Security through the Eyes of AI: How Visualization is Shaping Malware Detection

Matteo Brosolo
Department of Mathematics,
University of Padova, Italy
matteo.brosolo@unipd.it

Mauro Conti
Department of Mathematics,
University of Padova, Italy
mauro.conti@unipd.it

Asmitha K. A.
Department of Computer Applications,
Cochin University of Science
and Technology, Kochi, India
asmitha@pg.cusat.ac.in

Rafidha Rehiman K. A.
Department of Computer Applications,
Cochin University of Science
and Technology, Kochi, India
rafidharehimanka@cusat.ac.in

Muhammed Shafi K. P.
Department of Computer Applications,
Cochin University of Science
and Technology, Kochi, India
shafikp@cusat.ac.in

Serena Nicolazzo
Department of Computer Science,
University of Milan,
G. Celoria, 20, Milan, Italy
serena.nicolazzo@unimi.it

Antonino Nocera
Department of Electrical, Computer
and Biomedical Engineering,
University of Pavia,
A. Ferrata, 5, Pavia, Italy
antonino.nocera@unipv.it

Vinod P.
Department of Computer Applications,
Cochin University of Science
and Technology, Kochi, India
vinod.p@cusat.ac.in

Abstract

Malware, a persistent cybersecurity threat, increasingly targets interconnected digital systems such as desktop, mobile, and IoT platforms through sophisticated attack vectors. By exploiting these vulnerabilities, attackers compromise the integrity and resilience of modern digital ecosystems. To address this risk, security experts actively employ Machine Learning or Deep Learning-based strategies, integrating static, dynamic, or hybrid approaches to categorize malware instances. Despite their advantages, these methods have inherent drawbacks and malware variants persistently evolve with increased sophistication, necessitating advancements in detection strategies. Visualization-based techniques are emerging as scalable and interpretable solutions for detecting and understanding malicious behaviors across diverse platforms including desktop, mobile, IoT, and distributed systems as well as through analysis of network packet capture files. In this comprehensive survey of more than 100 high-quality research articles, we evaluate existing visualization-based approaches applied to malware detection and classification. As a first contribution, we propose a new all-encompassing framework to study the landscape of visualization-based malware detection techniques. Within this framework, we systematically analyze state-of-the-art approaches across the critical stages of the malware detection pipeline. By analyzing not only the single techniques but also how they are combined to produce the final solution, we shed light on the main challenges in visualization-based approaches and provide insights into the advancements and potential future directions in this critical field.

***K*eywords** Visualization-based Malware detection, Neural Networks, Adversarial Attack, Concept drift, Explainability

1 Introduction

Malware samples are becoming increasingly sophisticated, often employing obfuscation and polymorphic techniques to evade traditional signature-based and behavior-based detection. Moreover, due to the increase in reliance on technology and connectivity, malware analysts have noted an expansion in the scope and strength of attacks against various types of infrastructure. Windows malware, in particular, has become a critical area of concern due to its prevalence and the potential for significant financial losses, as well as the threat it poses to the security of sovereign nations. Windows remains the operating system most targeted for malware, with ransomware attacks increasing 2.75 times in 2024, and 92% of these attacks affecting Windows devices[99]. Additionally, threat actors in 2025 are increasingly exploiting Windows vulnerabilities through AI-driven malware[110]. The widespread use of Windows-based systems in personal, enterprise, and government environments has made them a prime target for attackers seeking to exploit vulnerabilities in the operating system and associated software [41].

Other platforms, such as Android, are not safe either, and an increase in Android malware has also been observed in recent years after the widespread adoption of mobile phones [25]. The threats against Android devices differ in some aspects from those of their Windows counterparts. Firstly, the popularity and widespread adoption of Android mobile devices make them an attractive target for cyber-criminals, whereas Windows malware has historically targeted desktop computers. Additionally, the open nature of the Android platform allows for easier distribution of malicious apps. Furthermore, Android malware often uses different techniques and vulnerabilities specific to the mobile environment. Malware propagates across desktop and mobile platforms through various means. On desktops, it can infiltrate systems via malevolent email attachments, compromised websites, or tainted software downloads. Once present, malware exploits vulnerabilities, self-replicates, and extends its reach to connected devices and networks [8]. On mobile platforms, malware commonly disguises itself as legitimate apps or leverages third-party app stores. Users unknowingly install these pernicious apps, thus granting access to sensitive data or device control. Additional dissemination routes encompass text messages, phishing links, and compromised Wi-Fi networks, accentuating the substantial risk posed to users on both PC and mobile platforms [70].

Another fertile ground for malware is the IoT landscape. Malware spreads in IoT devices through the exploitation of vulnerabilities in interconnected systems, allowing it to quickly infiltrate and compromise a multitude of smart devices. The interconnected nature of IoT ecosystems provides a fertile ground for the swift propagation of malicious software, thus fostering the need for robust security measures to mitigate these risks [141].

In this context, security researchers face the challenge of combating widespread malware campaigns and safeguarding against both new and familiar strains. However, traditional approaches that rely on signatures are inadequate in the face of ever-evolving threats. Attackers employ sophisticated methods, such as polymorphic and metamorphic malware, to avoid detection by altering their code. Consequently, there is an increasing demand for innovative techniques such as behavioral analysis, Machine Learning, and dynamic analysis. These methods provide deeper insights into malware behavior, enabling analysts to better detect and respond to threats.

In this paper, we focus on one particular aspect, namely the use of malware visualization for the classification of malware, which is a novel and advanced method for malware analysis. With the integration of Deep Learning and image-based analysis, visualization methods have shown great promise in recent years for creating interpretable results for a variety of machine learning tasks, including malware classification [144]. In particular, Convolutional Neural Networks (CNNs) are effective in malware classification due to their ability to extract features from binary representations of samples without the help of domain experts.

We survey the state-of-the-art in machine learning-based malware classification across various platforms, focusing on high-quality works published over the past seven years, from 2018 to 2025. We categorize the research literature according to the fundamental steps of the visualization-based malware classification, namely: Dataset Collection, Image Generation, Feature Extraction, Classification, Evaluation, Model Robustness, and Adaptation. Furthermore, we explore the challenges associated with this approach, including the need for large and diverse datasets as well as the potential for adversarial attacks, and the most important technique for interpretability. Despite growing interest, visualization-based malware detection remains a relatively young field. Hence, by gaining a deeper understanding of the strengths and limitations of Machine Learning-based approaches to malware classification, we can better evaluate their potential for improving the accuracy and efficiency of malware analysis. Through our survey of the literature, we provide a comprehensive overview of the current state of the art in this area, propose best practices, and help unify future research identifying areas for forthcoming development.

In summary, the main contributions of this survey are as follows.

1. To the best of our knowledge, we are the first to have a comprehensive and methodologically rigorous survey on existing studies that employ visualization-based techniques in malware detection, structured around the proposed unified framework that captures key characteristics of existing approaches and enables systematic comparison, trend analysis, and identification of research gaps.
2. We distill the entire pipeline of visualization-based malware detection from numerous concrete image-based approaches. This provides readers with an overview, aiding in understanding the general process of visualization-based methods. The overview covers the datasets used, image representation, feature processing techniques, model generation, performance evaluation, and considerations about robustness and model adaptation.
3. We explore the aspects of explainability in various visualization techniques applied to malware detection, providing valuable insights.
4. To gain a deeper understanding of adversarial attacks in visualization-based malware detection and of the existing defenses against them, we systematically survey various state-of-the-art approaches. This includes exploring adversarial attack methods on ML-based malware classifiers, DL classifiers, and defensive strategies to enhance the adversarial robustness of visualization-based malware classifiers.
5. We delve deeper into the gaps present in current studies and thoroughly explore future research challenges and directions within the domain of visualization-based malware analysis and detection. This discussion presents readers with potential avenues for developing innovative solutions.

This survey is organized as follows. Section2 introduces related surveys, while Section 3 outlines the approach adopted to carry out this survey. In section 4, we present an overview of the main background concepts about malware detection and classification techniques. Section 5 describes the current literature classified according to the steps of visualization-based malware classification, namely Dataset Collection, Image Generation, Feature Extraction, Classification, and Evaluation. In particular, in Section 5.1 we examine the most used dataset about malware. Section 5.2 deals with the techniques employed to convert a malware file into an image, utilizing diverse file extensions acquired through static and dynamic analysis approaches. In Section 5.3, we explore how researchers have approached the processing of features, which they subsequently use as input for classifiers. Section 5.4 is devoted to the description of the various Machine Learning methods used to classify images through their extracted features. Section 5.5 provides a discussion of how authors decided to evaluate their classifiers, compare their results with those of other researchers, consider relevant parameters, and assess whether the choices made are fair. Section 5.6 provides a description of model robustness and adaptation, specifically targeting adversarial attacks targeting visualization-based malware detectors. In Section 6, we explore the issue of interpretability in the context of malware classification, examining how authors have addressed the problem and suggesting potential improvements for the situation. Section 8 lists the lessons learned from our investigation. Section 9 examines several open challenges and offers perspectives on possible directions for future research. Finally, Section 10 gives some general considerations on the state of research in the field of image-based malware classification and tries to advise to move forward.

2 Related Work

This section offers an overview of surveys that explore visual malware classification. For each survey, we describe along with its advantages and disadvantages. It is worth noting that most of the surveys dealt with general techniques in malware detection and only a few others addressed the visualization of visual malware images despite brief mentions of the concept in surveys with broader scopes. Table 1 offers a summary of the reviewed surveys, in which we consider the year of publication, the number of analyzed papers, and the main scope of the work (i.e., image generation technique analysis, feature extractors analysis, classifiers analysis, interpretability analysis, sustainability analysis, different training methods analysis, dataset coverage, adversarial attacks analysis, few-shot analysis, metrics analysis.

2.1 General Machine Learning and Malware Detection Surveys

In [77] the authors provides an extensive examination of ML models applied to malware classification and detection. The survey's main focus is not specifically on visualization methods. Emerging and popular challenges, such as interpretability and sustainability are briefly mentioned. Despite these limitations, this survey serves as a valuable starting point, providing an overview of traditional Machine Learning and Deep Learning workflows used in malware classification and detection. Ucci et al. in [207] give a thorough analysis of the Machine Learning techniques. They provide a well-thought taxonomy to categorize the different Machine Learning methods. The section on feature extraction techniques covers the subject in great depth. This survey excels in analyzing Machine Learning techniques broadly, but it falls short in giving adequate attention to recent advancements, such as Deep Learning, and specifically,

Table 1: Comparison with existing survey papers

Ref.	Year	Literature timeline	#Papers	Paper Scope									
				DC	IG	FE	$Class$	$Inter$	S	$Diff\_TM$	AA	FS	M
[77]	2020	2008-2019	67	○	●	◐	●	◐	◐	○	◐	○	◐
[155]	2021	2009-2020	20	◐	◐	◐	◐	○	○	○	◐	◐	◐
[249]	2021	2009-2019	-	○	○	◐	◐	○	○	○	○	○	○
[207]	2019	2001-2017	64	◐	●	◐	◐	○	○	○	○	○	○
[186]	2022	2016-2020	4	○	○	◐	○	○	○	○	○	○	○
[138]	2022	-	10	○	◐	◐	◐	○	○	○	○	○	○
[139]	2023	-	-	○	◐	◐	◐	○	○	○	◐	○	○
[60]	2023	-	-	◐	◐	◐	◐	○	◐	○	●	◐	◐
[32]	2024	2018-2023	-	◐	◐	◐	◐	◐	◐	◐	◐	◐	◐
Our	2025	2018-2025	103	●	●	●	●	●	●	●	●	●	●

○- Topic not covered or minimally analyzed; ◐- Topic addressed to a reasonable extent; ●- Topic covered in depth. DC: Dataset Coverage, IG: Image generation technique analysis, FE: Feature extractors analysis, $Class$: Classifiers analysis, $Inter$: Interpretability analysis, S: Sustainability analysis, $Diff\_TM$: Different training methods analysis, AA: Adversarial attacks analysis, Fs: Few-shot analysis, M: Metrics analysis.

the utilization of CNNs. Additionally, the survey lacks sufficient emphasis on the contemporary challenges researchers face today. Naik et al. in [155] provide a general overview of image-based malware analysis. The authors discuss the main steps of the workflow from the dataset to the evaluation. The inclusion of 20 analyzed papers also highlights the study's significance. However, it is essential to acknowledge the limitations of this study, such as the lack of in-depth analysis regarding datasets and feature extraction. Moreover, it is noteworthy that there is no discussion on current issues like interpretability and sustainability. Gopinath et al. [139] present a comprehensive survey encompassing a broad range of papers that employ Machine Learning and Deep Learning techniques. The survey primarily examines various approaches to developing classifiers rather than focusing on image-based visualization methods. The authors explain each paper with due detail, characterizing each model well. The survey divides the publications based on the approach used for the classifier, therefore each paper is named and explained just once. For this reason, it is hard to abstract the information received and categorize the different approaches to feature extraction and image generation. Furthermore, the authors do not extensively delve into contemporary challenges encountered in the field, aside from brief tangential discussions.

2.2 Visualization-Based Malware Detection Surveys

Ahmad et al. [138] survey visualization-based malware detection and classification. The authors present a detailed review of existing malware detection methods, leveraging both ML and DL techniques, and outline the implementation of eleven distinct models. This survey effectively identifies the essential components of each model and employs a concise listing approach, facilitating comparisons between them. However, It is worth mentioning that we have conducted a thorough analysis of the entire pipeline of visualization approaches. Shah et al. in [186] analyze visualization-based malware classification models. We particularly focus on the computational performance of the model. However, it is crucial to emphasize that the number of models considered in this survey is limited. Therefore, the authors cannot provide extensive insights into the broader landscape of visualization-based malware analysis. Deldar et al. [60] provide a survey specifically on the detection of zero-day malware. The taxonomy employed neatly divides the models and enables the reader to identify the possible choices at each step of the classification pipeline. However, the survey does not delve into the different possibilities in depth; instead, it merely presents or briefly discusses them. Remarkable is exploring the subject of most interest for zero-day classification, meaning adversarial attacks and few-shot learning. The authors do not directly deal with modern problems in Machine Learning malware visual classification, like sustainability and interpretability. The researchers in [249] apply computer vision to network security, broadly covering phishing detection, malware detection, and traffic anomaly detection. They highlight its broader cybersecurity implications, including applications in physical security and critical infrastructure protection, and bridge computer vision, machine learning, and network security. However, their study lacks an in-depth focus on visualization-based malware detection. In contrast, our study provides a detailed analysis of visualization techniques, starting from dataset collection, image types, feature extraction, classification methods, interpretability, robustness evaluation, and adaptation in malware detection.

3 Methodology

In this study, we design a structured search strategy to gather relevant research on visualization-based malware detection according to the objective of our survey article. Initially, we explore academic databases such as Google Scholar and Web of Science and review reputable repositories such as SpringerLink, IEEE Xplore, ScienceDirect, and ACM Digital Library. The search focuses on publications from 2018 to 2025 to ensure a comprehensive examination of recent advancements. An initial global scan reveals that, before 2018 research on visualization-driven malware detection remains limited, with only a few notable publications. As a result, we select a seven-year time frame for analysis to capture the most relevant and recent works. Recently, research on visualization-based malware detection has expanded rapidly, driving significant innovation in the field.

We ensure comprehensive coverage by formulating search queries using specific keywords. The primary search terms include "malware visualization" and "image-based malware detection". To refine our search, we incorporate additional terms such as "Deep Learning", "feature extraction from malware images", "Machine Learning", "Vision Transformers", "adversarial robustness in malware images", "obfuscation", "interpretability", and "sustainability". This approach captures a broad spectrum of studies related to malware visualization and classification. Figure 1 visually represents our study selection process in the PRISMA flow diagram, detailing the number of studies we identify, screen, exclude, and include in our final review.

