware families.

Malware Family	**Type**	**Characteristics**
Ramnit	Worm/Trojan	- Spreads through various methods such as email. attachments, drive-by downloads, and software vulnerabilities. - Steals sensitive information and can transform infected machines into a botnet.
Lollipop	Adware/Trojan	- Displays pop-up advertisements on the browser. - Often disguised as legitimate apps. - Can perform various malicious activities such as data theft or unauthorized access.
Kelihos-ver3	Botnet	- Creates a botnet network of infected computers. - Used for spam campaigns, distributing other malware, and executing DDoS attacks.
Vundo	Trojan	- Delivers pop-up advertisements.. - Modifies system files and registry entries. - Can download and install additional malware.
Simda	Trojan	- Spreads through exploit kits and infected websites. - Can download and execute additional malware on compromised systems.

Malware Family	**Type**	**Characteristics**
Tracur	Trojan	- Password-stealing trojans. - Enable a malicious hacker to gain backdoor access and control over a targeted PC.
Kelihos-ver1	Botnet	- Infected systems used for spam campaigns, distributing malware, and carrying out various malicious activities.
Obfuscator.ACY	Trojan	- Uses obfuscation techniques to make malware code difficult to analyze and detect.
Gatak	Trojan	- Targets businesses and organizations. - Often distributed through phishing emails. - Can steal sensitive information and perform espionage.

Table 3.1 – Malware families characteristics.

The total size of the dataset is around 400GB with approximately 200 GB for both the training and testing sets.

In the following sections, we present the methodology of the proposed approaches for malware classification.

3.2 Method 1: Malware Classification using CNN

Convolutional Neural Networks have gained significant attention in recent years for their ability to automatically learn and understand complex patterns in images. CNNs have also found extensive use in the field of malware classification, several notable studies, such as the work of Kalash et al. [34] and Kornish et al. [35], have demonstrated the effectiveness of CNN-based approaches in detecting and classifying malware.
In our first proposed method, we aim to reproduce their work by employing a similar approach. We will start by converting malware executables into image representations. Then, we will utilize a CNN to analyze these images and classify them accordingly.

3.2.1 Malware Bytes to Image Conversion

Visualizing malware allows for better differentiation between different malware families. Malicious programs belonging to the same family often share visual similarities and exhibit distinct patterns compared to images associated with other malware families. This concept is supported by the work of Nataraj et al. In their paper 'Malware images: visualization and automatic classification' [13], where they demonstrated the effectiveness of visual representations for distinguishing between various malware families.

In what follows, we explain the conversion process of machine code (byte files) into images. The provided byte files comprise sequences of hexadecimal values, ranging from 0 to F, along with the character '?'. Each consecutive pair of hexadecimal values is converted into its respective decimal representation, which spans the range of 0 to 255, while each pair of the character '?' is mapped to the value 0. These decimal values represent the intensity or color value of each pixel.

During the conversion process of the byte file, each calculated decimal value is appended to a list of pixel values. This accumulation of pixel values forms a sequential representation of the malware's visual characteristics. The resulting sequence of pixel values is reshaped into a matrix and further transformed into an image.

The dimensions of the reshaped matrix are not chosen randomly; they are determined based on the total number of pixels. To reshape the sequence into a square matrix, the square root of the total number of pixels is calculated. The integer part of the square root determines one dimension of the square matrix. To determine the second dimension, the total number of pixels is divided by the value of the first dimension, considering only the integer part. Any remaining pixels that cannot be evenly accommodated in the matrix are discarded.

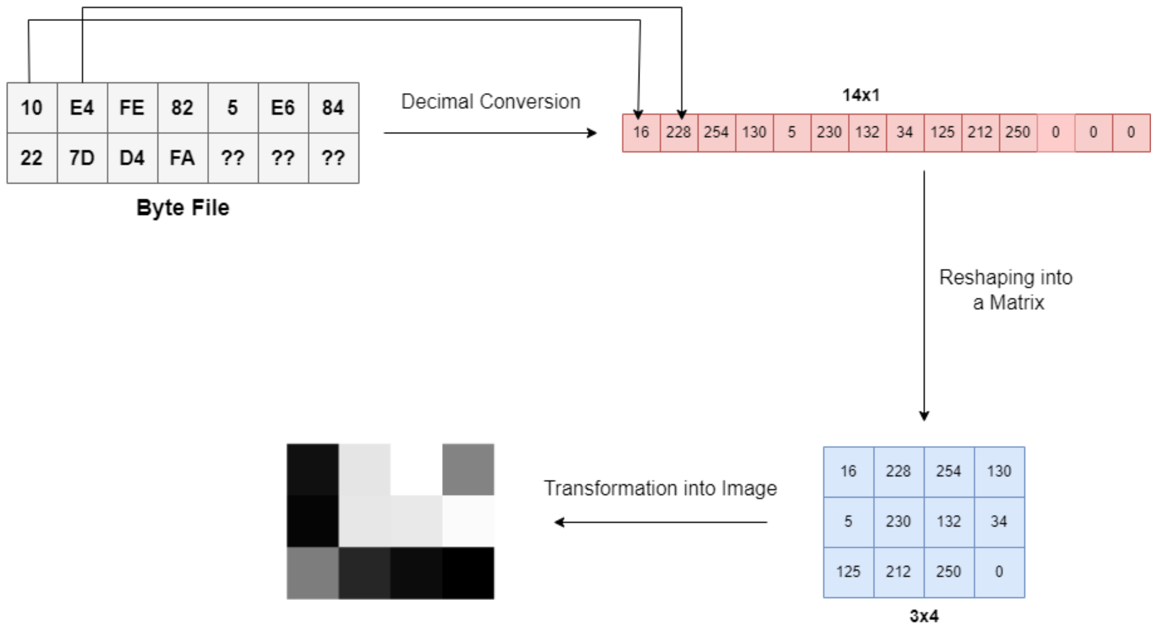


Figure 3.4 – Illustrative example: byte file to image conversion.

In the provided example illustrated in Figure 3.4, the byte file yields a total of 14 pixels for extraction. Consequently, the width of the resulting reshaped image is determined as the integer value of the square root of 14, resulting in a width of 3. The height of the image is obtained by dividing the total number of pixels by the width, resulting in a height of 4 and two discarded pixel values. Therefore, the size of the reshaped matrix is (3, 4).

To maintain uniformity, the resulting image is resized to a standardized dimension of 64x64, ensuring that all images share the same shape. These images help us visualize the patterns and structures within the malware, allowing for easier differentiation between various malware families. Figure 3.5 shows a sample of the resulting images from each malware family.

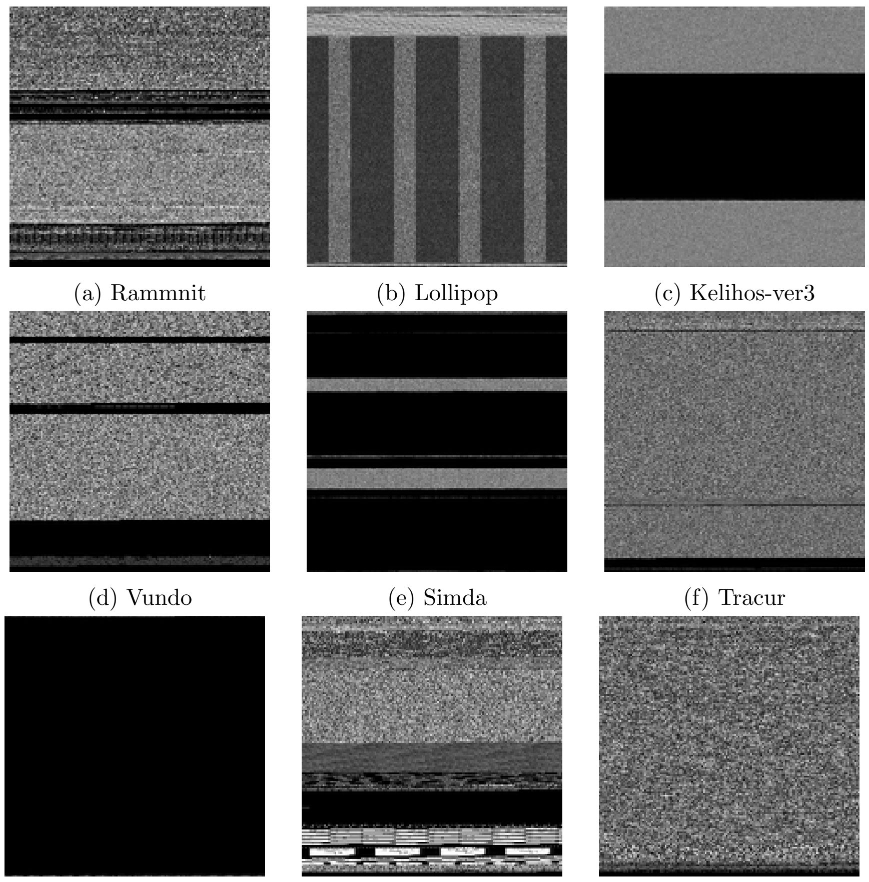

(a) Rammnit (b) Lollipop (c) Kelihos-ver3
(d) Vundo (e) Simda (f) Tracur
(g) Kelihos-ver1 (h) Obfuscator.DCY (i) Gatak

Figure 3.5 – Sample malware images by family.

Pixel values in an image typically range from 0 to 255, representing the intensity of gray levels in grayscale images. We normalize the pixel values through dividing them by 255.0. This scaling process ensures that the range of pixel values is within 0 to 1.

By dividing the pixel values by 255.0, we effectively map the original intensity values to a normalized range where 0 represents the lowest intensity (black) and 1 represents the highest intensity (white). This normalization [36] allows for better convergence during training and improves the overall performance of convolutional neural networks (CNNs).

3.2.2 Architecture of the Proposed Model

The proposed CNN architecture consists of four convolutional blocks, each consisting of a convolutional layer followed by batch normalization, rectified linear unit (ReLU) activation, max pooling, and dropout. These blocks progressively extract and refine relevant features from the input image data. Additionally, the model includes two dense layers with a decreasing number of neurons, contributing to the final classification task with a total number of 948,169 trainable parameters. The CNN's architecture is shown in Figure 3.6.