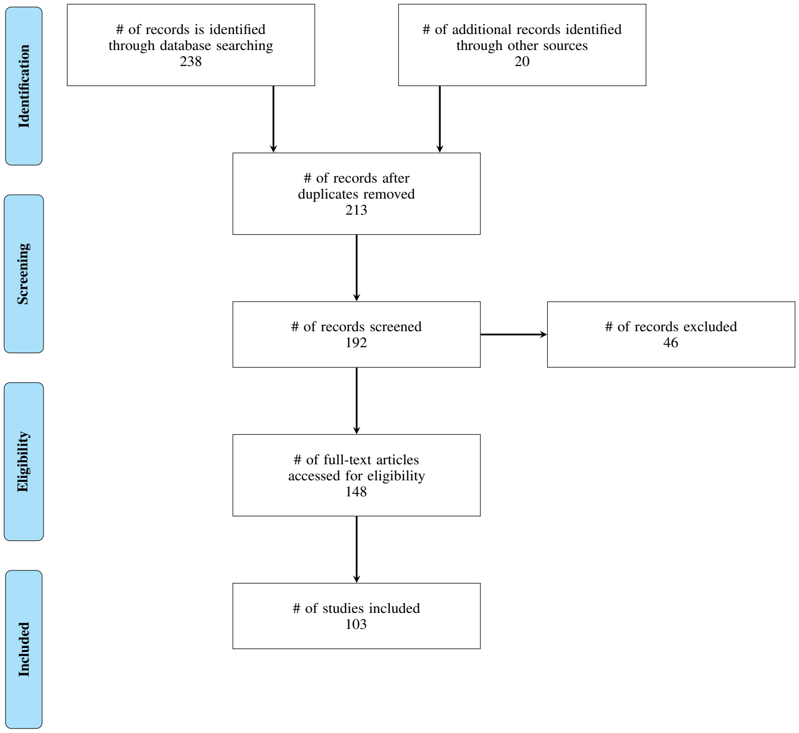
Identification
Screening
Eligibility
Included
# of records is identified through database searching 238
# of additional records identified through other sources 20
# of records after duplicates removed 213
# of records screened 192
# of records excluded 46
# of full-text articles accessed for eligibility 148
# of studies included 103

Figure 1: PRISMA Diagram

3.1 Selection and Filtering Criteria

This section explains the methodology we use to evaluate the quality and relevance of the academic papers considered in this survey on visualization-based malware detection. Articles are selected when they fulfill at least one inclusion criterion and are not disqualified by any exclusion criteria. We evaluate the suitability of the paper through the following selection criteria.

3.1.1 Inclusion Guidelines

We evaluate a paper's suitability for this survey article based on the following selection criteria

- The significance and credibility of the publication venue according to Scimago and Core.edu rankings.
- The citation impact through rankings on Google Scholar and Scopus.

- The publication date. In particular, we prioritize recent research, especially studies published in the last six years.
- Significance of the contribution to malware visualization.

3.1.2 Exclusion Guidelines

We also exclude studies that meet any of the following conditions.

- Publication date before 2018, as the article would fall outside the seven-year review period.
- Lack of focus on visualization-based malware detection.
- Lack of peer review and/or publication in a language other than English.

Table 2 provides a summary of the acronyms used throughout the paper.

Table 2: List of the acronyms used in the paper

Acronyms	**Description**
CNN	Convolutional Neural Network
DL	Deep Learning
DT	Decision Tree
ELM	Extreme Learning Machine
GAN	Generative Adversarial Network
IoT	Internet of Things
KNN	K-Nearest Neighbors
ML	Machine Learning
RF	Random Forest
RNN	Recurrent Neural Network
SFC	Space-Filling Curve
SVM	Support Vector

4 Background

An important task in cybersecurity is the categorization of malware in families. This task is crucial for categorizing malware based on characteristics and behavior, enabling researchers to understand its functionality and develop effective countermeasures. It also reveals trends in attacker techniques, guiding proactive prevention strategies. Malware classification differs from malware detection, which identifies the presence of malware in a system. Malware classification categorizes and organizes various types of malware. In contrast, malware detection directly identifies specific malware instances, allowing timely intervention to prevent or minimize their impact. When analyzing malware, researchers can use different techniques to help explore the structure and behavior of the malicious code. These techniques broadly fall into two macro-categories: *(i)* dynamic analysis and *(ii)* static analysis. Security researchers must tackle the challenge of widespread malware campaigns and identify and protect from new and old malware strains. However, traditional signature-based approaches to malware analysis are no longer enough in the face of constantly evolving malware [13]. Attackers use increasingly sophisticated techniques to evade detection, such as polymorphic and metamorphic malware, which can alter their code to avoid signature detection. As a result, there is a growing need for novel and advanced methods to analyze malware, including behavioral analysis, and integrating *(i)* Machine Learning and Deep Learning. These techniques allow for a more in-depth understanding of the behavior of malware and its impact on systems, enabling analysts to identify and respond to threats more effectively.

4.1 Static Analysis

Static analysis is a technique employed in malware analysis that involves examining the binary or code of a malicious program without executing it. This approach focuses on analyzing the structural and content aspects of malware samples to identify potential malicious behaviors and understand their operational mechanisms without direct execution. Widely used tools for static analysis include disassemblers, decompilers, and debuggers, which facilitate the extraction of the instructions used by malware to execute malicious actions, such as data theft, system modification, or propagation. In addition, static analysis enables the identification of indicators of compromise (IOCs), such as file names, network traffic patterns, registry keys, and other attributes associated with malware. By examining the code or binary, researchers can extract signatures or behavioral patterns that aid in detecting the presence of malware on a system [158]. These signatures can detect malware in a system by identifying matching patterns [219]. A convenient approach explored in this survey is to represent information extracted through static analysis by visualizing it in images, primarily due to visual similarity among variants of the same malware. Using images helps researchers to have a global view of the entire malware without losing local details. Combining feature extractors and Machine Learning models can use this characteristic to achieve accurate classification and detection results.

4.2 Dynamic Analysis

Dynamic analysis is a technique for examining malware behavior by executing it in a secure controlled environment, such as a sandbox or virtual machine. Unlike static analysis, which inspects code without running it, dynamic analysis allows direct observation of malware interactions with the operating system and network resources. Online sandbox platforms, including Any.Run [73], VirusTotal [194], Joe Sandbox [104], Cuckoo Sandbox [51], and Hybrid Analysis [10] provide automated environments for the safe execution of malware and the generation of detailed behavioral reports. Security analysts employ monitoring tools, such as system monitors, network analyzers, and debuggers, to detect malicious activities such as file creation, system modification, or communication with command-and-control (C2) servers. Furthermore, dynamic analysis reveals evasion techniques, such as anti-debugging or antivirtualization strategies [45], improving the understanding of sophisticated malware. By analyzing malware behavior within an isolated environment, analysts can more accurately assess the nature and potential threat of malicious code. Moreover, dynamic analysis can be augmented through visualization techniques, where run-time behavioral data is transformed into image-based representations, facilitating clearer interpretation and efficient handling of emerging malware variants.

4.3 Machine Learning

Machine learning has become a fundamental approach in malware analysis, enabling automated classification and detection of malicious software based on its inherent characteristics and behavior [207]. By examining various features, including file attributes (such as size and type), system interactions, network activity, and other indicators, machine learning algorithms can effectively distinguish between benign and malicious software. ML models monitor the runtime behavior of malware during dynamic analysis, detecting malicious patterns by examining its interactions with system components and network communications. In static analysis, these algorithms assess code structure and binaries to detect malicious patterns. The adoption of deep learning, particularly Deep Neural Networks (DNNs), represents a significant advancement in this domain. Over the past decade, deep learning techniques have gained prominence due to their remarkable performance across various fields, including image recognition and natural language processing. Convolutional neural networks (CNNs) excel in malware analysis by providing superior performance in malware visualization compared to traditional machine learning models. Despite their reliance on substantial computational resources and extensive datasets, DNNs have significant potential to improve malware detection and classification.

4.4 Malware Visualization

Malware visualization is an analytical technique that transforms the behavior and characteristics of malicious software into visual representations, facilitating its analysis and classification. Visualization provides a clear view of system and network activities, helping analysts identify patterns, uncover connections, and detect possible attack pathways. Visualization techniques generate graphical representations of different malware attributes, including network interactions, file operations, and system calls. By examining these visual representations, analysts can gain insights into the malware's operational techniques, such as evasion strategies and exploited vulnerabilities. In addition to aiding malware detection and analysis, visualization can also enhance classification tasks by depicting malware samples as images, facilitating the differentiation between various strains. Furthermore, malware visualization is instrumental in identifying minor variations among related malware samples. As malware authors frequently produce new variants with slight code modifications, visualization effectively captures these differences, aiding in the detection of related strains. By leveraging visual representations of malware behavior in combination with machine learning techniques, experts can efficiently identify similar and distinct patterns in malware samples, facilitating the classification of new or previously unknown threats.

5 Unified Framework for Visualization-based Malware Detection

This paper presents a unified framework for visualization-based malware detection. This framework aims to comprehensively survey various state-of-the-art research in the visualization domain and to identify future research directions to improve security. We outline the fundamental process of visualization-based malware classification into five phases: Dataset Collection, Image Generation, Feature Extraction, Classification, Evaluation, and Model

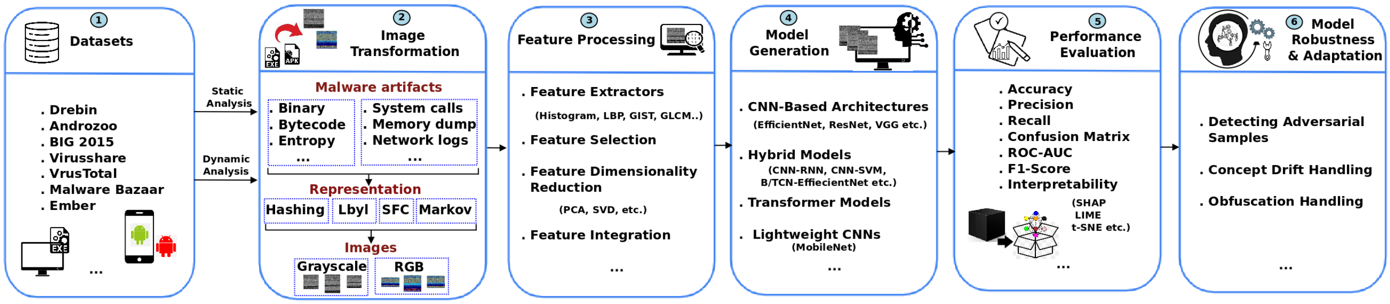

Figure 2: An End-to-End Framework for Visualization-Based Malware Detection Across Diverse and Evolving Threat Landscapes.

Robustness and Adaptation. These steps form the pipeline as seen in Figure 2. Figure 4 provides a more detailed taxonomy of the first part of the pipeline, from the selection of the file type to the feature extraction step.

1. **Dataset Collection**. Researchers collect datasets of malware and benign samples in row form or image format in this step. Some benchmark datasets are designed explicitly for visualization-based malware detection, such as MalImg.
2. **Image Transformation**. This phase distinguishes visualization-based malware detection approaches from other methods of malware detection. During this phase, researchers generate several images for each malware, employing various algorithms such as B2IMG and Word2Vec.
3. **Feature Processing**. In this phase, high-level features are extracted from malware images to highlight essential patterns, reduce dimensionality, and generate meaningful input for the classification model. Techniques such as Local Binary Patterns (LBP), GIST descriptors, and dimensionality reduction methods like Principal Component Analysis (PCA) can be employed to enhance feature quality and efficiency.
4. **Classification**. For predicting input features, we opt for Machine Learning (ML) and Deep Learning (DL) models as malware classifiers. Basic ML models like Support Vector Machine(SVM), Random Forest(RF), XGBoost, etc., and DL models such as Convolutional Neural Networks(CNN), Deep Neural Networks(DNN), etc., can serve as effective malware classifiers.
5. **Evaluation**. Visualization-based methods quantitatively evaluate the predictions of individual classifiers using accuracy measures and qualitatively assess them using explainability and interpretable terms. Binary classification methods provide probabilities for malicious or benign samples, and multi-classification methods offer probabilities for various malware families.
6. **Model Robustness and Adaptation**. This phase involves techniques to enhance the resilience of the model against adversarial attacks, obfuscation techniques, and concept drift. Additionally, adaptive learning techniques ensure that the model remains effective against evolving malware patterns.

5.1 Dataset Collection

Researchers have extensively explored a wide range of datasets in their experimental investigations. The selection of appropriate datasets is paramount, as it forms the foundation for conducting rigorous and reliable research. Consequently, it is crucial to identify and prioritize datasets that are most prevalent and widely used within the domain.

When researchers seek malware samples, they can adopt two distinct approaches. The most common approach involves selecting a malware dataset explicitly designed for classification tasks[236, 159, 200, 227, 117, 52]. These datasets provide readily available information that other researchers can easily leverage to construct image representations. However, these datasets have limitations regarding the flexibility of the available information and their overall size. An alternative approach is to create a dataset by utilizing expansive malware databases such as Malware Bazaar, Virus Share, or VX Heaven[93, 152, 21, 111, 102]. This approach offers authors more freedom in selecting modern malware families to consider and determining the type of information to gather. However, it comes with the trade-off of increased complexity in collecting the data and the opportunity to benchmark their model against services such as VirusTotal(VT). Researchers often tailor datasets created using malware repositories(MalwareBazaar, VirusShare, VX-Underground, etc.) to the specific use cases presented in their respective research papers, resulting in significant variations in content [222, 48].

The MalImg and BIG2015 datasets have gained significant attention in studies focusing on executable malware, as highlighted in Table 3. These datasets collectively account for a substantial percentage of citations among the datasets. Among them, the MalImg dataset [157] is the most frequently cited dataset for visualization-based malware detection using PE malware. It is a publicly available image dataset comprising $9,339$ malware executables from 25 families. The dataset derives its executables from the Anubis analysis system, converting each into a grayscale image by mapping individual bytes to pixels. This dataset comprises a mixture of packed and unpacked malwaanging from e, with specific malware families such as Yuner.A, VB.AT, Malex.genJ, Autorun.K, and Rbotgen being packed using UPX [175]. Additionally, the dataset exhibits a high level of imbalance, spanning from $2,949$ samples belonging to the Allaple.A family to merely 80 samples associated with the Skintrim.N family.

The second most cited dataset, the Microsoft Malware Classification Challenge Dataset (BIG 2015 or MMCC) [178], comprises $10,860$ malware executables belonging to 9 distinct families. Each malware executable in the dataset has a unique identifier consisting of a 20-character hash value paired with an integer denoting the malware family name. In addition, the dataset includes metadata that collects various information, such as function calls, strings, and more, from the binaries. It also provides assembly files associated with each binary. Similar to MalImg, the dataset exhibits imbalanced distribution among the malware families. Furthermore, the Dumpware10 dataset [29] consists of $3,686$ malware executable samples belonging to 10 different families. The dataset also includes 608 benign samples. The images in this dataset are in PNG format and available in dimensions 224, 300, and 4096.

The Malevis dataset [28] is an RGB-based dataset specifically designed for multi-class malware categorization. It comprises $9,100$ training samples and $5,126$ validation samples, with images distributed across 26 classes. Among these classes, 25 correspond to different types of malware, while one class represents benign samples. The dataset was constructed by extracting binary images from malware files provided by Comodo Inc. The images are available in two square resolutions, namely 224×224 and 300×300 pixels. Additionally, the Malheur dataset [177], contributed by a security research team at the University of Erlangen, encompasses $3,131$ samples from 24 malware families. Among the research papers considered, samples from the VirusShare repository rank third in terms of usage. The authors of these papers generated their dataset by randomly selecting samples from VirusShare, resulting in variations in the number of families considered. Another frequently referenced dataset in the literature is the MALICIA dataset introduced in [156]. This dataset comprises samples collected by analyzing servers responsible for distributing drive-by malware downloads. In addition to these datasets containing raw executable malware samples, researchers have provided datasets consisting solely of pre-extracted feature vectors from the binaries. Notable examples include the EMBER dataset [12] and the more recent BODMAS dataset [240]. Such datasets typically offer a more significant number of samples and malware families, but the absence of actual binary files limits them.

Most of the research conducted on visualization-based Android malware has mainly focused on two datasets, Drebin and MalGenome, as visible in Table 3. The Drebin dataset holds significance in Android malware analysis as it contains $5,560$ malware samples and $123,453$ benign applications collected between August 2010 and October 2012. Similarly, the MalGenome dataset [252], collected between 2010 and 2011, consists of $1,260$ malware applications. It is important to note that these datasets are relatively old and, as technology advances, the samples they contain become outdated and less representative. The authors of [7] introduced a novel Android malware dataset for binary classification, comprising $16,868$ samples ($14,285$ malware, $2,583$ benign). This dataset, derived from the Drebin and Androzoo repositories, is publicly available, along with the accompanying code, to facilitate reproducibility and future research. Table 3 provides a comprehensive overview of the frequently used datasets for visualization-based malware detection. For each benchmark dataset listed, we provide accompanying details, namely the title of the dataset, the year of publication, the number of malware and benign samples, the number of families (identified by $\#Fam$), the time duration for which the samples are collected (identified as POT), the type of file identified as TOF (e.g., an image, apk, executable, elf, or another file extensions), the availability status of each dataset (i.e., whether it is publicly accessible or not), the total number of citations for the dataset identified as $\#Cit.$, the references leveraging the dataset. All datasets listed in Table 3, except AMD and Malgenome, are publicly available, facilitating further analysis and comparison. MalImg, Malheur, and Dumpware10 are image datasets, while all other datasets are available in their original .exe or .apk format. However, it is worth noting that AMD and MalGenome are no longer active projects, rendering their data inaccessible. As a result, the MalImg and Drebin datasets remain the primary choice for obtaining malware samples. Hence, researchers commonly use these datasets for visualization-based malware detection. However, specific datasets like BIG2015 have limitations, such as the unavailability of PE files and the absence of benign samples. These limitations hinder the comprehensive extraction of features and make these datasets more suitable for close-set inference tasks focused on identifying known malicious files rather than general malware detection.

Regarding Android malware, Drebin stands out as the most widely used dataset in numerous studies. However, [98] observed that the Drebin dataset may contain duplicate files, limiting access to genuine APKs and reducing the amount of usable malware data. Despite these limitations, researchers continue to extensively utilize Drebin and the aforementioned PE malware datasets, even in recent research, as the primary sources of malware data for developing

Table 3: The most used datasets for visualization-based malware detection

Dataset Name	Year	#Malicious Samples	#Benign Samples	#Fam.	POT	TOF	Avail.	#Cit.	Ref.
VX-Heaven	1990	0	0	0	1990-2010	any	✓	217	[233, 90, 129, 228, 91]
VirusTotal	2004	0	0	42	2004-2025	any	✓	0	[161, 102]
Leopard Mobile	2009	14,733	2,486	0	-	apk	✓	217	[154]
MalImg	2011	9,339	0	25	-	Grayscale	✗	1,089	[117, 52, 227, 78, 134, 6, 215, 154, 212, 29, 100, 211, 180, 80, 216, 168, 24, 15, 199, 11, 67, 251, 118, 143, 48, 162, 172, 183, 113, 190, 86, 245, 241, 213, 9, 7, 247, 189, 120]
Malheur	2011	3,131	0	24	2006-2009	.exe	✗	890	[152, 195]
VirusSign	2011	0	0	0	2020-2025	any	✓	0	[127]
Contagio Mobile	2011	0	0	0	2011-2025	apk	✓	0	[146, 24]
Malgenome	2012	1260	0	69	2010-2011	apk	✗	2,842	[66]
VirusShare	2012	61,455,426	0	0	2012-2025	any	✓	1,041	[93, 152, 21, 93, 212, 176, 221, 24, 129, 223, 67, 253, 190, 241, 105]
Drebin	2014	5,560	123,453	4	2010-2012	apk	✓	2,359	[192, 174, 62, 145, 238]
BIG2015	2015	10,860	0	9	2006-2015	.exe	✓	327	[236, 159, 200, 227, 133, 78, 112, 171, 250, 215, 29, 63, 180, 204, 168, 76, 57, 2, 199, 11, 228, 26, 101, 79, 118, 143, 48, 162, 37, 169, 61, 113, 190, 86, 241, 213, 7, 247, 243, 65]
AndroZoo	2017	22,707,999	0	42	2011-2025	apk	✓	0	[111]
AMD	2017	24, 553	0	71	2010-2016	apk	✗	440	[229, 146, 97]
CICAndMal 2017	2018	426	5, 065	42	2015-2017	apk	✓	217	[127]
Ember2018	2018	300, 000	300, 000	0	2012-2018	.exe	✓	338	[217, 134]
MaleVis	2019	8, 750	5, 476	25	2017-2018	RGB-image	✓	24	[67, 180, 241, 9, 7]
CICInvesAnd Mal2019	2019	426	5, 065	42	2017-2019	apk	✓	143	[62]
MDA	2019	500	500	10	2019-2025	csv	✓	23	[241]
SOREL-20M	2020	10, 000, 000	10, 000, 000	0	2020-2020	.exe	✓	122	[245]
CICMalDroid 2020	2020	17, 341	0	0	2017-2018	apk	✓	94	[174, 238]
MalwareBazaar	2020	0	0	0	2020-2025	any	✓	217	[222, 48]
Dumpware10	2021	3, 686	608	10	-	RGB image	✓	44	[29, 204]
BODMAS	2021	57, 293	7, 142	0	2019-2020	.exe	✓	45	[39, 113, 145]
Android Malware	2025	14, 285	2, 583	2	2025	Grayscale	✓	2	[7]

visualization-based algorithms targeting modern malware detection. The AMD, BIG2015, and Ember datasets are the exceptions to the rule, as they cover only short periods and do not account for the significant transformation and progression of malware over time. Neglecting this aspect can substantially impact the effectiveness of malware detectors, as the significance of features in effectively distinguishing malware may vary over time. Furthermore, many recent detection methods rely on outdated malware data sets to develop and evaluate solutions. This approach poses a potential risk to the models' ability to generalize to new and emerging malware.

For robust and accurate malware detection, it's essential to consider malware's dynamic nature and use up-to-date datasets reflecting the current threat landscape. Due to differing objectives and methodologies in sample collection, many datasets lack balance in the number of samples per family, with only a few authors addressing this issue. Commonly used techniques are oversampling families with fewer samples or undersampling families with the most samples [52] [26]. Other authors group different family variations in a single family with similar content to show a coarse-grained classification [78]. An alternative to these solutions proposed in [100] employs a weighted linear system to solve the weights of the output layer of the convolution neural network (CNN) to correct the bias of the Neural Network toward more common families.

Figure 3 illustrates the percentage of datasets used in the articles considered during the years 2018-2025. When researchers assess models, they face choices like how to partition datasets into training, validation, and test sets. The different approaches vary substantially, even on the same dataset. Often, clarification is needed on how the test set integrates with training. Some authors repurpose the test set for validation during training, reporting its metrics as final results. Others follow a similar process but assess the model across the entire dataset.

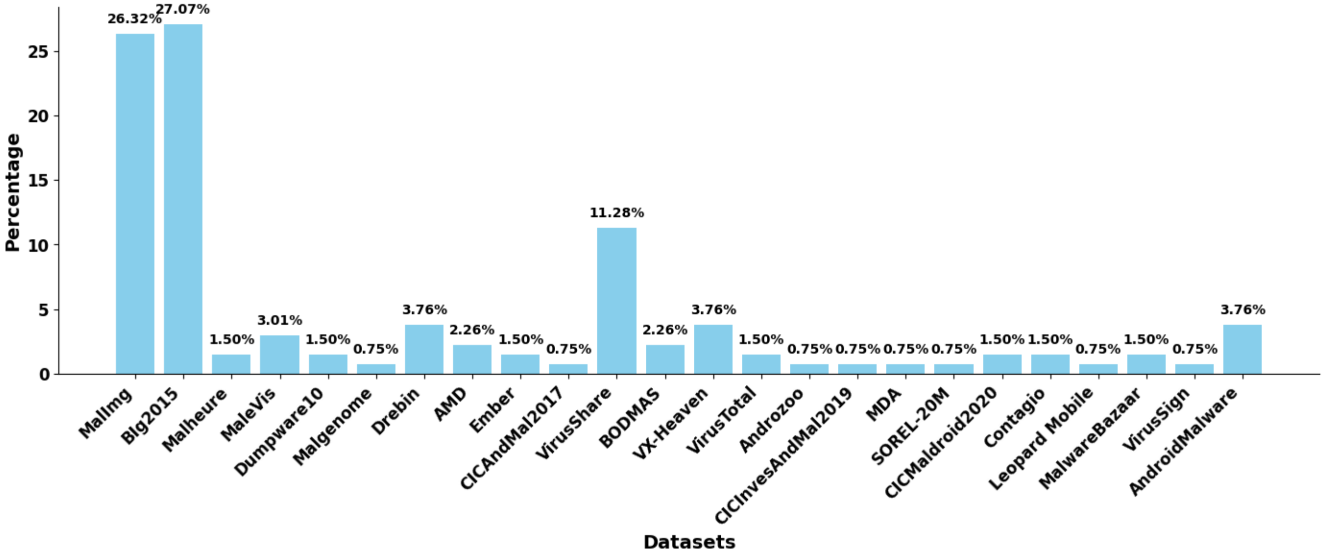

Figure 3: The percentage of datasets used in the considered papers during the year 2018-2025

Few reserve the test set exclusively for model evaluation. Moreover, specific datasets pose unique challenges. For instance, Big2015 comprises $10,000$ samples in both its training and test sets. Many articles only utilize the $10,000$ samples allocated for training in their experiments. Another problem is related to the issue of imbalance that we have discussed before. Different ways of addressing this issue may cause some models to outperform others simply because they oversampled or undersampled their dataset more effectively. Researchers must consider these problems because they limit the ability to confront and explain models and might generate results that are not representative of the model's real power. In the last few years, the accuracy of the best models reached almost 100% [61, 212].

5.2 Image Generation

This section defines the methods used in the literature to visualize malware files in image form. First, we introduce which characteristics, dynamic or static, are usually employed to generate images. After this, we explore the techniques used to translate the data into a graphical form.

5.2.1 Image Visualization Techniques

Our observations on the types of features utilized in visualization have led us to identify three distinct approaches: static visualization, dynamic visualization, and hybrid visualization.

Static visualization of malware entails representing the content of malicious software, either in the form of hexadecimal binary code or source code written in assembly format, as images without executing them. The hexadecimal view displays the machine code as a series of hexadecimal values. Analysts can examine and graphically represent various pieces of information within the machine code, such as instruction code data structures, starting at a memory address. Additionally, we can visualize the entropy information derived from the hexadecimal code. Conversely, the assembly language source code encompasses not only the symbolic machine code instructions but also details about memory allocation, function calls, and variables.

Dynamic-based visualization techniques involve the execution of files in a sandbox environment, such as the Cuckoo sandbox, to extract relevant features representing malware behavior. These features are then processed to generate grayscale or color images based on the experimental requirements. However, compared to static visualization, we have observed a relatively limited amount of effort dedicated to dynamic visualization approaches (see Table 4). Static visualizations lack real-time and interactive features and cannot depict temporal changes. Dynamic visualization techniques are essential to capture the evolving patterns and trends that unfold over time in malware utilizing polymorphic and obfuscation techniques. Dynamic features such as API calls [223], memory dumps [29], network traffic [230] [191], system calls [35], are widely used for malware visualization.

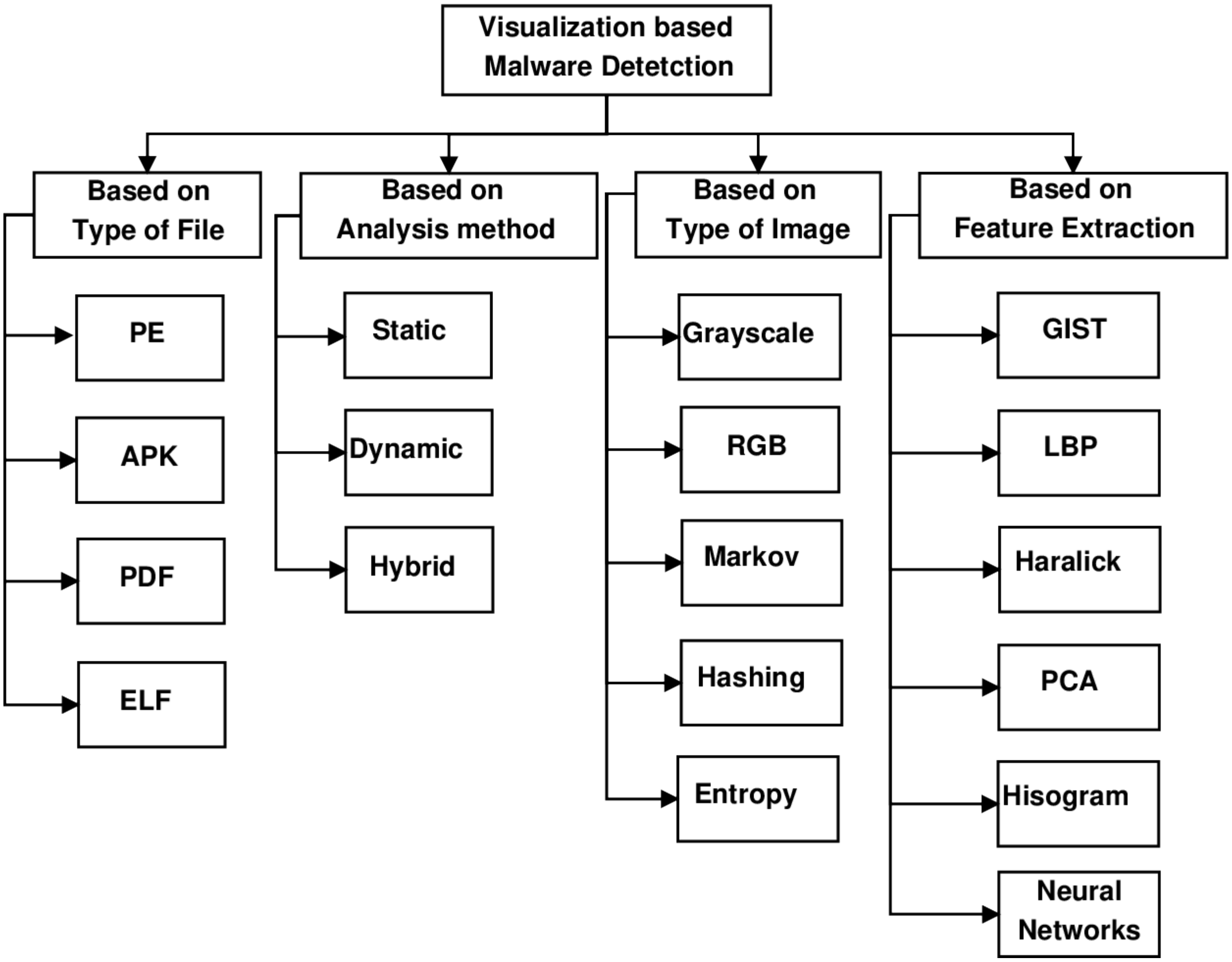

Figure 4: Taxonomy of visualization-based malware detection

Hybrid visualization aims to harness the advantages of static and dynamic techniques. Static features are adept at capturing the structural organization and composition of the underlying binary, thereby uncovering commonalities among different malware variants. Conversely, researchers specifically consider dynamic features to model the behavioral aspects of malware. Within the framework of hybrid visualization, these two feature types can be amalgamated and consolidated into a unified image [95] [161] [38] [109]. In particular, Huang et al. [95] used static analysis to generate an RGB representation of the static code using the byte sequence. Then, they used dynamic analysis to extract the API call sequence of the samples and generated malware images by assigning RGB values to each call. Finally, one can produce a hybrid image for classification by placing the dynamic and static representations one below the other.

We can visualize malware files as images using various techniques, including *(i)* raw bytes, *(ii)* meta-data information, or *(iii)* runtime data. In the following, we will dive into the details of these visualization approaches.

Raw Bytecode Images. Malware often appears as executable files[205]. Raw byte visualization converts these files into images by mapping their binary content, enabling analysis of structure, obfuscation patterns, and hidden data. This technique helps reveal embedded functionality and highlights similarities or differences between malware samples. Several studies have investigated the conversion of raw bytes from executable formats (e.g., EXE, APK, ELF) to grayscale or RGB images for visualization-based malware detection [157, 229, 173]. Nataraj et al. [157] proposed a technique known as grayscale image conversion for visualization-based malware detection. This approach has gained widespread adoption in academic research for malware detection [142, 46, 132]. They generate grayscale images by utilizing the byte values from the binary representation of executables or applications, mapping them to a range of 0 to 255, representing various levels of gray. These images consist exclusively of shades of gray, with black representing the lowest intensity and white representing the highest intensity. Many researchers divide binary executables into 8-bit substrings and convert each substring into a 1-D vector of decimal numbers, regardless of the image type or color map, before generating the grayscale image. However, some researchers have chosen different substring lengths, such as 16-bit [147], or calculated fixed lengths [115], based on their specific requirements. Since the authors might not have access to the samples in binary form, they need to adapt the algorithm to transform the malware into an image to suit the specific format of the samples. The B2IMG conversion algorithm offers a variant of the standard bytecode image technique, suitable for cases where we use the sample's hex dump [204]. We can apply a similar procedure with opcodes instead of directly on the binaries by interpreting opcodes as integers and *selecting only the most common ones* [101] [76] [129] [200]. The values at each position can then be interpreted as bytes and transformed into grayscale images. Qiao et al. [171] presented another method that utilizes the Word2Vec model to compute a fixed-length vector representation for each assembly instruction in every malware sample. The word vectors are combined, generating a matrix of values for each sample, which can be interpreted as a grayscale image. Although researchers widely use and find it relatively easy to implement the bytecode method, which makes it the standard technique in most research papers. However, they often limit its utilization due to the potential violation of the *locality principle* during the translation from binary to image. Directly converting non-image data to pixel values may fail to preserve the spatial relationships or visual coherence, leading to inaccuracies in representing the original data distribution within the resulting images. Advanced methods employ techniques such as hashing[159] and space-filling curves [21] to tackle this issue. By large, the bytecode method excels in its versatility, as it can process different types of information from a malware file, regardless of whether it is dynamic or static. It processes the information as a sequence of bytes, transforming it into a visual image for effective visualization.