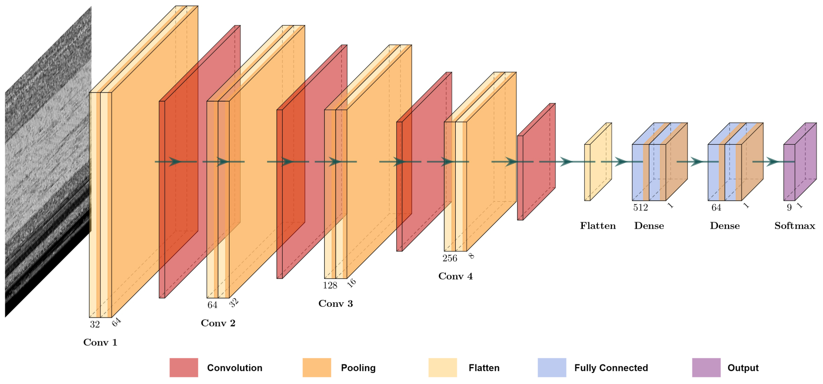


Figure 3.6 – CNN's architecture.

The initial layer is a convolutional layer with 32 filters and a kernel size of (3, 3). This layer is followed by batch normalization and rectified linear unit (ReLU) activation, which helps in normalizing and introducing non-linearity to the learned features, as proposed by Nair and Hinton [37]. Subsequently, a max pooling layer with a pool size of (2, 2) is applied to downsample the spatial dimensions. To prevent overfitting, a dropout rate of 25% is employed [38].

The subsequent layers follow a similar pattern. Another convolutional layer is added, now with 64 filters and a (3, 3) kernel size, followed by batch normalization, ReLU activation, max pooling, and dropout. This process is repeated for two additional convolutional layers, each with an increased number of filters (128 and 256) and the same kernel size of (3, 3).

After the final convolutional layer, a flattening operation is performed to transform the 3D feature maps into a 1D vector. This is followed by two fully connected dense layers. The first dense layer consists of 512 neurons with ReLU activation and a dropout rate of 30%. The second dense layer has 64 neurons with ReLU activation and a dropout rate of 30%.

To enhance regularization and improve training efficiency, batch normalization layers were added after each dense and convolutional layer, and before the activation function.

Batch normalization [39] involves normalizing the outputs of intermediate layers within a network using the mean and variance of the current mini-batch during training. By adjusting and scaling the outputs, batch normalization stabilizes the training process and accelerates convergence.

The output layer is a dense layer with 9 neurons, corresponding to the number of malware classes. It utilizes the softmax activation function to generate a probability distribution over the classes.
Figure 3.7 illustrates the first method which consists mainly of two steps:

0. Converting byte files into images.
0. Training the proposed CNN on byte images.

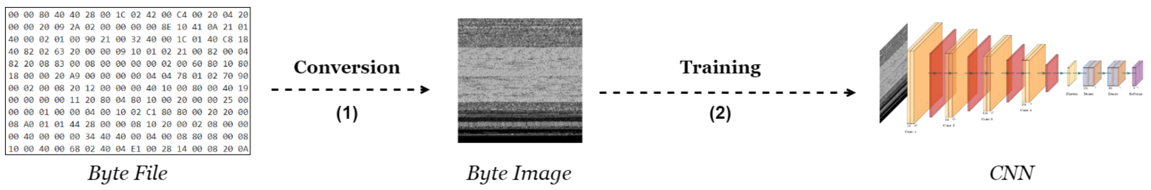


Figure 3.7 – End-to-End illustration of the CNN proposed approach.

3.3 Method 2: A Multi-Modality Approach for Malware Classification

In this section, we present the second proposed method employed for classifying malware families inspired by the work of Gessert et al. [40] in the field of medical imaging, where they have integrated dermoscopic images and patient metadata for skin lesion classification. We adapt their approach to the realm of malware analysis by integrating two types of data: byte images extracted from malware's byte code and assembly-extracted features within a unified deep learning model.

We begin by extracting features from malware executables. Following the extraction process, we perform feature selection to identify the most informative and relevant features. These selected features, along with the images previously extracted from byte files for the implementation of the first method, are combined to train the proposed multimodal.

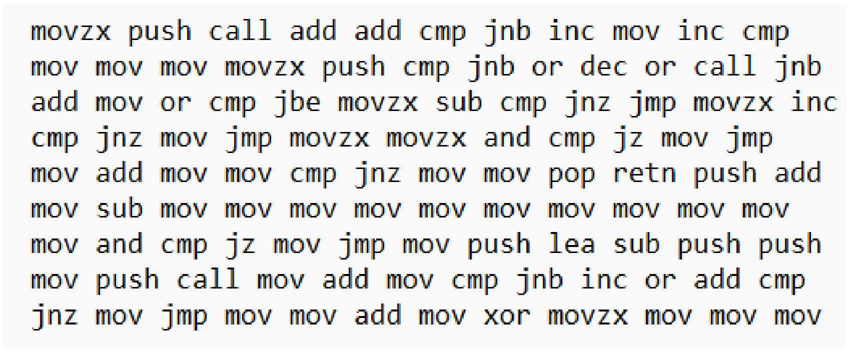
3.3.1 Feature Extraction

We have extracted various features from each malware executable. These features include N-gram counts [41](with N ranging from 1 to 3), byte size, assembly size, assembly-byte ratio, opcodes size, and opcode-assembly ratio. In what follows, we will delve into a detailed exploration of each of the extracted features.

N-gram Counts

Operation codes, also known as opcodes, are fundamental instructions in machine code that define the operations performed by a program. For example, the opcode "ADD" is used to perform addition between two values, while the opcode "JUMP" allows for jumping to a different location within the program's execution flow. Our objective is to extract opcode sequences from an assembly file in order to capture the essential behavior and functionality of the assembly code.

To extract operation codes from each asm file within the data, we employ a regular expression pattern. This pattern allows us to match and capture opcode instructions from the asm files. By scanning the contents of the files, we can identify lines that contain these opcode instructions. Figure 3.8 shows a snippet of operation codes extracted from an assembly file.



```
movzx push call add add cmp jnb inc mov inc cmp
mov mov mov movzx push cmp jnb or dec or call jnb
add mov or cmp jbe movzx sub cmp jnz jmp movzx inc
cmp jnz mov jmp movzx movzx and cmp jz mov jmp
mov add mov mov cmp jnz mov mov pop retn push add
mov sub mov mov mov mov mov mov mov mov mov mov
mov and cmp jz mov jmp mov push lea sub push push
mov push call mov add mov cmp jnb inc or add cmp
jnz mov jmp mov mov add mov xor movzx mov mov mov
```

Figure 3.8 – Opcodes sequence sample.

After obtaining the opcode sequences from the assembly files using the regular expression approach, the subsequent step is to apply the N-Gram language modeling algorithm to extract N-gram counts. This process involves breaking down the sequences into tokens and then counting the occurrences of N-grams within the data.

For the purpose of malware classification, we specifically extract the unigram opcodes, bigrams, and trigrams and count their occurrences. Unigrams represent individual opcodes in the sequence, providing valuable insights into the individual instructions used in the program. Bigrams capture pairs of consecutive opcodes, while trigrams represent triplets of consecutive opcodes. This allows us to identify opcode patterns and dependencies within the code.

For example, let's consider the opcode sequence "movzx push call add add cmp jnb inc mov inc cmp". By analyzing this sequence, we can extract the 1-grams, 2-grams, and 3-grams. The 1-grams would be ("movzx", "push", "call", "add", "cmp", "jnb", "inc"). The 2-grams would be ("movzx push", "push call", "call add", "add add", "add cmp", "cmp jnb", "jnb inc", "inc mov", "mov inc"). Finally, the 3-grams would be ("movzx push call", "push call add", "call add add", "add add cmp", "add cmp jnb", "cmp jnb inc", "jnb inc mov", "inc mov inc", "mov inc cmp").

By counting the occurrences of these N-grams, we can obtain valuable statistical information about the frequency and distribution of opcodes in the program.

Figure 3.9 suggests certain N-grams, such as "mov," "dd," "push," "call," "sub," "add," and "pop," are commonly and prominently present in assembly files compared to other opcodes.

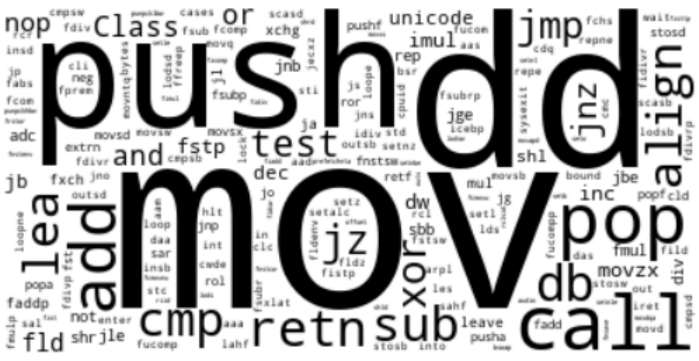


Figure 3.9 – Opcodes cloud.

3.3.2 Size Features

In addition to extracting n-gram counts, we also extract features that describe various size-related characteristics of the data. These features include different dimensions of the content, including the assembly size, byte size, opcode size, the ratio between assembly and byte sizes, and the ratio between opcodes and assembly size described in Table 3.2.

Feature	**Description**
Size of Assembly File	Represents the size (in bytes) of the assembly file corresponding to a sample.
Size of Byte File	Represents the size in bytes of the byte file corresponding to a sample.
Assembly-Byte Size Ratio	Calculates the ratio between the size of the assembly file and the size of the byte file. Provides a measure of the relative size of the assembly code representation compared to the raw byte representation of the sample.
Opcode Sequence Size	Represents the size (in bytes) of the extracted operation codes sequence from each assembly file sample.
Assembly-Opcode Ratio	Calculates the ratio between the size of the opcode sequence file and the size of the assembly file. Provides insight into the relative size of the opcodes compared to the assembly code representation of the sample.

Table 3.2 – Description of extracted size features.

3.3.3 Feature Selection

Motivation for Feature Selection

The number of extracted features exceeded 40,000, primarily due to the utilization of the n-gram method. This resulted in a feature space greater than the number of rows in our dataset, introducing the need for feature selection.