Metadata Information Images. The metadata of malware files encompasses supplementary information linked to the file, which extends beyond the binary code of the malware. Metadata can help analyze and understand malware because it provides additional context and information that can help identify and classify the malware. Various types of metadata information can be used in different ways to create visual elements that aid malware analysis. For example, metadata from the *Portable Executable (PE) header*, such as the address of entry points, sections, and optional header details, can be transformed into visual elements to help understand the structural aspects of the files. The PE header contains valuable content, including the file structure description and other information that researchers can exploit to identify and classify malware. Moreira et al. [149] used this technique to classify 25 ransomware families. In their paper, the authors used the raw PE header without assuming the length of the fields or requiring expert knowledge to extract features. The PE header has a fixed length of 1024 bytes, which researchers can easily interpret as a 32×32 matrix with the application of an SFC. Researchers have also experimented with the use of boundaries on executables sections. In particular, Xiao et al. [228] used the section seizes extracted from the malware PE header to mark the limits of each section with a different color to facilitate visual distinction between each part. Metadata related to *imported and exported functions or modules*, including their relationships and dependencies, can also be visualized to provide insights into the code's functionality and the interactions between different components. Metadata pertaining to *digital signatures* can be overlaid on the images of the binaries to visually depict the trustworthiness or authenticity of the file, aiding in the identification of potentially malicious executables. In addition, metadata like *file size*, *creation date*, and *file extensions* can be encoded as visual features and incorporated into the visualization to provide additional context and information about malware samples. The technique of utilizing metadata for malware visualization is particularly prevalent in analyzing Android malware, owing to the abundance of useful metadata information available in the APK files. Application signatures, certificates, and information in the `AndroidManifest.xml` file serve as valuable metadata sources for the analysis and comprehension of Android malware. In [20], Bakir et al. use the `classes.dex` file as the basis for their analysis. This file can be read as an integer array and resized to form a 2-D matrix. Singh et al. [192] use the `classes.dex`, `resources.arsc`, `AndroidManifest.xml` files, and the content of the `META-INF` folder in order to generate the visualization representation of the malware. These files are interpreted as an array of bytes, resized in a matrix, and combined to represent the final 2-D grayscale image. Metadata information provides crucial details that aid researchers in understanding the characteristics and behaviors of Android malware.

Run-time Information for Image Generation. Another approach for generating an image from malicious binaries involves gathering features obtained through behavioral or runtime analysis. This method executes the malware within a secure environment or sandbox and uses the collected data to create images. This data includes network communications logs, file access, API calls, return values, call times, call process numbers, system calls, resource consumption, registry keys, memory artifacts, and more. The most commonly adopted technique is to monitor system calls [223]. Different approaches exist in terms of interpreting the extracted information. For instance, [195] assigns a hexadecimal code to each extracted API call, resulting in a byte sequence that can interpreted as a grayscale image. Conversely, [76] models the image as a matrix, where each value corresponds to an API call, and it assigns a value of 0 if the malware does not use the API call or 1 if it does. Another method to interpret dynamically extracted information as images is achieved by substituting the dynamic analysis report with the malware sample's original binary and generating a grayscale image from that [140]. In this case, the report file itself is used instead of the malware file to represent the sample in further analysis steps. The dynamic analysis data is gathered from `.json` files and converted into images

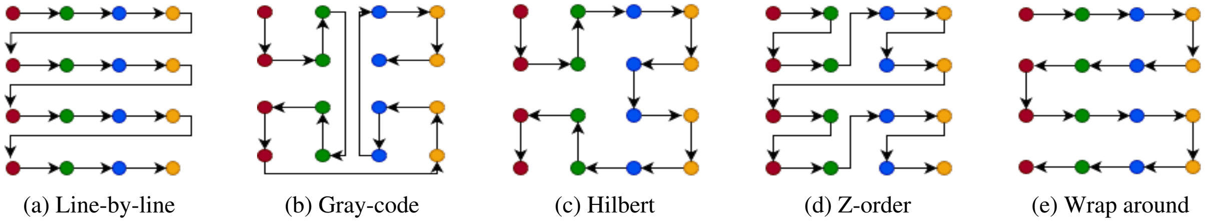

Figure 5: The figure shows a representation of the most common SFC used in the literature. The line-by-line method is the most common way grayscale bytecode images are generated.

using the standard grayscale method. This way, the techniques usually applied on the binary file for extracting static features can be applied on a `.json` file that contains the representation of dynamic information.

In their study, Bozkir et al. [29] employed memory forensics techniques to extract temporary data from the computer's physical and virtual memory. They utilized tools such as "Procdump" (reference missing) to dump this data while executing the malware sample in a virtual environment. Due to the substantial file sizes produced by this method, researchers adopted RGB images. In this approach, each image pixel represents three sequential bytes, allowing for the inclusion of color, in contrast to the traditional grayscale method that relies on bytecode. The approach used by Xu et al. [230] involves the use of network-captured packets as the foundation for image generation. Concatenating the captured PCAP files for each sample, researchers interpret their bytes as a grayscale image. Agrafiotis et al. [3] have more recently explored using PCAP files in conjunction with transformers.

In summary, dynamic information visualization has excellent potential for malware analysts [203]. However, most of the research focuses on the static feature-based image generation technique (see Table 4. The emphasis on static feature-based image generation arises from the challenges and diverse approaches associated with dynamic analysis, which become more complex with larger datasets, making testing challenging. Additional experimentation is needed to explore the real potential of dynamic analysis and compare it with more established static analysis techniques.

5.2.2 Image Representation

This section elucidates the methodologies employed in existing literature to convert extracted dataset information into visual images. The utmost importance lies in choosing an optimal technique for image representation because it defines how to organize the data visually. These approaches represent data in a visual format, maintaining essential features such as relational structures and interconnections. Researchers must balance creating an effective image layout for feature extractors and classifiers while minimizing information loss during data transformation.

Space Filling Curves. The conventional bytecode method for translating a binary file into an image leads to information loss, including a partial degradation of locality. This implies that arranging bytes in rows will cause line breaks to separate bytes originally adjacent in the code. In an attempt to address this issue, researchers have conducted experiments on the byte layout of the image, which refers to the order in which the pixels are generated and arranged within the image. Several studies, including those by Rustam et al.[183], Tekerek et al.[204], Qiu et al.[172], Pratama et al.[169], and Chaganti et al. [37], discuss the most commonly used and straightforward approach, known as the "line-by-line" or "carriage return" method. In this method, pixels are arranged in a specific order, placing each pixel after the previous one until reaching the preferred line width. Subsequently, the next pixel moves to the start of the following line. However, this method overlooks the issue of locality.

In addition, researchers have explored the utilization of various Space-Filling Curves (SFC) to generate grayscale images. Space-filling curves are mathematical constructs that traverse and fill a given space continuously. In malware visualization, one can use these curves to fill a matrix of dimensions $N \times M$, where each pixel represents a value derived from the original malware sample. This approach allows for the generation of the corresponding image of the sample.

In [161], O'Shaughnessy et al. use three different SFC techniques: Z-order, Gray-code, and Hilbert curves. Similarly, in the study conducted by the authors in [221], the wrap-around and H-curve approaches are utilized in addition to the standard line-by-line method. Hashemi et al. [90] also adopt the wrap-around method, among others, to generate the images. Another study by authors in [21] leverages the website binvis.io [49] to generate the images. This website offers various ways to represent ASCII printable characters by assigning each a distinct color and arranging the pixels using an SFC of choice, including line-by-line, Z-order, and Hilbert. The Figure5 illustrates the most commonly used SFC in the literature.

Markov Images. In various studies, [61, 39, 168, 90] researchers explored the assumption that the order of instructions in an executable file is not random but based on the locality principle. Their experiments employed Markov chains to generate information-rich images, recognizing that the preceding attribute influences each attribute (bytes, mnemonics, opcodes, etc.) in the file. To represent files visually, one approach is to treat each byte in the byte stream as a state, with 256 possible states for each element. By assuming that the next state depends only on the current state, we can conceptualize the byte sequence as a Markov chain, such that $P(s_{i+1}|s_0, ..., s_i) = P(s_{i+1}|s_i)$ if s_i represents the attribute in position i. The formula in Equation 1 expresses the probability of transitioning from state m to the subsequent state n.

$$P_{m,n} = P(n|m) = \frac{f(m,n)}{\sum_{n=0}^{256} f(m,n)} \tag{1}$$

Where $f(m, n)$ is the frequency of the transition from state m to state n. In contrast, the denominator $\sum_{n=0}^{256} f(m, n)$ signifies the sum of frequencies for all possible transitions from state m. This approach allows for the extraction of a matrix of dimensions 256×256. The resulting matrix, often called a Markov image, encapsulates the transition probabilities and can visually represent the underlying malware characteristics.

Hashing Schemes. Researchers have used different hashing algorithms to generate images, allowing them to extract locality-sensitive information and combine local and visualization data in feature images. One particular algorithm, "Minhash", can create a concise visual summary or fingerprint of malware samples. Minhash [31] is a prominent hashing algorithm employed for estimating the similarity between sets. It finds extensive utilization across diverse domains, including document similarity analysis, recommendation systems, and clustering. In the context of malware visualization, we represent each sample using a set of opcodes or opcode groups, applying the minhash hashing function to them. This method produces a fixed-size image that can be used in the following steps of the pipeline. In malware visualization, researchers extract relevant attributes or characteristics from the samples, such as opcode sequences, API calls, byte-level n-grams, or other suitable representations of the malware file. Next, researchers extract specific elements(i.e., shingles) of a fixed length contiguous subsequences and apply the hashing algorithm to the resulting set. This procedure culminates in the creation of grayscale images [200]. Charikar et al. [40] presented *Simhash* as a locality-sensitive hashing scheme primarily used for recognizing similar text and identifying duplicated web pages. The idea behind *Simhash* is the use of rounding algorithms to maintain the locality principle while hashing. A rounding algorithm converts numerical values to a specified level of precision or rounding interval. This procedure maintains locality in its output by ensuring that nearby values in the input remain close to each other in the rounded result, preserving the proximity of data. Common hashing algorithms have the desirable property of low collision rate, *Simhash*, similar to *Minhash*, instead aiming to make the hashes of two similar inputs similar themselves. In [159], [57] and [61]. The technique of *Simhash* involves representing a document as a vector with a specified number of bits. Each bit in the vector corresponds to a hash value calculated from the document's keywords. To be more specific, the value of the n-th bit of the vector is determined by computing the hash value of all the keywords present in the document. We can determine the value of the n-th bit as 1 or 0 based on whether the count of hash values with the n-th bit (for all keywords) set to 1 is greater than the count of hash values with the n-th bit set to 0 or not.

Entropy Images. Entropy is a concept derived from information theory and statistics [187]. It measures the randomness or uncertainty of a system or data. Experts in malware analysis primarily utilize it to study the techniques employed by malware authors to obfuscate or conceal their malicious intentions. Calculating entropy values involves dividing malware binaries into segments [72] and then determining the entropy value for each segment. Subsequently, these entropy values are converted into pixels to create an image representation. Another approach, proposed by [253], [227] and [39], involves using entropy graphs where the entropy of the entire binary serves as the image itself. This method utilizes entropy time series to capture the content of local discriminative features in each file. Employing this approach proves beneficial for the classifier in accurately distinguishing the samples into their respective malware classes.

Colored Images. Researchers have recently embraced the use of RGB and other colored images in malware detection due to their ability to convey more texture information compared to grayscale images, resulting in improved detection rates [154, 101, 61]. However, the true advantage of RGB images lies in their capability to incorporate different representations within the three color channels. There are two primary approaches to generating RGB images. The first approach involves utilizing the three RGB channels to represent three distinct characteristics [231], while the second approach involves directly extracting the three channels from the attributes of malware binaries [211]. In the first case, we create three grayscale images using different techniques, and we use their output as each of the three channels of RGB [61, 168]. For example, in [48], the authors introduce a novel image variant referred to as the GEM image. It is a 3-channel image combining the gray-level matrix, Markov, and entropy graph images.

Authors can utilize a combination of dynamic and static data to create more accurate representations of malware and its unique characteristics in generated images. Another approach involves reducing image dimensions and overlaying various binary sections, as described in [67]. Although this approach does not provide additional information to feature extractors and classifiers using RGB images, it can be useful for resizing images to a fixed dimension. In [211], researchers utilized a color map to convert grayscale images into colored images, which improved classifier accuracy. Additionally, researchers have used RGB images to incorporate specific information extracted from executable headers. For instance, Xiao et al. [228] used different colors to highlight the PE sections, improving classifier performance.

5.3 Feature Processing Methods

This section describes the data preprocessing phase utilized in the literature. This phase entails crafting a more manageable and informative dataset suitable for training and evaluating the models. Here, we examine the techniques for *(i)* extracting, *(ii)* selecting, and *(ii)* reducing features' dimensionality. Table 4 describes for each analyzed paper,

5.3.1 Feature Extractors

This section addresses the use of different feature extractors usually employed in combination with Machine Learning techniques. The algorithms presented here provide an explainable methodology to extract valuable characteristics from images and extrapolate discriminating features. In the following section we will analyze the literature related to Gabor-based Image Segmentation Technique (GIST), Local Binary Pattern method (LBP), Scale-Invariant Feature Transform (SIFT), Gray-Level Co-occurrence Matrix (GLCM), Histogram, Neural Network (NN), and the integration of more than one techniques.

Gabor-based Image Segmentation Technique (GIST). The Gabor-based Image Segmentation Technique (GIST) [84] was explicitly designed to segment and represent images by analyzing the spatial distribution of local spatial frequencies and orientations. Gabor filters imitate the functioning of the human visual cortex and consist of sinusoidal signals that detect texture variations in an image based on their frequency and orientation. These filters act as local bandpass filters, capturing localization properties in both the spatial and frequency domains. A set of Gabor filters with various orientations and frequencies ensures spatial localization, extracting precise features by applying them to the image. Gabor filters possess the capability to identify distinctive textural characteristics that differentiate malware from benign files or distinguish among different families of malware as state in [157, 11, 168, 188, 152]. Tuning the parameters of these filters requires careful attention, and selecting suitable values can be challenging. Moreover, since GIST features depend on the global structure of an image, an adversary with knowledge of the method could potentially evade detection by rearranging different sections of the code, as mentioned in [78, 132].

In [50] the authors effectively distinguished between benign and malicious PDF samples by transforming the files into images by utilizing byte and Markov plots. Subsequently, distinctive visual attributes of the images were extracted using Keypoint descriptors such as SIFT (Scale-Invariant Feature Transform), Oriented FAST, and Rotated BRIEF, along with texture features like LBP (Local Binary Patterns), Gabor Filter, and Local Entropy. Remarkably, they achieved an impressive F1-score of 99.48$\%$ in Random Forest (RF) classification by employing a byte plot with Gabor Filter.

Local Binary Pattern method (LBP). Researchers widely employ the Local Binary Pattern method (LBP) [160, 137, 90, 226, 151, 222] as a feature extractor in malware visualization approaches. This method generates a new LBP image by comparing neighboring pixels and assigning them binary values (0 or 1), resulting in an 8-bit binary array. Afterward, these arrays are converted into decimal values by traversing either clockwise or anti-clockwise. Finally, store these decimal values in the LBP image at their respective pixel locations. To calculate the LBP value for each center pixel (LBP_c), researchers utilize Equation 2.

$$LBP_c = \sum_{n=0}^{n-1} L(g_n - g_c))2^n \tag{2}$$

Where n is the number of neighborhood pixels, g_n and g_c are the gray levels of the neighborhood pixel and center pixel, respectively. We compute $L(x)$ by using Equation 3 and then combine the resulting LBP_c values to generate the LBP image.

$$L(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x >= 0 \end{cases} \tag{3}$$

In [206], the authors propose a novel method for malware recognition based on byte code. The method incorporates several steps, including feature extraction using local neighborhood binary pattern (LNBP), concatenation of features, feature selection utilizing Neighborhood Component Analysis (NCA), feature reduction through Principal Component Analysis (PCA), and classification using Linear Discriminant Analysis. The LBP feature is known for its stability and noise resistance but is limited to focusing solely on local features[74]. In [226], researchers have introduced an approach that utilizes Local Binary Patterns and Principal Component Analysis to detect Android malware. Notably, their method achieved an impressive accuracy rate of 90$\%$.

Scale-Invariant Feature Transform (SIFT). SIFT descriptors can increase detection accuracy [135]. This method extracts features by computing gradient magnitude histograms in eight orientations of bins. Splitting the image into small sections generates these bins. A sliding window executes the computations, computing the gradient histograms of each image locality. Cascaded connection functions obtain the final image feature descriptors[134, 152] [154]. To classify Android malware images, DeepVisDroid [18] utilizes four distinct image-based local features and three diverse image-based global features. These features are employed to train a lightweight 1d-convolutional Neural Network model. Recently, the author in [119] converted the malicious file into grayscale images to extract local and global textural features. They employed SIFT and other descriptors for local feature extraction. GIST and other methods for global feature extraction. The author also developed a BoVW algorithm to construct a local feature map by selecting low-dimensional features from the grayscale images. These feature maps were combined to train K-NN, SVM, RF, NB, and ExtraTree classifiers.

Gray-Level Co-occurrence Matrix (GLCM). Gray-Level Co-occurrence Matrix-based feature extraction (GLCM) [193] is a popular method to extract texture information from images. Extracting GLCM features involves considering the spatial relationship of distance and orientation between pixels. A GLCM matrix is created by estimating the probability density function of gray-level pairs with a predetermined spatial relationship, usually a distance of one in the four cardinal directions. From that, we calculate relevant statistics to represent the initial image. These statistics can vary from one research work to another. Table 4 contains, for each analyzed paper, the year of publication; the information used to create the image (i.e., [S] for static information, [M] for metadata, [D] for dynamic information); the technique used for the binary representation of the image (i.e, [LbyL] if the paper uses the classic line by line binary representation, [Others] if the paper uses other Space-Filling Curve (SFC) techniques to lay the pixel in the image other than LbyL); the image generation technique (i.e., [Markov] if the Markov technique is used to generate the images; [Hashing] if hashing schemes have been used to generate the images; [Entropy] if Entropy Analysis have been used to generate images; [Original] if the image generation technique employed are original); the information about the image color (i.e., [A] for colored images that use the color to add information, [B] for the colored images that do not add information); the used feature extraction technique (i.e, [G/GIST] for GIST or Gabor filters, [LBP] for Local Binary Pattern; [SIFT] for Scale-Invariant Feature Transform, [H/GLCM] for Gray-Level Co-occurrence Matrix or Haralick features, [Hist] if the paper uses histrograms of any type to extract features, [NN] - if the paper uses a Neural Network to extract feature. The specific architecture of the NN will be specified if possible and a Greek letter will indicate if [α] the model is trained from scratch, [β] transfer learning has been applied, or [γ] if fine-tuning has been applied, [Original] if the feature extraction technique is original); if the model uses a fusion technique. Karanja et al.

[108] also utilized GLCM to extract features from a grayscale image. In particular, researchers used only five of the proposed statistical measures of texture features to extract the relevant data. The selected features are Entropy, Contrast, Correlation, Angular Second Moment (ASM), and Inverse Differential Moment (IDM). The authors in [216] combines first-order and GLCM-based second-order statistical texture features with ensemble learning techniques to classify malware images. Recently, in [192], the authors selected nineteen statistics that measured with the previously described method, resulting in 76 features.

Haralick textures [88] evaluate image texture properties like coarseness, smoothness, and regularity using a co-occurrence matrix and equations for contrast, correlation, and angular moment. The authors in [4] conducted experiments that focused on global properties using Haralick features, but modern techniques have now surpassed them. Unver et al.[209] transformed Android apps into grayscale images and used Haralick texture and Hu Moments to achieve over 98% accuracy on malware classification. Karanja et al. [108] proposed an approach to classify IoT malware using Haralick image features and classifiers like naïve Bayes, K-Nearest Neighbor, and Random Forest. The method involves converting a binary file into a grayscale image, computing GLCM for each image, and deriving five Haralick features.

Histogram. The histogram captures the color distribution within an image by comprising multiple bins, each representing the frequency of pixels at specific intensity levels [19]. For grayscale images, histogram construction involves 255 bins, which correspond to intensity values ranging from 0 to 255. One can employ contrast-limited adaptive histogram equalization algorithms to enhance the local contrast of each region in a malware image and shift the focus from identifying similar codes to identifying similar image regions within a malware family. These algorithms adjust the pixel intensities within individual regions of the image, effectively improving the distinction between different regions [251]. The evaluation of VisMal demonstrates an impressive accuracy rate of 96%. Kumar et al. [116] demonstrate the effectiveness of the HOG feature descriptor in extracting texture statistics from grayscale images. The experiments highlighted the HOG descriptor's efficacy, achieving remarkable accuracy and outperforming the Gabor filter and LBP analysis techniques. The researchers in [29] endeavored to generate RGB images from memory dumps to capture visual patterns. They create feature vectors using GIST and HOG descriptors, which were subsequently classified using various classifiers. Notably, the SMO algorithm achieved an impressive accuracy of 96.39% when utilizing GIST and HOG feature vectors. Another similar approach, as discussed in [54], involves extracting a memory dump file and converting it into a grayscale image. Subsequently, features are extracted from the image using the histogram of gradients technique.

Neural Networks (NN). The use of Neural Networks as feature extractors in visualization-based malware detection is gaining popularity due to its ability to automate the analysis and categorization of malware images [211, 59, 227]. By training NN on a large dataset of malware images, they can automatically identify crucial characteristics indicative of malicious behavior or traits. These features encompass shapes, textures, visual patterns, and other visual properties that distinguish malware images. Convolutional Neural Networks, in particular, are capable of detecting code modifications resulting from code reallocation by learning spatially invariant patterns [78]. Hence, researchers often prefer the simplicity and utility of using NN as feature extractors. In the study [228], a fine-tuned VGG16 model, specifically the first five convolution blocks, was employed as the feature extractor for images. Binary texture analysis has shown improved accuracy and reduced time consumption, making it effective in addressing obfuscation issues. However, the computational cost of extracting complex texture features from malware images remains high [211]. In contrast, the authors in [105] utilized the full set of convolutional layers from the pre-trained VGG16 (prior to fully connected layers). This choice allows the model to utilize rich features without the loss of spatial resolution that would occur in fully connected layers. The study[241] utilizes a deep residual network (ResNet) as the primary model for feature extraction. ResNet is known for its capability to learn complex patterns in data due to its deep architecture and residual learning mechanism. The network can effectively capture both local and global features of malware images after the enhancement process, improving the overall accuracy of the detection model. One significant advantage of feature extraction through Neural Networks is that it does not require human intervention or description during the extraction process.

Feature Integration. Various feature extraction methodologies can ensure a more faithful representation of the original data. Authors have experimented with using local and global feature extractors to get information about the general features of the malware without losing focus on local peculiarities. Naeem et al. in [152] designed an original feature extractor, LGMP, by fusing D-SIFT as the local feature extractor and GIST as the global feature extractor. Their test demonstrated a significant improvement in the accuracy of the classifier when using LGMP instead of just local (5.4% increase) or just global (9.2% increase) feature extractors. Feature integration, which involves synthesizing local and global features, achieves a balance in the influence of local and global features using Gaussian weights. Several authors have also experimented with the fusion of features extracted from different CNNs [212] [76][64], showing sensible improvements from the best results of a single CNN. The most used technique in feature integration among different CNNs is the simple concatenation of the first flattened layer, usually with previous dimensionality reduction, to make the ensemble flattened layer manageable. Gibert et al. [76] specifically use different generated images to extract features from the CNNs, all originating from the same binary file. Meng et al. [145] integrates features through a deep Neural Network model that combines local and global information from these distinct perspectives, allowing for a more comprehensive analysis of app behaviors. By leveraging multi-view vectorial fusion, LensDroid enhances the detection capabilities by capturing high-level semantics that are not inherently linked. The authors of [238] proposed the SAC framework that integrates malware features from DEX files by collaboratively modeling image and graph data. It uses a task-oriented CNN (IFNeXt) for local image features and a dual-channel GCN for global bytecode structure, capturing both content and structural malware characteristics. Yu et al. [243] proposed a lossless feature extraction and integration method that transforms malicious code into semantically intact images. This technique preserves bytecode information and code text correlations through strategic pixel arrangement and interleaved encoding, mitigating semantic truncation. A multi-scale feature extraction module ensures uniform embedding of variable-length samples into fixed-size feature maps, capturing both local and global contextual features as long-text sequences within a matrix. By integrating these features, this approach overcomes information loss inherent in traditional resizing and cropping, enhancing malware classification accuracy.

Recent advances in malware detection have explored integrating multiple visual modalities to enhance classification performance. One prominent approach is presented in [105], they propose a multimodal deep learning fusion framework that leverages grayscale images, entropy graphs, and simhash images generated from malware binaries. Using separate VGG16 models trained on each modality, the authors explored feature-level fusion using common operators such as addition, average, maximum, and concatenation. This study highlights the effectiveness of fusion in malware detection and the critical role of selecting suitable fusion operators for robust multimodal learning.

5.3.2 Feature selection

An encoding-based technique is mentioned in [153] to reduce the dimensions of D-SIFT and GIST features. During the local feature reduction phase, they employed D-SIFT to identify interest points in the malware image and extract 128-dimensional features from each patch obtained from a dense grid of patches. Subsequently, they applied the Bag of Features (BOF) model to perform dimensionality reduction. The BOF model consisted of multiple steps, including identifying salient local points and creating a dictionary of local features using Fisher vector encoding.

Table 4: Techniques used to extract features from the starting malware.

Ref.	Year	Info	Binary Repr.	Image Gen. Technique	Color	Feature Extraction Technique	Fusion Technique
Fu et al. [72]	2018	S	LbyL	SimHash	✓	H/GLCM, Others	✗
Yakura et al. [233]	2018	S	LbyL	-	✓	NN, Others	✗
Yang et al. [236]	2018	S	LbyL	-	✗	NN	✗
Darus et al.[58]	2018	S	LbyL	-	✗	G/GIST	✗
Kumar et al. [117]	2018	S	LbyL	-	✗	NN	✗
Su et al. [198]	2018	S	LbyL	-	✗	NN	✗
Cui et al. [52]	2018	S	LbyL	-	✗	Original	✗
Ni et al. [159]	2018	S	✗	Simhash	✗	NN	✗
Shiva et al. [195]	2018	D		Original		Original	✗
Xiao et al. [229]	2019	S	LbyL	Original	✓	NN	✗
Vinayakumar et al. [217]	2019	S	LbyL	-	✗	NN, Others	✗
Liu et al. [133]	2019	S	LbyL	-	✗	G/GIST, NN	✗
Hsiao et al. [93]	2019	S	LbyL	Average Hash	✗	NN	✗
Gibert et al. [78]	2019	S	LbyL	-	✗	NN	✗
Khormali et al. [112]	2019	S	LbyL	-	✗	NN	✗
Naeem et al. [152]	2019	S	LbyL	-	✗	G/GIST, SIFT	✗
Liu et al. [134]	2019	S	LbyL	-	✗	LBP, SIFT	✗
Akarsh et al. [6]	2019	S	LbyL	-	✗	NN	✗
Venkatraman et al. [215]	2019	S	LbyL	Entropy	✗	NN	✗
Hashemi et al. [90]	2019	S	Others	Markov	✗	LBP	✗
Baptista et al. [21]	2019	S	Others	-	✓	Original	✗
Yang et al. [237]	2019	M,D	✗	Original	✗	Original	✗
Chen et al. [42]	2019	S		Original		NN	✗
Qiao et al. [171]	2019	S		Original		NN	✗
Mercaldo et al. [146]	2020	S	LbyL	-		NN(FFNN)	✗
Lekssays et al. [127]	2020	S	LbyL	G/GIST		NN	✗
Naeem et al. [154]	2020	S	LbyL	-	✓	NN	✗
Vasan et al. [212]	2020	S	LbyL	-		NN	✗
Liu et al. [132]	2020	S	LbyL	-	✗	NN	✗
Karanja et al. [108]	2020	S	LbyL	-	✗	H/GLCM	✗
Xiao et al. [227]	2020	S	✗	Entropy	✓	NN	✗
Jain et al. [100]	2020	S	LbyL	Original		NN	✗
Vasan et al. [211]	2020	S	LbyL	-	✓	NN	
Zhao et al. [250]	2020	S	LbyL	-	✓	NN	✗
Roseline et al. [180]	2020	S	LbyL	-		Original	✗
Go et al. [80]	2020	S	LbyL	-		NN	
Verma et al. [216]	2020	S	LbyL	-		H/GLCM, Hist	✗
Ren et al. [176]	2020	S	LbyL	Original		NN	✗
Vu et al. [221]	2020	S	Others	Entropy	✓	H/GLCM, NN	✗
Girbert et al. [76]	2020	S	LbyL	-		NN, Others	✗
Iadarola et al. [97]	2021	S	LbyL	-		NN	✗
Singh et al. [192]	2021	S	LbyL	-		G/GIST, LBP, H/GLCM, NN	✗
Dib et al. [64]	2021	S	LbyL	-		NN (CNNα)	✓
Darem et al. [57]	2021	S	LbyL	Original		NN, Others	✗
Sun et al. [200]	2021	S	✗	MinHash	✗	✗	✗
Bozkir et al. [29]	2021	D	LbyL	-	✓	G/GIST, Hist	✗
Pinhero et al. [168]	2021	S	LbyL	Markov, Entropy	✓	G/GIST, NN	✗
Bagane et al. [15]	2021	S	LbyL	-		NN	✗
Li et al. [129]	2021	S	LbyL	-	✓	NN (CNNα)	✗
Acharya et al. [2]	2021	S	LbyL	-		NN (EfficientNet- B1β)	✗
Sudhakar et al. [199]	2021	S	LbyL	-		NN (MCFT-CNNγ)	✗
Anandhi et al. [11]	2021	S		Markov	B	G/GIST, NN (VGG3γ, DenseNetγ)	✗
Xiao et al. [228]	2021	S,M	LbyL	-	A	NN (VGG16γ)	✗
Wang et al. [223]	2021	D	LbyL	-		N (BiLSTM)	✗
Jian et al. [101]	2021	S	LbyL	-	A	NN (SEResNet50α, BiLSTM, Attention)	✓
Bouchaib et al. [26]	2021	S	LbyL	-		NN (VGG16γ)	✗
Zhu et al. [253]	2022	S	✗	Entropy	✗	✗	✗
Chai et al. [39]	2022	S	✗	Entropy	✗	✗	✗
Dharmalaksana et al. [63]	2022	S	LbyL	-	✓	NN	✗
Tekerek et al. [204]	2022	S	LbyL	-	✓	NN	✗
Bensaoud et al. [24]	2022	S	LbyL	-	✓	NN	✗
Wang et al. [222]	2022	S	LbyL	-		G/GIST, LBP, NN (ResNet50β)	✓
Olorunjube et al. [67]	2022	S	LbyL	Original	B	NN	✗
O'Shaughnessy et al. [161]	2022	S,D	Others	-	✓	G/GIST, LBP, Hist	✗
Zhong et al. [251]	2022	S	LbyL	-		Hist, NN (CNNα), Other	✗
Kumar et al. [118]	2022	S	LbyL	-		NN (VGG16β, VGG19β, Inception V3β, ResNet50β)	✗
Mallik et al. [143]	2022	S	LbyL	-		NN (VGG16, BiLSTM)	VGG16-BiLSTM
Conti et al. [48]	2022	S	LbyL	Markov, Entropy, GEM Image		H/GLCM, NN (Shallow-CNN, Siamese)	GEM
Paardekooper et al. [162]	2022	S	LbyL	-		NN (Fast-CNN GA)	✗
Girbert et al. [79]	2022	S,M	LbyL	Entropy		LBP, H/GLCM, NN (CNN)	✗