The presence of a large number of features can lead to several challenges:

- **Irrelevant information:** a higher number of features may introduce irrelevant information, hence, increasing the noise in the dataset.
- **Complex models:** Particularly for our study involving deep learning models, a larger number of features translates to a higher number of weights that need to be trained. This increases the complexity of the model, making it more demanding in terms of input data and computational power.

Feature selection [42] eliminates redundant or irrelevant variables, identifies and isolates the variables that contribute most significantly to the performance of the model. By selecting the most informative features, feature selection can enhance accuracy and mitigate the adverse effects of irrelevant or redundant features.

Feature Selection Process

The feature selection process employed in this study follows a progressive approach, aimed at identifying an optimal subset of features that maximizes accuracy while eliminating irrelevant and redundant features. The selection process is as follows:

0. **Random Forest Model Training:** The first step involves training a random forest model using the dataset. This ensemble learning technique effectively captures the underlying relationships between the features and the target variable. By leveraging the power of multiple decision trees, the random forest model provides reliable estimates of feature importances [21].
0. **Feature Importance Ranking:** From the trained random forest model, feature importances are obtained, allowing for the ranking of each feature based on their importance scores.
0. **Progressive Feature Selection:**
   — We train a random forest model iteratively using cross-validation, starting with the top ten most important features. We evaluate the model's performance and then gradually add more features to the model, increasing the subset size to twenty, thirty, and so on...
   — In each iteration, we retrain the model and assess its accuracy. The iterative process continues until we have considered and evaluated all features.
   — Finally, we determine the optimal subset of features by choosing the one that achieves the highest accuracy throughout the iterations.

In the Figure 3.10, it is observed that as the number of features increases too much, the accuracy starts to decrease. This suggests that having too many features doesn't always improve performance and can even lead to lower accuracy. Based on this observation, the subset of the top 350 features is chosen since it achieved the highest accuracy score 0.9883.

Figure 3.10 – Feature selection- accuracy by number of best features.

Figure 3.11 illustrates the top 20 features based on Gini impurity comprising the assembly, byte, and opcode sequences sizes, as well as various trigram patterns such as "call add pop" and "xor retn align." Additionally, it includes unigrams and bigrams like "sub lea" and "dd."

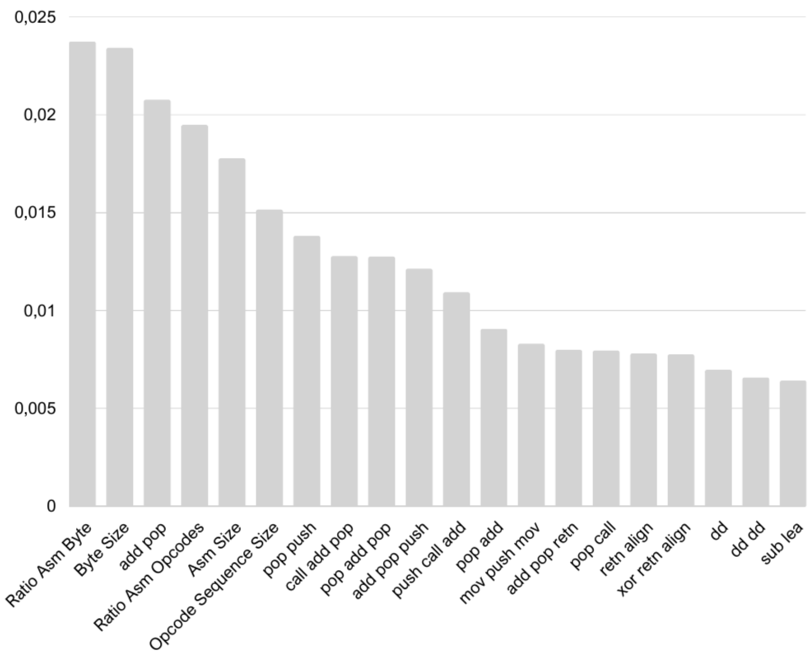


Figure 3.11 – Top 20 predictive features.

Standardization of Selected Features

A standardization process was applied to the selected features. Standardization involves transforming the data in such a way that each feature has a mean of 0 and a standard deviation of 1. This is achieved by subtracting the mean of each feature from each element in the feature and then dividing it by the standard deviation. The formula for standardization can be expressed as:

$$\bar{x} = \frac{x - \mu}{\sigma} \tag{3.1}$$

where x represents the original value of a feature, μ and σ are the mean and the standard deviation of that feature, respectively. While $\bar{x}$ represents the standardized value.

Standardizing the tabular features is important because it ensures that the features are on a similar scale, allowing for fair comparison and preventing any single feature from dominating the learning process. This process helps to address issues such as the bias towards features with larger numerical values and the impact on gradient descent optimization.

3.3.4 Architecture of the Proposed Model

The second approach proposed in this thesis, is based on multimodal neural networks trained on two data modalities images and tabular features. The image input is passed through a convolutional block. Concurrently, the tabular input is processed through fully connected layers, forming Feed-Forward Network that extracts meaningful representations from the tabular data.

After processing the image and tabular inputs separately, the outputs from both branches are concatenated. The concatenated features are then fed into additional fully connected layers that determine the class of the malware. A simplified model structure visualization is represented in Figure 3.12.

Figure 3.12 – Illustration of the multimodal approach.

The architecture of the proposed multimodal consists of two parallel networks, a Convolutional Network and Feed-Forward, each serving a distinct purpose. The first network specializes in extracting high-level features from the input byte images, while the second one extracts patterns and features from the assembly features. Figure 3.13 presents the detailed architecture of the multimodal.

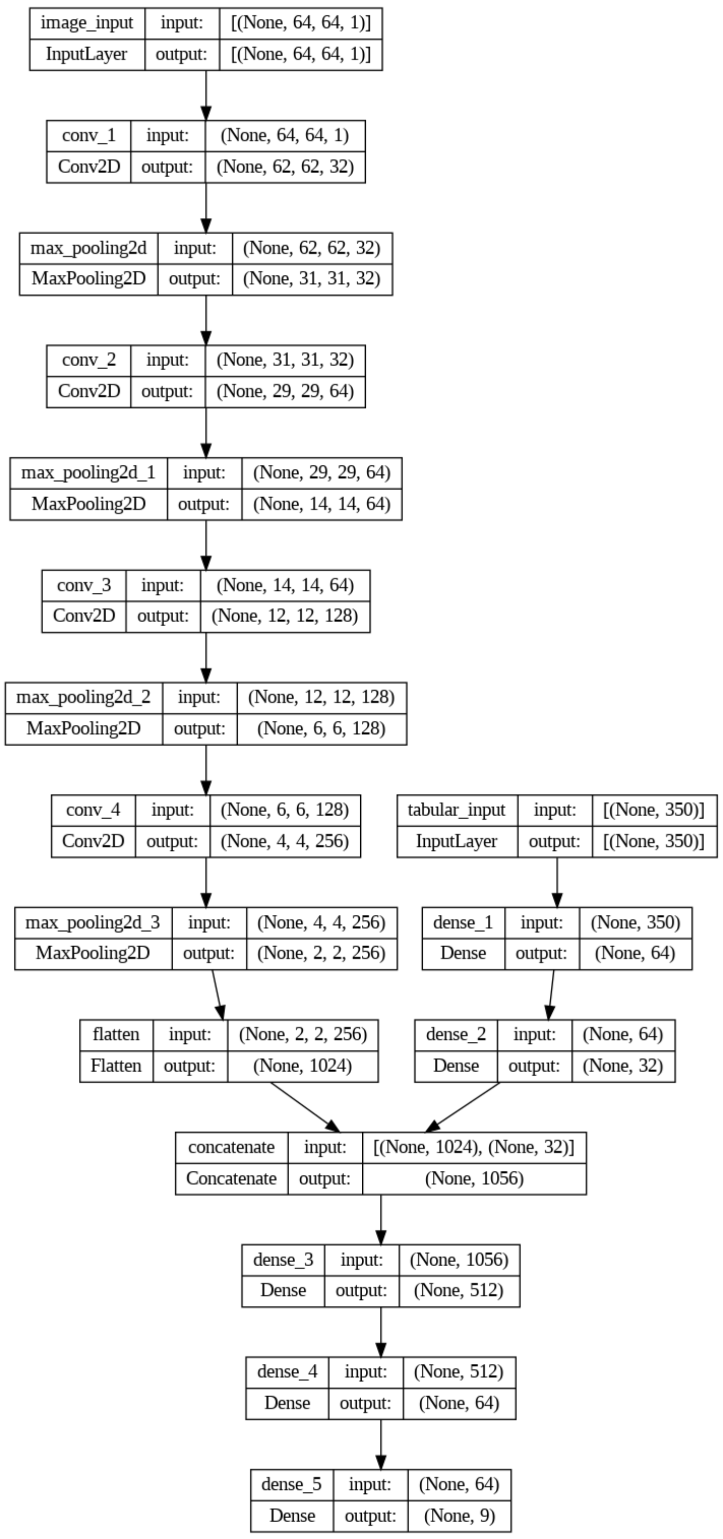

image_input | input: | [(None, 64, 64, 1)]
InputLayer | output: | [(None, 64, 64, 1)]
conv_1 | input: | (None, 64, 64, 1)
Conv2D | output: | (None, 62, 62, 32)
max_pooling2d | input: | (None, 62, 62, 32)
MaxPooling2D | output: | (None, 31, 31, 32)
conv_2 | input: | (None, 31, 31, 32)
Conv2D | output: | (None, 29, 29, 64)
max_pooling2d_1 | input: | (None, 29, 29, 64)
MaxPooling2D | output: | (None, 14, 14, 64)
conv_3 | input: | (None, 14, 14, 64)
Conv2D | output: | (None, 12, 12, 128)
max_pooling2d_2 | input: | (None, 12, 12, 128)
MaxPooling2D | output: | (None, 6, 6, 128)
conv_4 | input: | (None, 6, 6, 128)
Conv2D | output: | (None, 4, 4, 256)
tabular_input | input: | [(None, 350)]
InputLayer | output: | [(None, 350)]
max_pooling2d_3 | input: | (None, 4, 4, 256)
MaxPooling2D | output: | (None, 2, 2, 256)
dense_1 | input: | (None, 350)
Dense | output: | (None, 64)
flatten | input: | (None, 2, 2, 256)
Flatten | output: | (None, 1024)
dense_2 | input: | (None, 64)
Dense | output: | (None, 32)
concatenate | input: | [(None, 1024), (None, 32)]
Concatenate | output: | (None, 1056)
dense_3 | input: | (None, 1056)
Dense | output: | (None, 512)
dense_4 | input: | (None, 512)
Dense | output: | (None, 64)
dense_5 | input: | (None, 64)
Dense | output: | (None, 9)

Figure 3.13 – Architecture of the employed multimodal model.

The CNN network shares the same architecture and weights as the Convolutional part of the pre-optimized CNN described in Section 3.3.2. However, in this network, the weights are frozen, meaning they are not updated during training.

The Feed-Forward network receives as input the features extracted from assembly executables. Its architecture consists of two dense layers. The first layer has 64 units and applies the ReLU activation function to the input. The output of this layer is then fed into the second dense layer, which has 32 units and also uses the ReLU activation function. These layers help the network learn and extract complex patterns and features from the tabular data.

The extracted features from both networks are combined and passed through a dense layer with 512 units, followed by batch normalization, ReLU activation, and dropout regularization with a rate of 0.5. Then, another dense layer with 64 units is employed, along with batch normalization, ReLU activation, and dropout regularization with a rate of 0.3. Finally, a dense layer with 9 units and softmax activation is used for the classification output.

The overall hybrid network architecture encompasses 979,497 trainable parameters, which are optimized during the model training process.
The steps for the implementation of the multimodal approach are presented in Figure 3.14 and can be summarized as follows:

0. Conversion of byte malware executables into images.
0. Extraction of pertinent features from assembly files.
0. Selection of best features.
0. Training the proposed multimodal on byte images and the extracted tabular data.

Figure 3.14 – End-to-End illustration of the multimodal proposed approach.

3.4 Conclusion

To conclude, this chapter presented two distinct approaches for malware classification: one based on CNNs and the other utilizing a multimodal network. In the next chapter, we will delve into the results obtained from training these proposed models and engage in a comprehensive discussion of their effectiveness and implications.

Chapter 4

Experiments, and Discussion

In this chapter, we conduct a thorough performance evaluation of the appraoches proposed in the previous chapter, analyzing and providing a comparative analysis.

We begin by describing the experimental environment in Section 4.1, outlining hardware and software specifications used for code implementation. In Section 4.2, we perform data exploratory analysis on the extracted features. The performance of both models is presented in Section 4.3, followed by a comprehensive comparative analysis in Section 4.4. Furthermore, Section 4.5 provides valuable insights into the models' real-world performance and generalization abilities by discussing the results of evaluation on the test data. Lastly, in Section 4.6, we estimate the confidence intervals at which the performance of the evaluation metrics for the proposed models measures.

4.1 Experimental Environment

The implementation of the proposed models and all associated code including feature extraction, feature selection, and byte image conversion, have been executed on an Acer local computer with the following specifications: an 11th Gen Intel(R) Core(TM) i7-11370H processor running at 3.30GHz, 16.0 GB of RAM, an Intel Iris Xe Graphics GPU and Windows 11 operating system. The implementation has been carried out using Python 3.9.13 and Tensorflow framework version 2.11.0, utilizing the Jupyter Notebook development environment.

4.2 Exploratory Data Analysis

4.2.1 Data Destribution

Microsoft dataset [30] provides a valuable resource for the development and evaluation of effective techniques for the classification of malware.

In Figure 4.1, the number of samples available in each of the nine malware families is presented, providing a visual representation of the distribution of the dataset.

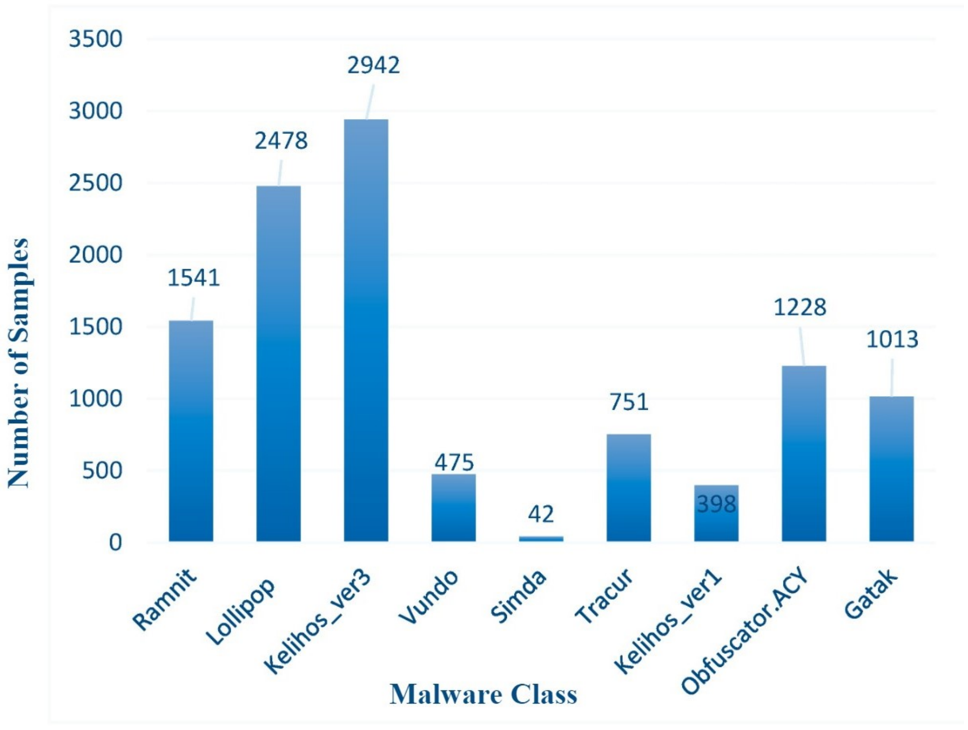


Figure 4.1 – Distribution of the dataset.

The histogram of data distribution reveals a significant data imbalance across the nine malware classes, with only 42 samples belonging to the Simda class to the much larger 2942 samples of the kelihos-ver3 class.

4.2.2 Boxplots Analysis

In this section we examine the boxplots representing the size features extracted from malware executables. The boxplots provide insights into the distribution of the extracted size features across the nine malware classes.

Size of Assembly File

Figure 4.2 presents a visual representation of the distribution of assembly sizes for the nine different malware families. Class lollipop exhibits higher median, suggesting larger average assembly sizes, while classes Kelihos-ver3, Kelihos-ver1, and Obfuscator.ACY have low medians relatively, indicating smaller average assembly sizes. Additionally, the spread of assembly sizes differs among the classes. For instance, class lollipop have a longer box, indicating a broader range of assembly sizes and higher variability compared to other classes that showed relatively tight clustering of assembly sizes. Furthermore, outliers are observed in all the classes except for Vundo and Simda, signifying the presence of larger and potentially exceptional assembly sizes within those classes.

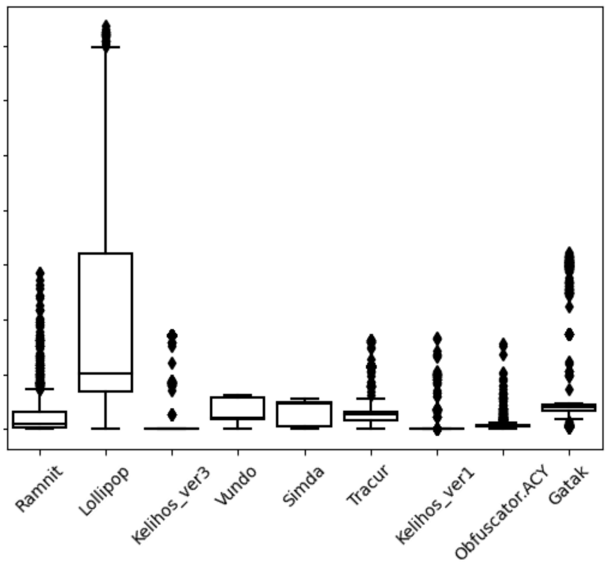


Figure 4.2 – Boxplot analysis: Assembly size.

Size of Byte File

Figure 4.3 illustrates distinct variations in byte size among the different classes. The boxplots reveal noticeable differences between classes in terms of their byte sizes. Classes Kelihos-ver3 and Kelihos-ver1 exhibit narrow boxes, indicating a relatively concentrated range of byte sizes. Furthermore, these classes demonstrate higher median byte sizes compared to the other classes.

On the other hand, classes Ramnit, Vundo, Tracur, and Obfuscator.ACY display wider boxes, suggesting a greater variability in byte sizes within these classes. The ranges of byte sizes in classes Lollipop, Simda, and Gatak are particularly notable as they exhibit the largest box spans, signifying a wide dispersion of byte sizes within these classes.

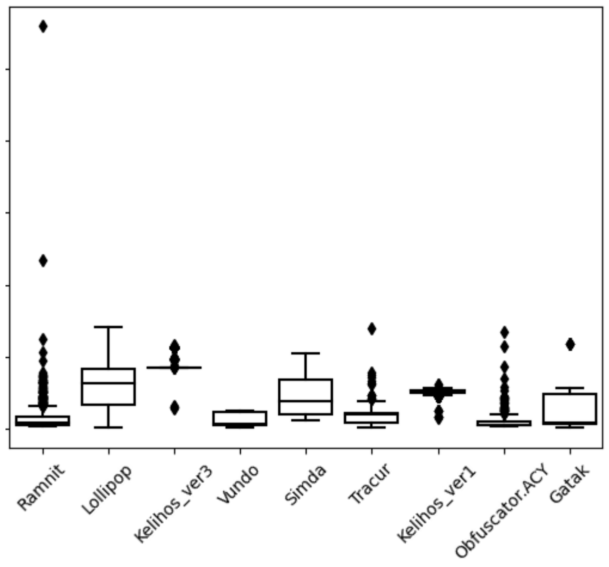


Figure 4.3 – Boxplot analysis: Byte size.

Assembly-Byte Size Ratio

Among the different classes, Kelihos-ver3 and Kelihos-ver1 exhibit distinct behavior in terms of the assembly/byte ratio as observed in Figure 4.4. These classes display significantly smaller assembly sizes relative to their byte sizes, approaching zero. This characteristic sets Kelihos-ver3 and Kelihos-ver1 apart from the other classes. Consequently, the assembly/byte ratio serves as a reliable metric to easily distinguish Kelihos-ver3 and Kelihos-ver1 based on this unique characteristic.