Ref.	Year	Info	Binary Repr.	Image Gen. Technique	Color	Feature Extraction Technique	Fusion Technique
Zhu et al. [253]	2022	S	LbyL	Entropy		NN (Siamese, VGG16γ, VGG16β, Inception V3γ, Xceptionγ, DNN, RNN)	✗
Vinayakumar et al. [174]	2022	S	LbyL	-	B	NN (EfficientNetγ, various pre-trained networks)	Features from EfficientNet B0 to B7
Chai et al. [39]	2022	S	LbyL	Markov, Original	A	NN (CNN), Others	✗
Chaganti et al. [37]	2022	S	LbyL	-		NN (EfficientNet, B1γ)	✗
Pratama et al. [169]	2022	S	LbyL	-	B	NN (EfficientNet, B0-B7γ)	✗
Qui et al. [172]	2022	S	LbyL	-		NN (ShuffleNetα)	✗
Ma et al. [140]	2022	S,D	LbyL	-		NN (CNNα)	✗
Huaxin et al. [61]	2023	S	LbyL, Others	Simhash	A	NN (custom-CNNα)	✗
Hashemi et al. [91]	2023	S	Others	Original	A	NN (AlexNetβ), Others	✗
Rustam et al. [183]	2023	S	LbyL	-		NN (VGG16β, ResNet50β)	VGG16 + ResNet50
Kim et al. [113]	2023	S	LbyL	Entropy, Original		NN (CNN α)	✓
Lee et al. [125]	2023	S		Original		Original	✗
Karbab et al. [109]	2023	S,D	LbyL	Original	-	NN (HNN), Others	✗
Chaganti et al. [38]	2023	S,D		-		NN (CNNα), Others	✓
Guo et al. [86]	2023	S,D		-		NN (CNN)	✗
Jiang et al. [102]	2023	S	LbyL	-		NN (Stripe Pooling-CNNα, Pyramid Stripe Pooling-CNNα)	✗
Shaukat et al. [190]	2023	S	LbyL	-	B	NN (15 CNNsγ)	✗
Moreira et al. [149]	2023	S,M	LbyL, Others	-	B	NN (Xceptionα)	✗
Zhan et al. [245]	2023	S	LbyL	-	✗	LeNet, VGGNet, ResNet, DenseNet, SqueezeNet	✗
Dhanya et al. [62]	2023	S	others	Markov	✗	NN	✗
Xuan et al. [231]	2024	S	LbyL	Original	A	NN	Multi-Feature Fusion
Vasan et al. [213]	2024	S	Others	Original	✗	Original	✗
Sharma et al. [189]	2024	S	Lbyl	Original	A	NN	✗
Kumar et al. [120]	2024	S	Others	Original	B	NN(CNN)	✓
Yang et al. [241]	2025	S	LbyL	Original	A	NN(ResNet)	✗
Johnny et al.[105]	2025	S	Others	Original	A	NN(VGG16)	✓
Meng et al. [145]	2025	S,D,M	Others	Original	A	NN(CNN)	✓
Yang et al.[238]	2025	S	Others	Original	A	NN(CNN))	✓
Ambekar et al.[9]	2025	S	Others	Original	A	NN(Siamese Network)	✗
Alam et al.[7]	2025	S	LbyL	Original	A	NN(Sp-CNN)	✗
Zhang et al.[247]	2025	S	Others	Original	A	GIST,NN(CNN)	✗
Yu et al.[243]	2025	S	Others	Original	B	Original	✓

They accomplished Fisher vector encoding by training a Gaussian mixture model (GMM) to estimate the Fisher vector of dimension 2DK for D-SIFT features. Additionally, they employed PCA to further optimize the results by reducing the dimensionality of the feature vector to 100. In the second stage, they extracted global features from the malware image using the GIST feature description and reduced the dimension of these global features to 256.

5.3.3 Feature Dimensionality Reduction

Feature reduction plays a crucial role in image-based approaches, specifically in the context of image-based malware detection. It encompasses reducing the dimensionality of image data while retaining pertinent information. This section will explore the feature-reduction and selection techniques in image-based malware detection approaches.

Principal Component Analysis (PCA) [92] is a statistical technique that can extract texture from an image through dimensionality reduction. When applying PCA to an image, it aims to identify a set of orthogonal axes, termed principal components, that effectively represent the highest variation in the image data. Consider these fundamental components as representative patterns of texture. The first principal component captures the direction of maximum variance, while subsequent components record decreasing variance. [212] provides another example of dimensionality reduction. The authors apply PCA to reduce the features considered in their ensemble model. Once the features are extracted from the fine-tuned ResNet50 and VGG16 models, they were further reduced from 2048 to 205 and 6144 to 615, respectively. The resulting feature vectors from both models retained the component that explains 90% of the data variation. Consequently, by employing this technique, the feature vector of the ensemble model was condensed to a mere 815 features.

PCA is utilized in the study by Naeem et al. [152] to extract the most pertinent features obtained through GIST. Similar to the approach in Vasan et al. [212], the authors employ PCA to merge the remaining features with others extracted using alternative methods. This capability of reducing the feature count without sacrificing representation power facilitates researchers in calibrating the number of features required for their experiments, thereby simplifying the combination of different techniques. Additionally, in the work by Karbab et al.[109], PCA is directly applied to the generated image to reduce dimensionality prior to the feature extraction step. The authors in [213] introduce IMCBL, a malware classification method that transforms malware binaries into grayscale images and applies Truncated Singular Value Decomposition (SVD) to reduce dimensionality, improving computational efficiency while preserving essential features. The article [153] introduces an encoding-based approach to decrease the dimensions of D-SIFT and GIST features. Firstly, they used D-SIFT to identify interest points in the malware image. The Bag of Features (BOF) model was used for dimensionality reduction, followed by PCA to reduce the feature vector to 100 dimensions. In the second stage, global features are extracted from the malware image using GIST.

5.4 Classification

In this section, we explore the classification models presented by researchers for visualization-based techniques in malware detection and classification. The analysis primarily focuses on two key aspects of modeling techniques found in current literature: *(i)* Conventional ML-based approaches and *(ii)* DL-based approaches. Furthermore, we discuss work related to transformers and attention models, model fusion, data augmentation, and Few-shot Learning.

Table 5 presents a summary of the state-of-the-art literature about classification models and explored methodologies during the classification phase. Each entry represents the primary contribution of the paper. The columns denote the following information: the paper reference; if the classification type is malware detection [D] or classification into families [C]); the type of classifier (i.e., [DT] for Decision Tree, [RF] for Random Forest, [KNN] for K-Nearest Neighbors, [SVM] for Support Vector Machine, [CNN] for Convolutional Neural Network classifier and specific architecture if applicable, [RNN] for Recurrent Neural Network classifier and specific architecture if available, [Other ML] for other Machine Learning techniques, [Other DL] other Deep Learning techniques; the split notation, that is the proportion of data allocated to training, validation, and testing, or the indication of k-fold cross-validation or of leave-one-out; if there are some consideration of explainability and interpretability issues (identified by “Explain.”); if there is the use of data augmentation and specified architecture (identified by “Aug”); if there is an exploration of adversarial attacks, obfuscation, or packing (identified by “Adv”); if there is an exploration of few-shot learning; if the paper considers sustainability and concept drift issues (identified by “Sust.”); if the model used is an ensemble of classifiers(identified by "Ensemble").

5.4.1 Conventional Machine Learning Classifiers

Machine Learning algorithms play a vital role in malware image visualization by classifying and labeling various categories of malware based on their visual characteristics. These algorithms utilize image recognition techniques to identify distinct patterns and features specific to each malware type, achieving high accuracy by being paired with proper feature extraction techniques and training on extensive datasets of malware images. Below, we briefly introduce some of the algorithms widely employed in a substantial body of literature. One important advantage of classic ML approaches is that they are easy to implement and inherently explainable.

The most commonly used Machine Learning classifiers in the literature are KNN, SVM, and Random Forests. The K-nearest neighbor (KNN) model uses proximity to predict sample classes. The algorithm identifies the k-labeled data points in the sample space that are closest to the example to be classified. If the majority of these k-nearest instances belong to a specific category, then the data sample to be classified is also assigned to that category. The algorithm possesses specific advantageous characteristics. Firstly, it does not assume any specific distribution of the underlying data, making it highly flexible and adaptable to various datasets. Secondly, KNN does not require a dedicated training phase. Instead, it compares new instances with the labeled instances in the training dataset during the prediction phase, enabling quick and efficient decision-making. Another key strength of KNN is its robustness to outliers. Unlike linear models and other algorithms, KNN’s decision boundaries are less affected by outliers. This resilience arises from the fact that class assignment in KNN is determined by the majority of neighbors, minimizing the impact of individual instances and preserving the overall integrity of the predictions. KNN models have been the classifier of choice for numerous models and serve as a benchmark technique against more complex classifiers [72, 237, 152]. Machine Learning algorithms like SVMs can perform regression or classification tasks. These algorithms utilize an n-dimensional space to classify samples, representing each data object as a point with n attribute features. The categorization process involves identifying the hyperplane that effectively separates the class labels, enabling accurate classification. SVM performs well in high-dimensional spaces, making it suitable for problems with many features preventing overfitting. A negative characteristic of SVM, compared to other classifiers, like KNNs, is that they are computationally expensive, especially with large datasets. The use of SVMs has found some success in the past but has been largely overshadowed by better-performing machine-learning algorithms. Despite this, it has been used in recent papers as a benchmark but, more importantly, as an integral part of hybrid models based mainly on Deep Learning and CNNs [212] with great success, achieving better results than the softmax classifiers of the standard CNNs. Decision Trees (DT) and Random Forests (RF) have been among the best-performing Machine Learning algorithms across different feature extractors and datasets [72, 237]. Roseline et al. [180] demonstrated that a variation of the Random Forest model, known as Completely Random Forest (CRF), outperforms the classic RF model. Random Forest and Random Trees are known for their strong predictive power because they harness the combined knowledge of multiple decision trees, resulting in more accurate predictions than individual trees. Their ability to handle high-dimensional data stems from their feature subsampling technique, which allows them to focus on relevant attributes while mitigating the curse of dimensionality. However, the computational expense arises from training multiple trees, and the interpretability trade-off occurs in Random Forests as the ensemble nature makes it harder to attribute decisions to individual features. Traditional machine-learning classifiers have been dominating the field of malware image visualization. However, a shift in preference increasingly relegates them to a supporting role in favor of Deep Learning models. Deep learning models typically outperform Machine Learning models when resource constraints are not a limiting factor. In particular, the advancements in hardware have been a big help in launching Deep Learning models among researchers. Another advantage is that Deep Learning allows for end-to-end learning, where the model learns to extract relevant features and make predictions without relying on manual feature engineering. This eliminates the need for domain-specific knowledge and simplifies the overall workflow. With a focus on visualization, the domain often involves working with large-scale datasets, such as image collections. Deep learning excels at processing and learning from massive amounts of data, leveraging techniques like mini-batch training and parallel computing to handle such datasets efficiently.

5.4.2 Deep Learning

Recently, interest in Deep Learning and malware image-based classification techniques has grown significantly. This is because these approaches can alleviate the necessity for extensive feature engineering tasks, typically required in Machine Learning-based malware detection methods. This streamlines the classification process [244, 218]. In this section, we provide a summary of the significance of various designs, including convolutional Neural Networks (CNN), recurrent Neural Networks (RNN), long short-term memory (LSTM), and others, in addressing visualization-based malware detection or classification challenges.

Convolutional Neural Networks (CNNs). Convolutional Neural Networks (CNNs) are the most widely used architecture for malware image classification and general image-based analysis due to their effectiveness in processing spatially structured data. Introduced by LeCun et al. [53], CNNs utilize local connectivity and weight sharing to efficiently capture spatial correlations. Through convolutional and pooling layers, they learn hierarchical and discriminative feature representations, enabling robust performance in image recognition and classification tasks. A CNN typically consists of several layers, each serving a specific purpose in the learning process:

- **Convolutional Layer**: This layer applies a set of learnable filters to the input data, performing convolutional operations. It extracts local features and learns spatial hierarchies;

- **Activation Layer**: Typically following convolutional layers, activation functions (e.g., ReLU, sigmoid) introduce non-linearity, enabling the network to learn complex patterns and relationships;

- **Pooling Layer**: This layer downsamples the input, reducing spatial dimensions and summarizing information using operations like max or average pooling;

- **Fully Connected Layer**: Also called a dense layer, it connects all neurons between layers to learn global patterns in the extracted features;

- **Dropout Layer**: Dropout prevents overfitting by randomly setting a fraction of input units to zero during training, promoting robust feature learning;

- **Batch Normalization Layer**: This layer normalizes the activations of the previous layer, making the network more stable and accelerating convergence.

Researchers must tune hyperparameters, such as learning rate, batch size, epochs, optimizer, weight initialization, kernel size, filter count, and stride, to optimize CNN performance without overfitting or underfitting for a specific dataset.

Table 5: Summary of classification models and explored methodologies.

Ref.	Year	Classif. Type	Classifier	Split	Explain.	Aug	Adv	Few Shot	Sust.	Ensemble
Fu et al. [72]	2018	C	RF, KNN, SVM	90:10	✗	✓	✗	✗	✗	✗
Yakura et al. [233]	2018	C	CNN	5f	✓	✓	✓	✗	✗	✗
Yang et al. [236]	2018	C	CNN	50/-/50	✗	✗	✗	✗	✗	✗
Darus et al. [58]	2018	D	KNN, SVM	70:30	✗	✗	✗	✗	✗	✗
Kurmar et al. [117]	2018	D	CNN	90:10	✗	✗	✗	✗	✗	✗
Su et al. [198]	2018	C	CNN	-	✗	✗	✗	✗	✗	✗
Cui et al. [52]	2018	C	CNN	-	✗	✓	✗	✗	✗	✗
Ni et al. [159]	2018	C	KNN, SVM, CNN	80:20	✗	✗	✗	✗	✗	✓
Shiva et al. [195]	2018	D	CNN	10f	✗	✗	✗	✗	✗	✗
Xiao et al. [229]	2019	D	CNN	68:12:20	✗	✗	✗	✗	✗	✗
Vinayakumar et al. [217]	2019	C	RF, DT, KNN, SVM, CNN, RNN (LSTM), Other ML	10f	✗	✗	✓	✗	✗	✓
Liu et al. [133]	2019	D, C	RF, SVM, CNN	-	✗	✗	✓	✗	✗	✗
Hsiao et al. [93]	2019	C	CNN(Siamese), KNN	-	✗	✗	✗	Siamese	✗	✗
Girbert et al. [78]	2019	C	CNN	10f	✗	✗	✗	✗	✗	✗
Khormali et al. [112]	2019	D	CNN	-	✗	✗	✓	✗	✗	✗
Naeem et al. [152]	2019	C	KNN, SVM	10:90,20:80, 30:70,40:60, 50:50,60:40, 70:30,80:20	✗	✗	✗	✗	✗	✗
Liu et al. [134]	2019	C	RF, KNN	10f	✗	✗	✗	✗	✗	✗
Akarsh et al. [6]	2019	C	CNN, RNN (LSTM)	70:30	✗	✗	✗	✗	✗	✓
Venkatraman et al. [215]	2019	D, C	CNN, RNN (LSTM)	90:10	✗	✗	✗	✗	✗	✓
Hashemi et al. [90]	2019	D	KNN	10f	✗	✗	✗	✗	✗	✗
Baptista et al. [21]	2019	D	Other ML	-	✗	✗	✗	✗	✗	✗
Yang et al. [237]	2019	C	RF, KNN, Other ML	5f	✗	✗	✗	✗	✗	✗
Chen et al. [42]	2019	D	CNN (InceptionV3)	-	✗	✗	✓	✗	✗	✗
Qiao et al. [171]	2019	C	CNN (LeNet5)	50:50	✗	✓		✗	✗	✗
Liu et al. [132]	2020	D	RF, SVM, CNN	10f, 5f	✗	✗	✓	✗	✓	✗
Karanja et al. [108]	2020	D, C	RF, KNN	-	✗	✗	✗	✗	✗	✗
Xiao et al. [227]	2020	C	SVM, CNN	50:50	✗	✗	✗	✗	✗	✗
Mercaldo et al. [146]	2020	D, C	RF, DT, Other DL	50:50	✗	✗	✗	✗	✓	✗
Lekssays et al. [127]	2020	D, C	KNN, CNN (3 Conv+2 Dense)	-	✗	✗	✗	✗	✗	✗
Naeem et al. [154]	2020	D, C	CNN	70:30	✗	✗	✗	✗	✗	✗
Vasan et al. [212]	2020	C	CNN	70:30:-	✗	✗	✓	✗	✗	VGG16+ ResNet50
Jain et al. [100]	2020	C	CNN, ELM	-	✗	✗	✗	✗	✗	50 ELM models
Vasan et al. [211]	2020	C	Fine-tuned CNN	70:30	-	✓	✗	✗	✗	✗
Zhao et al. [250]	2020	C	Region-CNN	70:30	✗	✗	✗	✗	✗	✗
Roseline et al. [180]	2020	C	Deep RF	80:20:-	✗	✗	✗	✗	✗	Complete-RF
Go et al. [80]	2020	C	Other DL (ResNeXt)	10f	✗	✗	✗	✗	✗	✗
Verma et al. [216]	2020	C	RF	10f/ leave/ one out	✗	✗	✗	✗	✗	✗
Ren et al. [176]	2020	D	CNN (Dex-CNN), Other DL (DexCRNN)	80:10:10	✗	✗	✗	✗	✗	✗
Vu et al. [221]	2020	C	CNN, Other ML (XGBoost)	80:10:-	✗	✗	✗	✗	✗	✗
Gibert et al. [76]	2020	D, C	CNN	80:20	✗	✗	✗	✗	✗	✗
Iadarola et al. [97]	2021	C	CNN	80:20, 80:20:-	Grad-CAM	✗	✗	✗	✗	✗