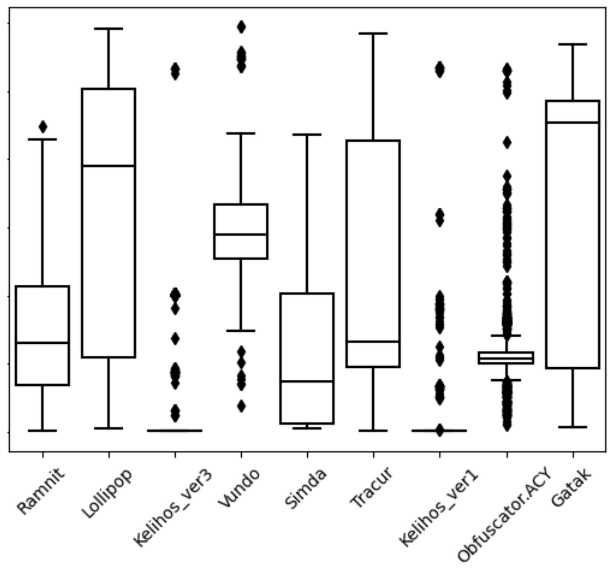


Figure 4.4 – Boxplot analysis: Assembly-Byte size ratio.

Opcode Sequence Size

In the box plot analysis of the opcodes size presented in Figure 4.5, it can be observed that the Lollipop class exhibits a wide range of sizes and the highest median size of opcode sequences when compared to the other classes. In contrast, the Kelihos-ver1, Kelihos-ver3, and Obfuscator.ACY classes contain smaller opcode sequences compared to the rest. The remaining classes, they demonstrate an intermediate range of opcode sequence sizes.

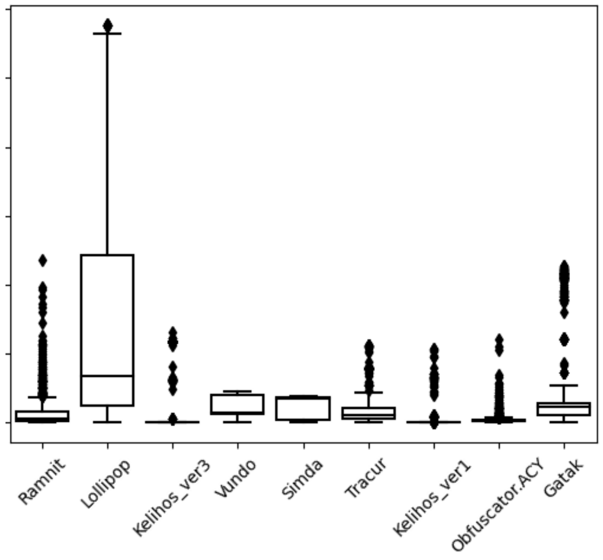


Figure 4.5 – Boxplot analysis: Opcodes size.

Assembly-Opcode Ratio

Based on the boxplot of opcode-assembly size ratio shown in Figure 4.6, it can be observed that the classes Lollipop and Gatak demonstrate a broad range of ratio values, indicating a significant spread of values for the opcodes assembly size ratio.

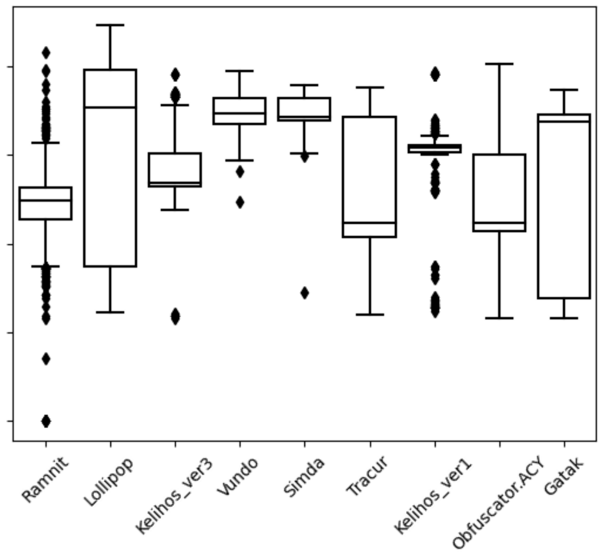


Figure 4.6 – Boxplot analysis: Opcodes-assembly size ratio.

The classes Tracur and Obfuscated Malwares also have large boxes, indicating a significant range in the opcodes assembly size ratio, but relatively smaller than the previous two classes. In contrast, the remaining classes exhibit narrower boxes, indicating a shorter range in the opcodes-assembly size ratio.

The differing sizes of the boxes, and the range of values demonstrate that the malware classes do not behave in the same manner. Some classes exhibit a wide range and greater variability, while others display a narrower range and more consistent behavior in terms of the assembly-size ratio.

The boxplots analysis demonstrates that the malware families exhibit different tendencies and patterns, indicating that the extracted size features are relevant and allow to diffrentiate between the classes.

The radar chart in Figure 4.7, represents the 5 extracted size features in a two-dimensional space. we can observe that each of the nine families displays a different positioning of data points across the extracted size features. The distinct patterns and variations observed across the classes suggest that these features are effective in differentiating between malware families.

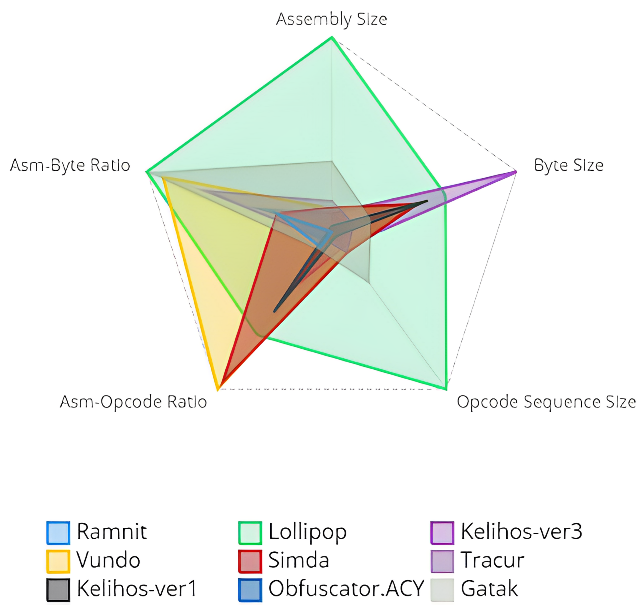


Figure 4.7 – Radar chart of size features.

4.2.3 Feature Space: 2D Visualization

The Figure 4.8 illustrates a t-SNE plot of the selected features [43]. t-SNE, which stands for t-Distributed Stochastic Neighbor Embedding, is a dimensionality reduction technique commonly used for visualizing high-dimensional data in a lower-dimensional space. It aims to capture the underlying structure of the data preserving the relationships between samples.

In this t-SNE plot, each point represents a malware executable sample, its position in the two-dimensional space is determined by the t-SNE algorithm, while its coloring is determined by the class it belongs to.

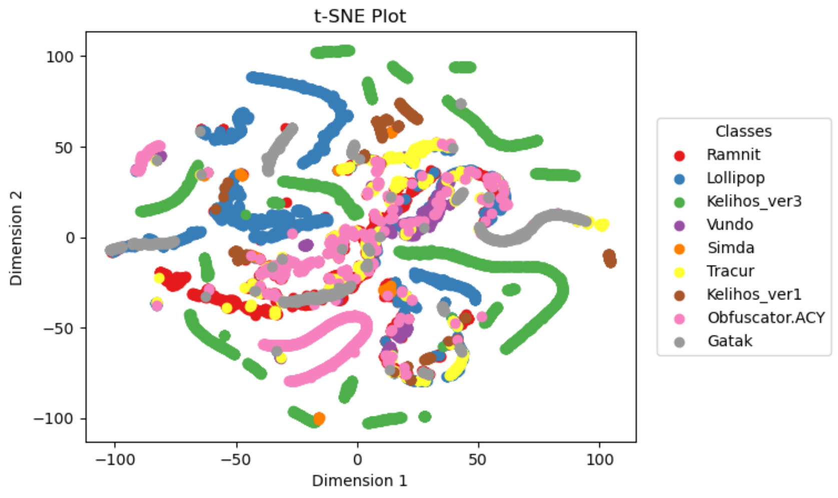


Figure 4.8 – t-SNE of the selected features.

It can be observed from the t-SNE plot that the classes exhibit a certain level of separation. Some classes, such as "Kelihos-ver1," display distinct clusters, indicating a high degree of separability in the reduced feature space. However, there is some overlap and confusion observed, particularly with the "Obfuscator.ACY" class, which appears to have similarities with other classes.

4.3 Performance Evaluation

In this study, the dataset was initially shuffled to remove any inherent ordering or bias. Subsequently, the shuffled dataset has been divided into training and validation sets, with approximately 80% of the data allocated for training and 20% for the purpose of validation and assessment of the model's generalization ability.

4.3.1 Method 1: CNN-based Malware Classification

In this section, we present the performance of the Convolutional Neural Network (CNN) model for malware classification.

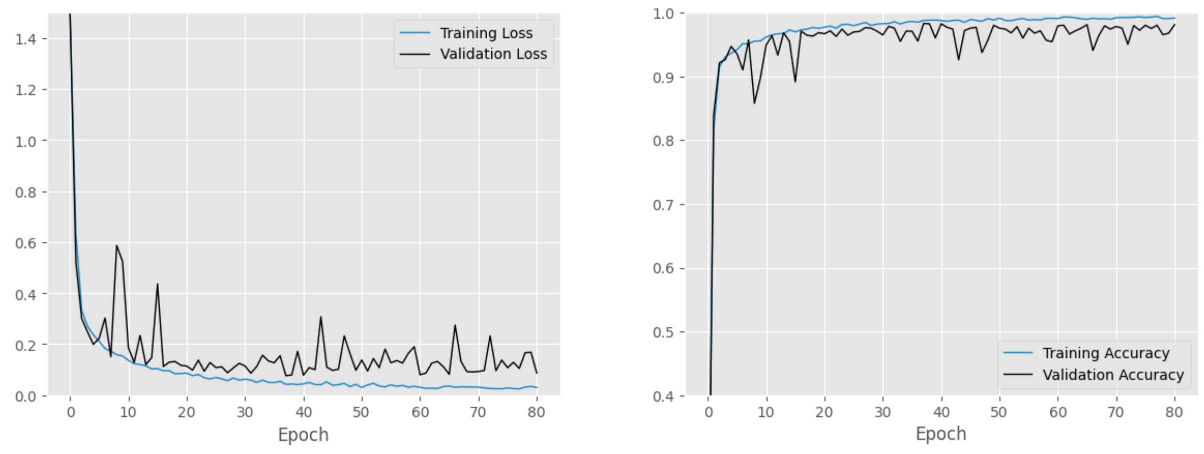
Training Process

The Convolutional Neural Network model described in section 3.2.2 is trained using the Adam algorithm for the optimization of the network's weights [44] with a categorical cross-entropy loss function. The learning rate of the optimizer is set to 0.001 (default). The training is performed over 80 epochs with a batch size of 32. The validation data is used to evaluate the model's performance during training, ensuring generalization to unseen data and the weights of the epoch with the least validation loss are saved to be used when training the multimodal.

The selection of the optimal training epoch is based on the criterion of achieving the lowest validation loss. In our study, the convolutional neural network (CNN) attained a validation loss of 0.0764, accompanied by an accuracy of 98.29%. The corresponding metrics on the training data were a loss of 0.0233 and an accuracy of 99.33%.

Loss and Accuracy

During the training process, we monitored the training and validation loss to assess the convergence and generalization capability of the CNN model. Figure 4.9 presents the loss and accuracy curves for both the training and validation datasets. Notably, the model consistently reduces its training loss over the course of iterations along with the consistent increase of the accuracy, indicating a gradual improvement in its ability to minimize errors and effectively learn from the training data. Concurrently, the validation loss also demonstrates a downward trend, signifying the model's aptitude for generalization to previously unseen data.



(a) Categorical Cross-Entropy (b) Accuracy

Figure 4.9 – Performance evaluation- method 1.

Moreover, the absence of a significant divergence between the training and validation loss curves demonstrates that our model did not experience overfitting. Overfitting is a critical concern in machine learning, wherein a model becomes excessively tailored to the training data, leading to poor performance on new, unseen data. The parallel behavior of the training and validation loss curves in Figure 4.9 signifies a well-balanced model that captures meaningful patterns in the training data while maintaining robust generalization capabilities.

Confusion Matrix

To gain further insights into the CNN model's classification performance across the different malware families, we present the confusion matrix of the validation data in Figure 4.10. The rows represent the true labels, while the columns represent the predicted labels.

From the confusion matrix, we can observe that the CNN model achieved high accuracy for the majority of the families. The model accurately classified most samples of the 'Kelihos-ver1' and 'Gatak' families. However, the 'Simda' family has shown a relatively lower accuracy 40% compared to other families due to the limited number of samples available for training and validation.

In addition, the class 'Obfuscator.ACY' was occasionally confused with other classes, this highlights the complexity of accurately distinguishing the unique characteristics of obfuscation-based malware within the classification process likely due to Obfuscation methods employed to the malicious code in order to conceal its true intent and functionality.

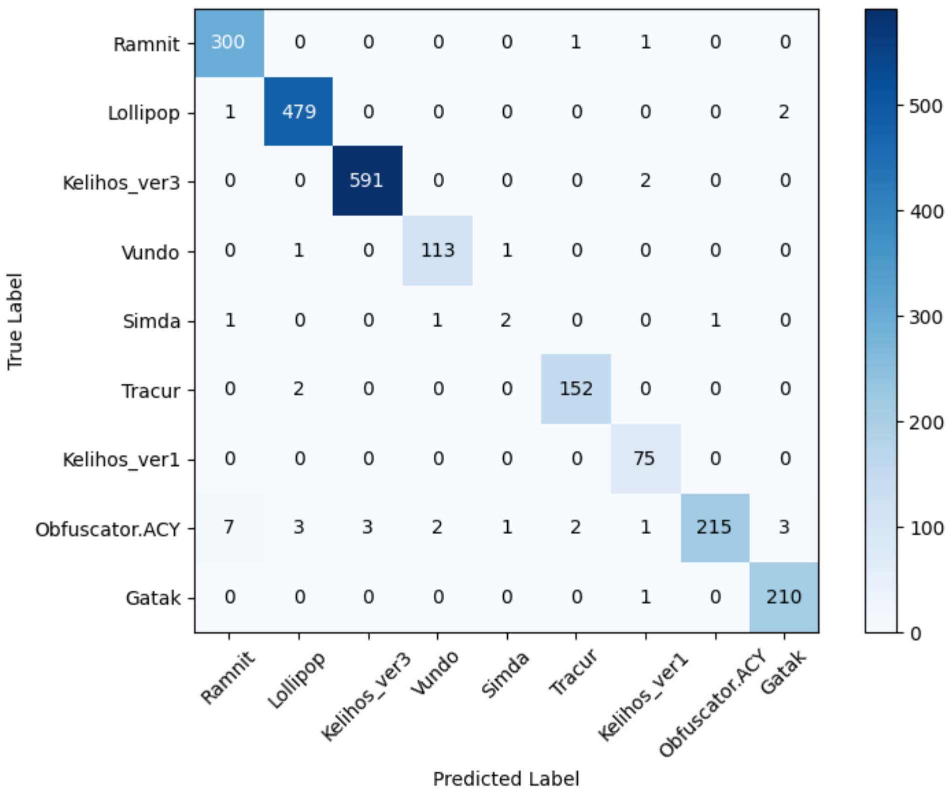


Figure 4.10 – Confusion matrix- method 1.

4.3.2 Method 2: Multimodal-based Malware Classification

In this section, we evaluate the performance of the Multimodal approach for malware classification.

Training Process

In Section 3.2.3, we introduced the Multimodal Deep Network, which combines data from different sources tabular and visual representations, resulting in improved performance compared to the CNN model trained solely on images.

The training process is set to run for 80 epochs, each epoch the model adjusts its parameters to minimize the categorical cross-entropy loss. The Adam optimizer updates the model's weights based on the computed gradients. A checkpoint callback is included to save the weights of the best-performing model based on the validation loss.

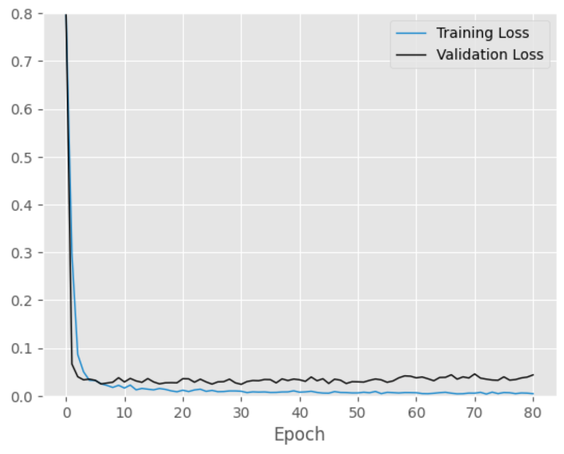
The model's performance was evaluated based on the criterion of achieving the lowest validation loss, which would enhance the accuracy in classifying malware family.

Loss and Accuracy

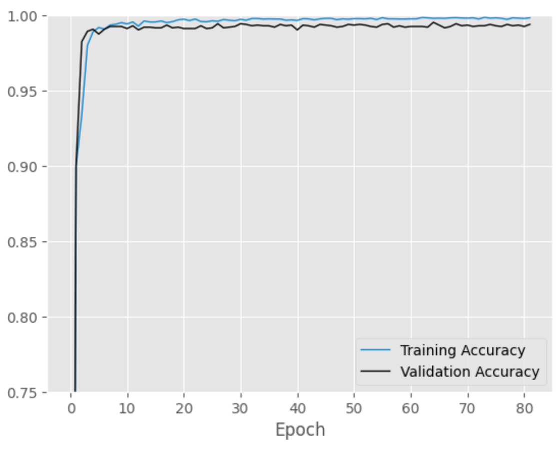
Similar to the CNN model, we monitored the training and validation loss of the Multimodal approach during the training process. Figure 4.11 tracks the performance of the model over 80 epochs.

On the validation dataset, the Multimodal achieved a categorical cross-entropy loss of 0.0238, accompanied by an accuracy of 99.40%. While, on the training data, the model claimed an accuracy of 99.90% with a corresponding loss value of 0.0028.

The curves of loss and accuracy revealed no hints of overfitting, as both the training and validation metrics decreased without a significant divergence between them.



(a) Categorical Cross-Entropy



(b) Accuracy

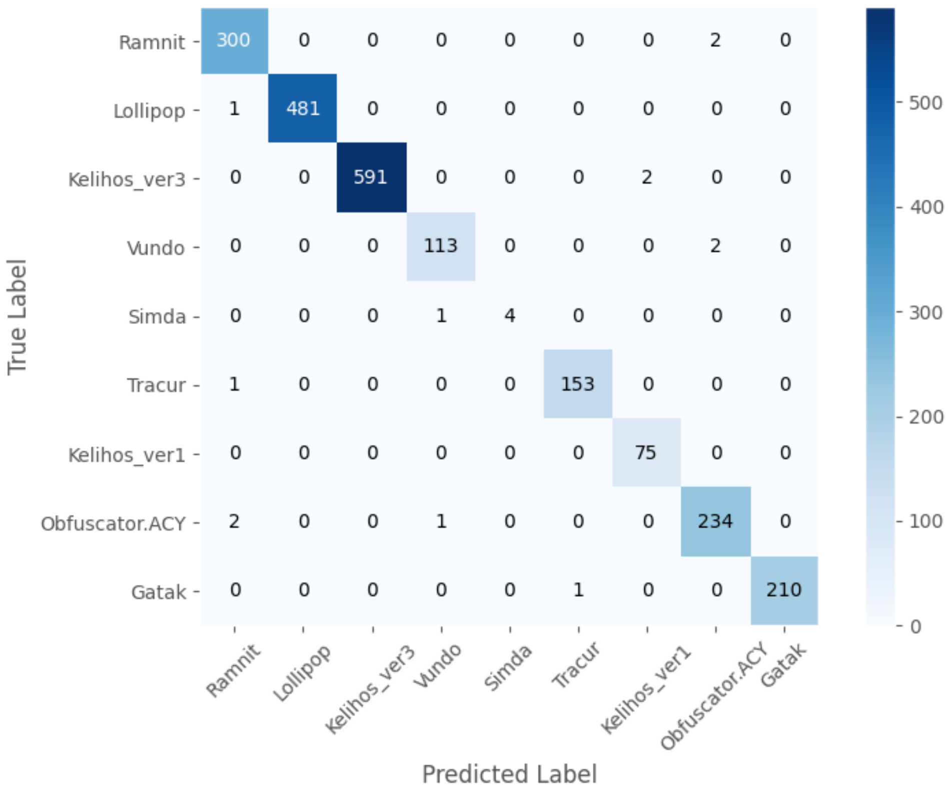
Figure 4.11 – Performance evaluation- method 2.