Ref.	Year	Classif. Type	Classifier	Split	Explain.	Aug	Adv	Few Shot	Sust.	Ensemble
Singh et al. [192]	2021	C	Random Forest, KNN, SVM, CNN	-	✗	✗	✗	✗	✗	✗
Dib et al. [64]	2021	C	CNN,LSTM	10f	✗	✗	✗	✗	✗	✗
Bozkir et al. [29]	2021	D, C	RF, DT, SVM, Other ML	80:20:-, 3f	✗	✗	✗	✗	✗	✗
Sun et al. [200]	2021	C	CNN, RNN	-	✗	✗	✗	✗	✗	✗
Darem et al. [57]	2021	D	CNN, Other ML (XGBoost)	f	✗	✗	Obfuscation Detection	✗	✗	CNN, XGBoost
Pinhero et al. [168]	2021	D, C	CNN (VGG3, ResNet50)	70:30	✗	✗	✗	✗	✗	✗
Bagane et al. [15]	2021	C	CNN (LSTM, BiLSTM)	80:20	✗	✗	✗	✗	✗	✗
Li et al. [129]	2021	C	CNN (Attention, SPP)	80:20	✗	✗	✗	✗	✗	✗
Acharya et al. [2]	2021	C	CNN (EfficientNet-B1)	75:25	✗	✗	✗	✗	✗	✗
Sudhakar et al. [199]	2021	C	CNN (ResNet50)	75:25	✗	✗	✗	✗	✗	✗
Anandhi et al. [11]	2021	D, C	CNN (DenseNet201)	70:30	✗	✗	Additive Noise	✗	✗	✗
Xiao et al. [228]	2021	C	Linear-SVM, CNN (VGG16)	10f	✗	✗	✗	✗	✗	✗
Wang et al. [223]	2021	C	RNN (BiLSTM)	73:20:20, 62:20:20	✓	✗	✗	✓	✗	✗
Jian et al. [101]	2021	D	CNN (SEResNet50), RNN (BiLSTM)	80:10:10, 70:15:15, 60:20:20	✓	✓	✗	✗	✗	SEResNet50, BiLSTM
Bouchaib et al. [26]	2021	C	CNN (VGG16)	50:50	✗	✓	✗	✗	✗	✗
Zhu et al. [253]	2022	C	CNN (VGG16, Xception, InceptionV3)	80:20	✓	✗	✗	Siamese	✗	✗
Chai et al. [39]	2022	C	CNN, Other ML, Other DL	-	✗	✗	✗	✓	✗	✗
Dharmalaksana et al. [63]	2022	C	CNN	-	✗	✗	✗	✗	✗	✗
Tekerek et al. [204]	2022	C	CNN (DenseNet121)	80:20, 10f	✗	Cycle GAN	✗	✗	✗	✗
Bensaoud et al. [24]	2022	D, C	CNN	5f	✗	Cycle GAN	Obfuscation	✗	✗	✗
Wang et al. [222]	2022	C	Random Forest, Other ML	Stratified KFold	✗	✗	✗	✗	✗	MLP, RF, XGBoost
O'Shaughnessy et al. [161]	2022	C	RF, KNN, SVM	5f	✗	✗	✗	✗	✗	✗
Zhong et al. [251]	2022	C	CNN (3conv, 3FC)	10f	✗	✗	✗	✗	✗	✗
Gibert et al. [79]	2022	D	Other ML (XGBoost, GBT)	10f	✗	✗	✗	✗	✗	✗
Olorunjube et al. [67]	2022	D, C	RF, DT, KNN, SVM, CNN (ResNet50), Other ML (Extra Tree Classifier, XGBoost, Bagging, Naïve Bayes)	-	✗	✓	DGAN	✗	✗	✗
Kumar et al. [118]	2022	C	Random Forest, KNN, SVM, CNN (Fine-tuned VGG16 + FC, VGG19, Inception V3, ResNet50)	90:10	✗	✗	✗	✗	✗	✗
Mallik et al. [143]	2022	C	Decision Tree, KNN, RNN (BiLSTM), Other DL	80:20	✗	Rotation shifting flipping	✗	✗	✗	DCNN, BiLSTM
Conti et al. [48]	2022	C	Shallow CNN	70:30	✗	✗	✗	Siamese	✗	✗
Chaganti et al. [37]	2022	C	CNN (EfficientNetB1)	70:30	✓	✗	✗	✗	✗	✗
Pratama et al. [169]	2022	C	CNN (EfficientNetB7)	70:20	✗	✗	✗	✗	✗	✗
Zhu et al. [253]	2022	C	CNN (InceptionV3, ResNet50, VGG16, Xception)	80:20	t-SNE	✓	✗	Siamese	✗	✗
Paardekooper et al. [162]	2022	C	CNN (2Conv, 1 FC)	90:10	✗	✗	✗	✗	✗	✗
Chai et al. [39]	2022	C	CNN	65:20:20, few-shot	Ablation study	Random Rotation	✗	✓	✗	✗
Qiu et al. [172]	2022	C	CNN (ShuffleNet)	70:30	✗	✗	✗	✗	✗	✗
Ma et al. [140]	2022	C	CNN	80:20	✗	✗	✗	✗	✗	✗

Ref.	Year	Classif. Type	Classifier	Split	Explain.	Aug	Adv	Few Shot	Sust.	Ensemble
Vinayakumar et al. [174]	2022	D	CNN (Variants of DenseNet, EfficientNet, Inception, MobileNet, ResNet, VGG)	75:25	t-SNE	✗	✗	✗	✗	Stacked SVM, RF, LR
Hashemi et al. [91]	2023	D	CNN (AlexNet)	50:50	✗	✗	✗	✗	✗	✗
Huaxin et al. [61]	2023	C	CNN (MCTVD)	10f	✗	✗	✗	✗	✗	✗
Rustam et al. [183]	2023	C	RF, KNN, SVM, Other ML	80:20	✓	✗	✗	✗	✗	✗
Kim et al. [113]	2023	C	CNN	10f	✗	✗	✗	✗	✗	✗
Jiang et al. [102]	2023	D, C	CNN (VGG, Stripe Pooling-CNN, Pyramid Stripe Pooling-CNN)	80:20	✗	✗	✗	✗	✗	✗
Lee et al. [125]	2023	D, C	RF, DT, Other ML	-	✓	✗	✗	✗	✗	RF, MLP
Shaukat et al. [190]	2023	D, C	SVM	60:20:20, 10f	✗	✓	✗	✗	✗	✗
Guo et al. [86]	2023	C	CNN	80:20	✗	✗	✗	✗	✗	✗
Chaganti et al. [38]	2023	D	CNN (1conv+ 2Dense), RNN (LSTM), Other DL (4 DNN)	73:27	✓	✗	✗	✗	✗	✗
Moreira et al. [149]	2023	D	RF, KNN, SVM, CNN (Xception, Inception-ResNetV2 EfficientNetV2S), Other ML (NB, LR, SGD)	10f	✓	✗	✗	✗	✗	✗
Karbab et al. [109]	2023	D, C	Other ML	5f	Compressed Byte Entropy Matrix Visualization	✗	✗	✗	✗	✗
Zhan et al. [245]	2023	D	CNN	80:10:10	Grad-	✗	✓	✗	✗	✗
Dhanya et al. [62]	2023	D	CNN	50:50, 60:40, 70:30, 80:20, 90:10	✗	✗	✗	✗	✓	✗
Xuan et al. [231]	2024	C	BiTCN-TA EfficientNet	5f	✗	✗	✓	✗	✗	✗
Vasan et al. [213]	2024	C	BLS-RVFLNN	10f	✗	✗	✓	✗	✗	✗
Sharma et al. [189]	2024	C	CNN	80:10:10	GradCAM	GAN	✗	✗	✗	✗
Dong et al. [65]	2024	D	CNN-GRU, VAE, CAE, Vanilla AE	70:30	✗	✓	✗	✗	✗	✗
Kumar et al. [120]	2024	C	CNN, KNN, SVM, RF	-	✗	✗	✓	✗	✗	Soft voting
Yang et al. [241]	2025	D	CNN, RNN	-	✗	✓	✗	✗	✗	✗
Meng et al. [145]	2025	D	DNN	-	GNN Explainer, GradCAM	✗	✗	✗	✗	✗
Yang et al. [238]	2025	D	CNN,GCN	10f	✗	✗	✗	✗	✗	✗
Johnny et al. [105]	2025	D,C	CNN(VGG-16)	-	GradCAM, SHAP, t-SNE	✓	✓	✗	✗	✗
Ambekar et al. [9]	2025	D	Other DL (Siamese)	KFold	✗	✓	✓	Siamese	✗	✗
Alam et al. [7]	2025	C	CNN(spatial)	-	✗	✓	✗	✗	✗	✗
Zhang et al. [247]	2025	C	CNN	90:10	✗	✓	✓	✗	✗	✗
Yu et al. [243]	2025	C	CNN	60:40	✗	✗	✓	✗	✗	✗

The authors experimented with the design of CNN architecture from scratch to better fit the problem of visual malware classification. With this method, researchers are not bound by the designers' choices of other commonly used CNNs and can better select the best parameters for their CNN. In [159], the authors adapted the original multi-layer CNN by LeCun et al. The Neural Network architecture consists of a sequence of two convolutional layers, two subsampling layers, and three fully connected layers. They applied 32 filters of size 2x2 for the convolution process, and for subsampling, they utilized max pooling with a size of 2x2. In their study, the authors employed the BIG2015 dataset and obtained a classification accuracy of 99.260%. Cui et al.[52] presented a Deep Learning-based model that utilized 2D malware images from the Malimg dataset. They applied a custom algorithm to balance the dataset and achieved an accuracy of 94.5% using CNN. The authors of [154] introduced an architecture that integrates malware visualization with a DCNN model, enabling the detection of malicious activities within the IIoT environment. They achieved a detection accuracy rate of 97.81% on the Leopard Mobile Malware Dataset and 98.47%on the MalImg dataset. In [78], Gibert et al. implemented an original convolutional Neural Network (CNN) comprising three convolutional layers, followed by a fully-connected layer. The model proficiently classified samples from the MalImg and BIG2015 datasets, achieving accuracies of 98.48% and 97.49%, respectively. Recent studies have also been performed by integrating Bi-Directional Temporal Convolutional Networks (BiTCN), which process input sequences in both directions to capture contextual dependencies, transfer learning with EfficientNet for feature adaptation, and Atrous Spatial Pyramid Pooling(ASPP) for multi-scale feature extraction. The authors improved the classification accuracy. In [213], the authors integrate a Broad Learning System (BLS), based on the Random Vector Functional Neural Network (RVFLNN), allowing dynamic model expansion without retraining, distinguishing it from traditional CNN-based methods.

Researchers have used 1D-CNN [89] [81] [56] as well as 2D-CNN for malware classification. However, some researchers argue that the large number of parameters in 2D CNN adversely affects training speed [248]. Additionally, standard convolution methods need to be improved for extracting features from bytecode sequences, as it is essential to take into account the sequential nature of the bytecode, which determines the intensity of successive pixels. Representing this sequential data as a 2D image introduces distortions and disrupts the original sequence. Furthermore, 2D CNN is ineffective for analyzing sequentially structured data, as the convolutional operations consider neighboring pixels in a 2D space, even when they are not inherently related. The Researchers of [7] proposed a novel Spatial Convolutional Neural Network (Sp-CNN). In contrast to conventional CNNs employing uniform symmetric kernels, Sp-CNN introduces a partitioning strategy along the height dimension. This enables the network to focus on distinct input slices for individualized feature extraction. The iterative and independent processing of these slices facilitates the preservation of unique characteristics across malware classes and effectively captures nuanced spatial relationships between pixels. Furthermore, Sp-CNN operates on high-dimensional features generated by preceding conventional CNN layers, ensuring the retention of critical spatial information throughout the classification pipeline. This architecture demonstrates a significant enhancement in the accurate identification and categorization of diverse malware types, particularly in the context of imbalanced datasets.

In [247], the authors presented Image-based Malware Classification with Multi-scale Kernels (IMCMK), a variant Convolutional Neural Network. Unlike standard CNN architectures that leverage stacked layers of small convolutional kernels to increase the receptive field, IMCMK introduces a Multi-scale Kernel (MK) block. This block combines the benefits of both large and small kernels, enabling the model to extract both detailed and broad contextual information from malware binary images. Additionally, an improved Squeeze-and-Excitation (SE) block is integrated to capture channel dependencies, allowing for enhanced feature selection and representation. The IMCMK architecture also employs efficient multi-scale kernel fusion strategies to mitigate the parameter overhead associated with larger kernels, thereby improving computational efficiency while preserving high classification performance. In [243], the authors employed Spatial Pyramid Pooling (SPP) within a CNN framework to mitigate the challenge of varying input image dimensions inherent in malicious code representations. SPP addresses this issue by performing pooling operations across multiple spatial scales, producing fixed-size outputs irrespective of input image dimensions. This process divides feature maps into progressively finer sub-regions, capturing essential visual information at different spatial hierarchies. Consequently, the model achieves enhanced robustness and preserves semantic coherence within feature representations, leading to more accurate malware family classification by ensuring uniform input dimensions for subsequent processing stages. This is particularly advantageous for malicious code analysis, where input image sizes exhibit significant variability due to code complexity.

In most of the research, CNNs have shown high accuracy (98-99.9%) in detecting malware based on visual features extracted. They can learn to differentiate between benign and malicious visual patterns, leading to reliable detection results[211, 132, 204]. Compared to traditional ML techniques, CNNs have the capability of hierarchical representation learning, where they can capture low-level features and more abstract and complex features in different layers, which allows the network to learn progressively and combine features at different levels of abstraction.

Recurrent Neural Networks (RNN). Researchers have extensively explored the application of Recurrent Neural Networks (RNNs) for image-based malware detection, capitalizing on their remarkable ability to handle sequential data. RNN analyzes the images, pixel by pixel or via extracted feature vectors, to identify malware-indicative patterns. It remembers past inputs through internal states, enhancing pattern recognition when context matters. Precisely, RNN architectures like Long Short-Term Memory LSTM) effectively capture spatial dependencies within images through sequential processing. By transforming image data into sequences of vectors, RNNs empower the network to discern and recognize patterns indicative of malware during the training process. The significant advantage of RNNs lies in their capability to capture essential sequential information, such as pixel arrangement, contributing to highly effective malware detection.

Transfer Learning and Fine Tuning. In the field of transfer learning, pre-trained Neural Networks are employed for tasks that differ from their original training domains. This approach harnesses knowledge gained from various tasks, enhancing the effectiveness of visualization-based malware detection. Rather than training powerful and deep Neural Networks from scratch on malware images, transfer learning allows researchers to utilize existing models, saving time and resources. Fine-tuning is an additional technique to improve Neural Network performance in the specific domain of malware image classification. This method leverages the pre-trained model's knowledge from the larger dataset and adapts it to the smaller, task-specific dataset. During fine-tuning, the earlier layers of the pre-trained model are often frozen, while the later layers are updated to accommodate the new dataset. Transfer learning is particularly beneficial for limited or imbalanced datasets, which is common in malware detection. Researchers have developed a middle-ground approach that combines creating a convolutional Neural Network (CNN) from scratch with a known architecture. This method involves starting with a general CNN structure and then adapting the Neural Network to achieve an optimal balance between power and simplicity. This technique is commonly employed to facilitate transfer-learning with a simplified CNN, as demonstrated in studies [61] and [118].

Kalash et al. [107] utilized a deep Convolutional Neural Network (D-CNN) model to categorize malware binaries in grayscale image format, where the images had dimensions 224x224. Their study introduced a CNN model architecture based on VGG-16. They achieved an accuracy of 98.52% on the MalImg dataset and 99.97% for the BIG2015 dataset. Vasan et al. proposed IMCFN [211], a malware classification algorithm that utilized malware visualization with a pre-trained VGG-16 model. Subsequently, they compared the performance of their proposed solution with ResNet50 and Inception V3, achieving an accuracy of 98.82% and 97.35% for the Malimg malware dataset and IoT-android mobile dataset, respectively. The authors proposed an alternative approach and conducted experiments with ResNet models in [183] [199]. The enhanced model obtained an accuracy of 99.18% on the MalImg malware dataset and 98.63% on the BIG2015 malware dataset by modifying the final layer of ResNet50 in [199]. Another commonly used Neural Network archetype is the EfficientNet family of CNNs developed in 2019 [202]. These networks have demonstrated superiority over older counterparts in both the ImageNet challenge and the realm of malware analysis, as evident in studies by [169] [37] [2].