Notice that the Multimodal converged quickly in the early stages of training from the first epoch, that is due to the integration of the pre-trained CNN within the Multimodal architecture. By freezing the weights of the previously trained CNN and utilizing it as part of the Multimodal model, we leveraged the valuable knowledge and representations learned from the CNN's prior training. This initialization provided the Multimodal model with a head start, enabling it to quickly adapt and fine-tune its parameters using the combined information from the images and tabular features.

Confusion Matrix

To assess the multimodel's classification performance, we present the confusion matrix in Figure 4.12. The rows represent the true labels, while the columns represent the predicted labels.

Analyzing the confusion matrix of the validation data, we observe that the multimodel approach demonstrated further improvements compared to CNN's confusion matrix across all the classes.



True \ Predicted	Ramnit	Lollipop	Kelihos_ver3	Vundo	Simda	Tracur	Kelihos_ver1	Obfuscator.ACY	Gatak
Ramnit	300	0	0	0	0	0	0	2	0
Lollipop	1	481	0	0	0	0	0	0	0
Kelihos_ver3	0	0	591	0	0	0	2	0	0
Vundo	0	0	0	113	0	0	0	2	0
Simda	0	0	0	1	4	0	0	0	0
Tracur	1	0	0	0	0	153	0	0	0
Kelihos_ver1	0	0	0	0	0	0	75	0	0
Obfuscator.ACY	2	0	0	1	0	0	0	234	0
Gatak	0	0	0	0	0	1	0	0	210

Figure 4.12 – Confusion matrix- method 2.

Notably, the multimodal approach has achieved a significant reduction in confusion between the 'Obfuscator.ACY' class and other classes, indicating an enhanced capability to recognize the distinct features of obfuscation-based malware.

4.4 Comparative Analysis

In this section, we conduct a performance comparison between the Convolutional Neural Network (CNN) model and the Multimodal approach. Table 4.1 provides a comprehensive side-by-side analysis of key metrics, including accuracy, precision, recall, and F1 score.

Method	Dataset	Accuracy	F1 Score	Precision	Recall
CNN	Train	99.33	99.34	99.37	99.33
	Validation	98.29	98.27	98.30	98.29
Multimodal	Train	99.90	99.90	99.90	99.90
	Validation	99.40	99.41	99.41	99.40

Table 4.1 – Performance comparison of the proposed methods.

As shown in the comparison table, the Multimodal approach outperforms the CNN model. The integration of the tabular modality significantly enhances the validation performance of malware family prediction by more than 1%.

4.5 Evaluation on Test Data

When evaluating the performance of the proposed methods, particular attention has been given to assessing their ability to generalize on unseen data. To accomplish this, the test dataset, comprising 10,873 files, has been utilized. It is important to note that the test dataset provided by Kaggle does not include the output class. Consequently, the scores are obtained by submitting the predictions to the Kaggle platform and the performance evaluation relied on log-loss metric.

Table 4.2 presents a comparison of the test losses alongside the corresponding validation and training losses of the proposed models.

Dataset	Train	Validation	Test
CNN	0.0233	0.0764	0.1150
Multimodal	0.0028	0.0238	0.0579

Table 4.2 – Log-loss comparison of train, validation, and test sets.

Discussion:

The evaluation of the models' performance on the test, validation, and train sets provides insights into their generalization capabilities. The size of the test set, which consists of 10,873 malware executables, is larger compared to the train and validation sets, which combined together are composed of 10,868 executables. This allows for a comprehensive assessment of the models' ability to generalize to new unseen data.

The CNN model demonstrates reasonably good generalization, with comparable log-loss values on the validation (0.0764) and test sets (0.1150). However, the slight difference in log-loss suggests potential for further improvement to enhance its generalization.

On the other hand, the multimodal achieved significantly lower log-loss values on both the validation (0.0238) and test sets (0.0579). The small difference between the log-loss values indicates that the multimodal model effectively generalizes to new unseen data.

Comparing the performance of the CNN and multimodal models sheds light on the advantages of incorporating multiple modalities for malware classification. The CNN model, as a single-modality approach, demonstrates reasonable performance. However, its log-loss values on the validation and test sets are higher compared to the multimodal model. This performance gap emphasizes the benefits of utilizing additional modalities, as the multimodal outperforms the CNN model across all datasets.

4.6 Performance Confidence Estimation

When training deep learning models, it is common to observe variations in performance across different runs. This can be attributed to the random initialization of the model's parameters, such as weights and biases. Each training run starts from a different set of initial values, leading to different trajectories of optimization and potentially different solutions.

To address the variability and provide a more comprehensive understanding of the model's performance, we estimate the confidence intervals of evaluation metrics within which we expect the true performance of the model to fall [45, 46].

4.6.1 Validation Set

We will conduct 8 training experiments. After each experiment, we will record the validation scores of the evaluation metrics for both methods CNN and multimodal. By analyzing these scores, we can calculate the mean of the 8 experiments and the standard deviation, which quantifies the variation or spread around the mean. We then estimate the confidence interval using the following formula:

$$\text{Confidence Interval} = \text{Experiment Mean} \pm (\text{Critical ValueStandard Error}) \tag{4.1}$$

where the Critical Value represents the value obtained from the appropriate distribution (t-distribution) corresponding to the desired level of confidence. For a 95% confidence level and 8 experiments, the critical value is around 2.365, which estimates the range that captures approximately 95% of the population values within the confidence interval.

The Standard Error is calculated by dividing the standard deviation by the square root of the number of experiments .
The results of the estimation of confidence intervals for the CNN and multimodal models are shown in Table 4.3 and Table 4.4, respectively.

Method1: CNN-based Approach

Experiment	**Accuracy**	**Loss**	**F1-Score**
1	0.9829	0.0764	0.9827
2	0.9825	0.0733	0.9825
3	0.9811	0.0736	0.9809
4	0.9848	0.0734	0.9847
5	0.9806	0.0818	0.9806
6	0.9853	0.0697	0.9852
7	0.9834	0.0698	0.9831
8	0.9765	0.0847	0.9765
Range Estimate	0.98213 ± 0.00218	0.07534 ± 0.00423	0.98199 ± 0.00222

Table 4.3 – CNN confidence intervals.

Method2: Mulitmodal-based Approach

Experiment	Accuracy	Loss	F1-Score
1	0.9940	0.0238	0.9941
2	0.9926	0.0295	0.9926
3	0.9940	0.0282	0.9940
4	0.9931	0.0298	0.9930
5	0.9917	0.0288	0.9917
6	0.9917	0.0252	0.9916
7	0.9913	0.0285	0.9913
8	0.9885	0.0299	0.9884
Range Estimate	0.99211 ± 0.00139	0.02796 ± 0.00176	0.99208 ± 0.00143

Table 4.4 – Multimodal confidence intervals.

The estimated confidence intervals provide a measure of uncertainty for the performance metrics: accuracy, log-loss, and f1-score on the validation set. While analyzing the results of both CNN and multimodal proposed methods, slight variations have been observed, the narrow range within the estimated confidence intervals suggests that the models' performance remains closely aligned. This indicates the stability of both models in their malware classification task.

4.6.2 Test Set

Similarly to the previous section, where we estimated the confidence intervals for the evaluation metrics of both the CNN and multimodal models on the validation set, we will now continue by estimating the confidence intervals for the loss on the test set for both models. This allow us to observe the range of performance showed by the models on the test data, providing insights into their ability to generalize to unseen data.

It can be observed from Table 4.5 that the CNN model achieved a performance of 0.10550 with a confidence interval of ±0.00615 on the test set. On the other hand, the multimodal model showed a narrower range of 0.04310 ± 0.006.

While the loss range for both models on the test set is indeed larger than on the validation set, it remains relatively narrow. This indicates that both models consistently perform well and maintain a level of stability.

Experiment	**CNN**	**Multimodal**
1	0.11499	0.05792
2	0.09457	0.04199
3	0.09836	0.05066
4	0.10276	0.03873
5	0.11351	0.04486
6	0.09837	0.03862
7	0.10519	0.03538
8	0.11266	0.03755
Range Estimate	0.10550 ± 0.00615	0.04321 ± 0.00600

Table 4.5 – Test confidence intervals.

4.7 Conclusion

In this chapter we conduct a comprehensive analysis of the proposed approaches, evaluating their performance and making comparisons. The performance of each approach has been thoroughly examined, and their ability to generalize has been assessed. As part of our analysis, we have estimated confidence intervals for the models, providing a measure of uncertainty and a range of potential values for their performance. Moving forward, the next chapter will summarize the main findings of this study and provide suggestions for future work.

Chapter 5

Conclusion and Future Work

5.1 Summary of Findings

In this master's study, we have focused on the problem of malware classification and have employed deep learning techniques to address this challenge. Utilizing a large publicly available dataset, we have conducted a comprehensive evaluation of our proposed approaches, which incorporated both byte and assembly executables.

By extracting information from both byte images and assembly features, we have prepared the data to train two models: a multimodal and a baseline CNN. The multimodal has incorporated both data modalities to enhance the accuracy and effectiveness of malware family classification, while the CNN was trained solely on byte images.

The results revealed that the multimodal model achieved in average a validation accuracy of 99.21% with a loss of 0.0278, outperforming the single-modality CNN with an average classification accuracy of 98.21% and a loss of 0.07534. This improvement in accuracy demonstrates the value of incorporating assembly-extracted features alongside byte images, allowing the model to capture a broader range of malware characteristics and better discriminate between different malware classes.

Additionally, we evaluated the performance of the proposed models on new, unseen data approximately five times the size of the validation set. The multimodal consistently demonstrated good performance on this new dataset, with an average loss of 0.0278 on the validation set and 0.0432 on the test set, respectively. The slight difference between the validation and test set losses indicates that the model generalizes well and does not appear to be overfitting to the training data.

These findings highlight the advantages of leveraging multimodal data sources, specifically byte images and assembly-extracted features, for malware classification. By combining these modalities, we provided the model with a comprehensive understanding of malware behavior, inspired by the multi-sensory nature of human perception. Similar to how humans rely on multiple senses such as vision, hearing, and touch to navigate and comprehend the world, the proposed multimodal approach emulates this concept by integrating different types of information. Byte images represent the visual aspect, capturing patterns and structures within the malware executables, while assembly-extracted features provide a deeper insight into the functional and behavioral aspects of the malware.

5.2 Future work

To enhance the conducted work in this thesis and broaden its potential impact, there are several potential directions for future work. These include exploring data augmentation techniques, expanding the scope of malware classification to encompass a wider range of malware families, and integrating dynamic analysis techniques into the existing classification framework.

Data Augmentation

The dataset utilized for the task of malware classification is imbalanced, certain malware families are underrepresented compared to others. This can pose a challenge for the classification process as the model may struggle to accurately identify and classify the minority classes. To address this issue, a potential avenue for improvement is to implement data augmentation techniques. Data augmentation involves generating additional samples by applying various transformations to the existing data. By increasing the representation of the underrepresented malware families, data augmentation can help create a more balanced dataset. This, in turn, enables the classification model to learn from a wider range of examples and improves its accuracy and reliability in identifying different malware classes.

Expanding the Malware Classification to a Larger Number Families

One potential avenue for future research is to expand the behavior classification to include a larger number of malware families. While the current study focused on nine malware families, there are numerous other families that exhibit distinct behaviors and characteristics. By expanding the classification to include a larger number of malware families, we can enhance the effectiveness and applicability of the proposed approach. However, achieving this expansion would require substantial efforts in terms of data collection, annotation, and computational resources.

Integrating Dynamic Analysis Techniques

The work conducted in this thesis falls within the domain of static analysis, which involves extracting features from malware executables without executing them. Static analysis provides insights into the structural and behavioral characteristics of malware by examining the code, metadata, and other attributes present in the executable file. It allows for the identification of patterns and signatures that can be used to classify and categorize malware based on their inherent properties.

Another avenue for future work involves incorporating dynamic analysis techniques into the classification framework. Dynamic analysis, in contrast to static analysis, involves executing malware samples in a controlled environment, such as a sandbox, and capturing their runtime behaviors.

Dynamic analysis provides insights into the behavior of malware during execution. It involves monitoring various aspects, such as memory access patterns, registry modifications, network traffic, and system interactions. By executing malware samples and observing their behavior, dynamic analysis can reveal additional information about the intentions, capabilities, and potential threats posed by the malware.

Combining static and dynamic features would enable the classification model to leverage the strengths of both approaches. Static analysis offers insights into the structural and static characteristics of malware, while dynamic analysis provides information about the runtime behaviors and interactions with the system. By incorporating dynamic features, the classification model can gain a deeper understanding of the malware's intentions, capabilities, and potential threats, enhancing its accuracy and effectiveness in identifying and classifying malware samples.

5.3 Conclusion

In conclusion, the work in this master thesis has addressed the task of malware classification employing deep learning models, aiming to enhance cybersecurity measures and mitigate emerging digital threats. The exploration conducted in this thesis has demonstrated the effectiveness of multimodal deep learning approaches, particularly the combination of images and tabular features for the classification task. By integrating these diverse data modalities, the multimodal has shown improved accuracy and robustness in classifying malware family. These findings contribute to the field by highlighting the importance of leveraging multiple data sources for malware classification and for further advancements in cybersecurity.

Bibliography

[1] AV-Test Institute. https://portal.av-atlas.org/malware. Accessed: June 6, 2023.

[2] Yuxin Gao, Zexin Lu, and Yuqing Luo. Survey on malware anti-analysis. In *Fifth International Conference on Intelligent Control and Information Processing*, pages 270–275. IEEE, 2014.

[3] Guillermo Suarez-Tangil, Santanu Kumar Dash, Mansour Ahmadi, Johannes Kinder, Giorgio Giacinto, and Lorenzo Cavallaro. Droidsieve: Fast and accurate classification of obfuscated android malware. In *Proceedings of the seventh ACM on conference on data and application security and privacy*, pages 309–320, 2017.

[4] Philip O'kane, Sakir Sezer, and Kieran McLaughlin. Detecting obfuscated malware using reduced opcode set and optimised runtime trace. *Security Informatics*, 5:1–12, 2016.

[5] Tristan Carrier, Princy Victor, Ali Tekeoglu, and Arash Habibi Lashkari. Detecting obfuscated malware using memory feature engineering. In *ICISSP*, pages 177–188, 2022.

[6] Simone Aonzo, Yufei Han, Alessandro Mantovani, and Davide Balzarotti. Humans vs. Machines in Malware Classification. working paper or preprint, November 2022.

[7] Igor Santos, Felix Brezo, Xabier Ugarte-Pedrero, and Pablo G Bringas. Opcode sequences as representation of executables for data-mining-based unknown malware detection. *information Sciences*, 231:64–82, 2013.

[8] Mohammad M Masud, Latifur Khan, and Bhavani Thuraisingham. A scalable multi-level feature extraction technique to detect malicious executables. *Information Systems Frontiers*, 10:33–45, 2008.

[9] Trung Kien Tran and Hiroshi Sato. Nlp-based approaches for malware classification from api sequences. In *2017 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES)*, pages 101–105. IEEE, 2017.

[10] Chen Li and Junjun Zheng. Api call-based malware classification using recurrent neural networks. *Journal of Cyber Security and Mobility*, pages 617–640, 2021.

[11] Aparna Sunil Kale, Fabio Di Troia, and Mark Stamp. Malware classification with word embedding features. *arXiv preprint arXiv:2103.02711*, 2021.

[12] Qi Xie, Yongjun Wang, and Zhiquan Qin. Malware family classification using lstm with attention. In *2020 13th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 966–970. IEEE, 2020.

[13] Lakshmanan Nataraj, Sreejith Karthikeyan, Gregoire Jacob, and Bangalore S Manjunath. Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security*, pages 1–7, 2011.

[14] Danish Vasan, Mamoun Alazab, Sobia Wassan, Hamad Naeem, Babak Safaei, and Qin Zheng. Imcfn: Image-based malware classification using fine-tuned convolutional neural network architecture. *Computer Networks*, 171:107138, 2020.

[15] Barath Narayanan Narayanan and Venkata Salini Priyamvada Davuluru. Ensemble malware classification system using deep neural networks. *Electronics*, 9(5):721, 2020.

[16] Xuan Wu, Yafei Song, Xiaoyi Hou, Zexuan Ma, and Chen Chen. Deep learning model with sequential features for malware classification. *Applied Sciences*, 12(19):9994, 2022.

[17] Kaspersky. `https://www.kaspersky.com/resource-center/threats/data-theft`. Accessed: July 01, 2023.

[18] NordVPN. `https://nordvpn.com/blog/remote-access-trojan/`. Accessed: July 01, 2023.

[19] Kaspersky. `https://www.kaspersky.fr/resource-center/threats/ransomware`. Accessed: July 01, 2023.

[20] Kaspersky. `https://www.kaspersky.fr/resource-center/threats/adware`. Accessed: July 01, 2023.

[21] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

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

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[24] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 689–696, 2011.

[25] Nitish Srivastava and Russ R Salakhutdinov. Multimodal learning with deep boltzmann machines. *Advances in neural information processing systems*, 25, 2012.

[26] Naufal Hilmizen, Alhadi Bustamam, and Devvi Sarwinda. The multimodal deep learning for diagnosing covid-19 pneumonia from chest ct-scan and x-ray images. In *2020 3rd international seminar on research of information technology and intelligent systems (ISRITI)*, pages 26–31. IEEE, 2020.

[27] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134:19–67, 2005.

[28] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.

[29] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information processing & management*, 45(4):427–437, 2009.

[30] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi. Microsoft malware classification challenge. *arXiv preprint arXiv:1802.10135*, 2018.

[31] Microsoft. Malware classification challenge (big 2015). Retrieved from `https://www.kaggle.com/c/malware-classification`, 2015.

[32] Justin Ferguson. *Reverse engineering code with IDA Pro.* Syngress, 2008.

[33] Chris Eagle. *The IDA pro book.* no starch press, 2011.

[34] Mahmoud Kalash, Mrigank Rochan, Noman Mohammed, Neil DB Bruce, Yang Wang, and Farkhund Iqbal. Malware classification with deep convolutional neural networks. In *2018 9th IFIP international conference on new technologies, mobility and security (NTMS)*, pages 1–5. IEEE, 2018.

[35] David Kornish, Justin Geary, Victor Sansing, Soundararajan Ezekiel, Larry Pearlstein, and Laurent Njilla. Malware classification using deep convolutional neural networks. In *2018 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, pages 1–6. IEEE, 2018.

[36] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[37] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

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

[39] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[40] Nils Gessert, Maximilian Nielsen, Mohsin Shaikh, René Werner, and Alexander Schlaefer. Skin lesion classification using ensembles of multi-resolution efficientnets with meta data. *MethodsX*, 7:100864, 2020.

[41] Thorsten Brants, Ashok C Popat, Peng Xu, Franz J Och, and Jeffrey Dean. Large language models in machine translation. 2007.

[42] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[43] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

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

[45] Jinzhang Zhang, Kok Kwang Phoon, Dongming Zhang, Hongwei Huang, and Chong Tang. Deep learning-based evaluation of factor of safety with confidence interval for tunnel deformation in spatially variable soil. *Journal of Rock Mechanics and Geotechnical Engineering*, 13(6):1358–1367, 2021.

[46] Bo Jiang, Xuegong Zhang, and Tianxi Cai. Estimating the confidence interval for prediction errors of support vector machine classifiers. *The Journal of Machine Learning Research*, 9:521–540, 2008.