Researchers in [204] opted for the family of DenseNet CNNs [94]. DenseNet-based CNNs exploit their dense connectivity, which allows each layer to access and utilize information from all preceding layers. DenseNet promotes feature reuse, reduces vanishing gradient issues, and enhances overall network performance. The authors in [5] employed InceptionV3 to obtain a classification accuracy of 98.76% on the BIG15 dataset. In addition, the author in [42] also experimented using InceptionV3 and obtained 90% accuracy for a self-created dataset of $10,849$ malware. The research detailed in [172] delves into ShuffleNet, an alternative CNN-based model. ShuffleNet divides filters into groups, allowing for feature map shuffling within each layer. This approach promotes cross-channel interactions and facilitates streamlined information flow for improved efficiency. The customized ShuffleNet V2 model was employed in the Malimg dataset, achieving an accuracy of 99.03%. The study [120] introduced Intelligent Malware Classification using Deep Convolutional Neural Networks (IMCNN), a novel approach leveraging pre-trained CNN models (VGG16, VGG19, InceptionV3, and ResNet50) for efficient malware image feature extraction. Unlike traditional CNN methods, IMCNN customizes these models specifically for malware detection, enhancing feature relevance and reducing complexity to achieve effective and high-performing classification. This targeted application highlights the adaptability of pre-trained CNN architectures in malware analysis.

5.4.3 Transformers and Attention

Attention mechanisms, autoencoders, and transformers have emerged as pivotal advancements, drawing considerable interest due to their capacity to process and represent information proficiently. These cutting-edge techniques have sparked new avenues of research and application, particularly in domains such as natural language processing and computer vision. Attention [16] is a mechanism that allows models to focus on relevant parts of input when processing an element. Transformers [214] is a Neural Network architecture mainly used in the field of NLP that extensively employs attention. It can be used as an explainability tool for black-box Neural Networks or exploited to generate novel malware classification models. Transformers in computer vision actively assign importance to various image regions through self-attention, enabling them to capture complex spatial structures and extended dependencies. By analyzing the entire input sequence at once, self-attention equips the model with access to global context. This contrasts with traditional sequential models like RNNs, where the fixed processing order limits information flow.

Authors in [86] employ an encoder with a multi-scale CNN structure to generate malware images and organize tokenized malware features into sentences. It then leverages language models as textual encoders. This model obtained an accuracy of 99.32% on the MalImg dataset, which is equal to the state of the art. Chen et al. [44] use a combination of a transformer with a lambda layer which replaces the original attention mechanism to calculate the potential association information between different tokens by substituting it with a quicker single linear function. The outcomes achieved using this method showcase state-of-the-art performance, reaching an accuracy of 99.30% on the Big2015 dataset. An alternative method employed by [126] involves employing distinct autoencoders, each fine-tuned to recognize a specific malware family. In this approach, every autoencoder undergoes exclusive training solely with samples from a particular class. The fundamental concept lies in the premise that if an input image corresponds to the family a specific autoencoder is specialized in, the reconstruction error—indicating the distinction between the original and reconstructed images—will be minimal. Researchers leverage this reconstruction error observation to determine the family or class to which an input image pertains. Lately, researchers in [3] applied both CNN and Vision Transformer (ViT) to analyze images derived from data within PCAP files.

5.4.4 Model Fusion

Enhancing malware visualization systems involves leveraging model fusion, integrating feature representations from multiple models, or combining predictions made by individual models, as explored in studies by Lad et al. [123], Vasan et al. [212], and Jian et al. [101]. Ensembling strategies, such as weighted voting, majority voting, stacking, or boosting, stand out for their ability to generate robust and accurate outcomes. In a related contribution detailed in [181], authors introduced a unique approach to ensembling known as the hybrid stacked sequential flow. This method intertwines the cascade process with representation learning using ensembled tree forests, achieving an accuracy of 98.91% on the MalImg dataset and 96.84% on the Big2015 dataset. The approach, as proposed in [130, 36], entails constructing an ensemble of multiple CNNs and subsequently combining the outputs of these networks to address the malware classification problem. In [123], the authors aimed to explore the feature extraction capabilities of CNNs and harness the classification power of SVMs. The proposed hybrid CNN + L2-SVM model achieved an accuracy of 99.59%. Quan Le et al. [124] employed Deep Learning on raw input images and transformed them into 1D vectors using a generic image scaling algorithm. The resulting CNN-BiLSTM classification model achieved an accuracy of 98% on the BIG2015 dataset. However, a drawback of this approach was the considerable conversion time of the generic image scaling algorithm, which took approximately 191.2 seconds. More recently, researchers in [183] explored models created using the VGG family, and they introduced a bi-model architecture wherein they sequentially stacked the same models to achieve improved performance. They utilized the output of the first model to train the second one, resulting in 100% accuracy for the MalImg dataset.

5.4.5 Augmentation

The effectiveness of detection models heavily relies on the availability of a sufficient number of training samples. Unfortunately, this is a common issue with many datasets today, as there often aren't enough training samples. This presents a significant challenge for Deep Learning models, where the quality and quantity of data are critical to their effectiveness. For instance, the BIG2015 dataset, as shown in Table 3, has a high level of imbalance, with the `Kelihos_ver3` malware family consisting of 2942 samples, while the `Simda` malware family has only 42 samples. This imbalance adversely affects both the training process and classification performance. A trivial solution is using oversampling or undersampling. Inevitably, this technique discards some of the initial information, and researchers will experience a trade-off between achieving a good classifier and utilizing all possible information. The problem of imbalanced datasets has pushed authors to consider alternative ways to use the information found in the samples. Data augmentation techniques are frequently employed to tackle the issue of imbalanced datasets in image-based malware detection. These techniques generate additional training data by applying a broader range of data variations to the original images. These variations can generate similar but different images from the available sample information. The new images will not represent actual malware found in the wild but can be helpful for the classifiers to generalize features and patterns found in the images. Different methods can achieve these variations, with the most straightforward involving spatial transformations such as rotation, scaling, flipping, color adjustments, and noise injection. MIGAN[189] integrates image synthesis with classification using a generative adversarial network (GAN). The authors claim that they enhance malware detection by generating high-quality synthetic images, addressing class imbalance, and improving classification accuracy. Another two-phase framework uses autoencoders (vanilla, variational, and conditional) to reconstruct and enhance malware representations, then applies a CNN-GRU hybrid classifier to address class imbalance in greyscale and RGB IoT datasets[65]. According to the authors of [211], properly implementing data augmentation methods can mitigate overfitting problems and enhance the model's robustness. They utilized techniques such as rescaling and shearing, resulting in a modest 1.01% improvement compared to the absence of data augmentation on the MalImg dataset. In their work, as discussed in [26], the authors utilize the Synthetic Minority Oversampling Technique (SMOTE) algorithm, drawing from the principles of the K minority class nearest neighbors model, as outlined in the source [27]. This technique entails creating extra images for families with fewer samples while reducing the sample size of families with excess samples, addressing dataset balance issues.

Unsupervised learning becomes increasingly important when labeled data is scarce, and knowledge extraction is essential. Autoencoders use principles from data compression algorithms to capture the core features of the original data in a compressed feature set. Unlike traditional Neural Networks, autoencoders aim to approximate the input closely using fewer data without explicit input/output pairs or supervision. This autonomous approach effectively addresses the issue of dataset size in learning processes. D'Angelo et al. [66] converted the sequences of API calls invoked by apps into sparse image-like matrices called API images. They utilized autoencoders to extract the most informative features from these images. They subsequently presented the extracted features into an artificial Neural Network (ANN)-based classifier for detection. They obtained an accuracy of 0.95 on a customized dataset created using samples from Malgenome and contagio mobile datasets.

Another approach involves the utilization of the Variational Auto Encoders(VAE) [114], which comprises an encoder and decoder structure. While an Autoencoder primarily focuses on data compression and encoding, the VAE emphasizes the creation of a latent vector by estimating the probability distribution of input data through its encoder. In particular, VAE functions as a statistical probability distribution model that learns the distribution of the dataset and facilitates the generation of novel data samples. This capability enables the generation of new data samples using the decoder. Similarly, the Conditional Variational Autoencoder (CVAE)[196] follows the same structure and objectives as the VAE. However, in contrast to the VAE, the CVAE incorporates condition information and input data during the learning process. In [210], the authors introduce an attention mechanism to selectively assess weights in the VAE, facilitating the acquisition of crucial features in the latent space. Additionally, they integrate a lightweight CNN to capture lower-level features, ensuring a diverse representation of features. They obtained an accuracy of 99.40% using RandomForest on the MalImg dataset.

Another solution is to leverage Generative Adversarial Networks (GANs) [82], which consists of a generator and a discriminator Neural Network. Y. Lu and J. Li employed DCGAN [136], a GAN-based approach utilizing convolutional Neural Networks, which led to a 6% increase in accuracy. Using GAN-generated samples, they trained an 18-layer deep residual network as the malware classifier. The training process included multiple convolutional transpose layers, generating 2,250 synthetic samples for each class. The deep residual network achieved an overall average testing accuracy of 84%, with improved performance in all other metrics for classes with larger sample sizes. Researchers utilize a Cycle GAN to generate novel training samples in their study detailed in [204]. This GAN learns the distinctive features between input and output images, employing these feature maps to execute image-to-image transformations and create new samples. This approach aims to produce images closely resembling realistic malware images. Augmenting datasets with such authentic-looking images could significantly benefit the development of an improved classifier. However, it remains crucial to rigorously test the model on unseen test samples to ensure it doesn't erroneously generalize by relying on features absent in real malware. In a related investigation by Burks et al. [33], researchers compare a VAE model and a GAN model using a ResNet18 classifier. In the MalImg dataset, using GAN led to a performance increase of 6%, while the integration of VAE-generated samples resulted in a 2% improvement. Based on these results, the researchers reported that generating artificial malware data using GANs is more effective than using VAEs for Deep Learning-based malware classification.

5.4.6 Few-shot Learning

Few-shot learning (FSL) [69] is a Machine Learning technique that enables models to recognize and generalize to new classes with a limited number of labeled examples per class. Several Few-Shot Learning (FSL) models employ a meta-learning framework. This framework continually updates the model using mini-batches covering diverse tasks. These tasks are drawn from a latent task distribution based on the training set's characteristics. Consequently, the model acquires extensive and advanced knowledge that can be universally applied to all tasks, facilitating proficient classification across various mini-batches. The selection of the loss function holds paramount importance in Few-Shot Learning (FSL) since it steers the model in generalizing from a restricted set of examples.

The N-way-K-shot classification task is a common consideration, aiming to classify examples into N distinct classes accurately. The challenge lies in having only K-labeled examples available for each class, as discussed in works by Hsiao et al. [93] and Barros et al. [23]. The imbalanced data distribution among different malware families significantly influences the effectiveness and generalization ability of Few-shot learning approaches [17]. Researchers have proposed solutions using Siamese Neural Networks to tackle this challenge. A Siamese network, comprising two identical networks sharing weights, aims to embed malware instances into a continuous vector space. This network is trained on pairs of samples, treating instances from the same family as positive and those from different families as negative. The resulting feature extractor, trained by the Siamese network, can subsequently link to a classifier for predicting class labels. Hsiao et al. in [93] utilized Siamese Neural Networks to classify malware images, employing techniques such as average hashing and malware visualization during data preprocessing. Similarly, in[17], the suggestion was made to use Siamese Neural Networks for Android malware classification, connecting the Siamese Neural Network to a multilayer perceptron for classifying generated embeddings. Another approach in [179] involved generating grayscale images from network traffic data of malware and implementing a prototype-based few-shot learning model. The Siamese network utilized in the study [9] specifically employs a shared-weight architecture, where two identical sub-networks process pairs of images: one for the original malware image and the other for its adversarial counterpart created using the Fast Gradient Sign Method (FGSM). This design allows the network to learn a consistent feature embedding for both images, enabling it to compute a similarity score that quantifies the degree of similarity.

5.5 Evaluation

This section provides a comprehensive analysis of the evaluation methods used in the field with a particular focus on the comparability of results obtained by different research groups. We present the different software choices that researchers have to make to implement their model and how these have to be considered when presenting the results to the scientific community. Finally, we discuss the metrics commonly found in papers and their efficacy in judging a model.

5.5.1 Software

ML can uncover patterns and insights from large and complex datasets that may be difficult or impossible for humans to discern. Implementing ML algorithms from scratch can be a challenging and time-consuming task. Libraries provide pre-implemented algorithms, tools, and functionalities that simplify developing and deploying models. Selecting the suitable ML library holds significant importance during model implementation and when evaluating the results. Even when employing the same algorithm, the back-end implementation in various libraries may exhibit variations, leading to differences in the output that could significantly impact the overall model evaluation metrics [197, 75]. In the following paragraphs, we'll present the prevalent Python libraries frequently utilized by researchers within this field.

Several tools and libraries are crucial in visualization-oriented malware detection and machine learning research. OpenCV [30] and the Python Imaging Library (PIL) [208] are powerful image-processing libraries that support essential tasks such as image loading, transformation, enhancement, feature extraction, visualization, and image comparison, while also integrating seamlessly with Machine Learning frameworks. Scikit-learn [167] offers a wide range of ML algorithms focusing on simplicity and readability, enabling rapid prototyping and model validation. PyTorch [165], known for its dynamic computational graph and modular design, provides flexibility for implementing cutting-edge deep learning models, while TensorFlow [1], with its scalable architecture and efficient computation graph, is suited for building and deploying complex ML models. Keras [47], a high-level API running on TensorFlow, further simplifies neural network development with its user-friendly interface. For data visualization, Matplotlib [96] is indispensable, supporting diverse plot styles from basic charts to intricate 3D visuals. Additionally, AVClass [121] aids in automated malware classification by aggregating antivirus labels, offering consistent and meaningful insights into malware taxonomy.

Hardware setup is critical for evaluating ML and DL models, as computational resources directly impact training speed, model scalability, and reproducibility—yet over 40% of studies omit detailed hardware specifications.

5.5.2 Metrics

The metrics commonly used to evaluate the models are:
Accuracy. It is a metric used to evaluate the overall performance of a classification model. It represents the proportion of correctly classified instances relative to the total number of instances. Although widely used due to its simplicity, accuracy may not provide a reliable measure when dealing with imbalanced datasets. The formula for computing accuracy is given below:

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

here, TP (True Positives) denotes the number of actual positive instances that were correctly predicted, TN (True Negatives) indicates the number of correctly predicted negative instances, FP (False Positives) corresponds to negative cases incorrectly classified as positive, and FN (False Negatives) refers to positive instances that were wrongly predicted as negative.

Precision. It evaluates how many of the instances predicted as positive are actually correct. It emphasizes the reliability of positive predictions, making it particularly important in scenarios where false positives carry significant consequences. Precision is calculated as the ratio of true positive predictions to the total number of predicted positive instances (i.e., the sum of true positives and false positives). The formula for precision is as follows:

$$\text{Precision} = \frac{TP}{TP + FP}$$

, here TP (True Positives) represents the correctly predicted positive instances, and FP (False Positives) identifies the incorrectly predicted positive instances.

Recall. It is also referred to as sensitivity or the true positive rate, which measures the ability of a classifier to identify all relevant positive instances in the dataset. It reflects how well the model minimizes false negatives. Recall is defined as the proportion of true positive predictions out of the total actual positive cases, calculated using the following formula:

$$\text{Recall} = \frac{TP}{TP + FN}$$

, here TP (True Positives) represents the correctly predicted positive instances, and FN (False Negatives) represents the incorrectly predicted negative instances. **F1-score**. This metric combines precision and recall into a single value. It provides a balanced measure of a classifier's performance. F1-score is calculated as the harmonic mean of precision and recall, giving equal weight to both metrics.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Matthews Correlation Coefficient (MCC). MCC is A measure that takes into account all values in the confusion matrix, giving a more comprehensive assessment of the model's performance.

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

, here TP (True Positives) represents the correctly predicted positive instances, TN (True Negatives) refers to the correctly predicted negative instances, FP (False Positives) identifies the incorrectly predicted positive instances, and FN (False Negatives) represents the incorrectly predicted negative instances.

Jaccard Index (JI).It measures the similarity between predicted and actual classifications by calculating the ratio of their intersection to their union. It provides an indication of how closely the predicted labels match the actual labels.

$$JI = \frac{TP}{TP + FP + FN}$$

here TP (True Positives) represents the correctly predicted positive cases, FP (False Positives) identifies the incorrectly predicted positive instances, and FN (False Negatives) represents the incorrectly predicted negative instances.

Fowlkes-Mallows Index (FMI). It is defined as the geometric mean of precision and recall, offering a balanced evaluation of the performance of a classifier. It is often used as a complementary metric to assess the quality of the predictions.

$$FMI = \sqrt{\text{Precision} \times \text{Recall}}$$

Confusion matrices. A confusion matrix is a table used to evaluate the performance of a classification model by comparing predicted labels with actual labels. It includes four key components: true positives, true negatives, false positives, and false negatives. This matrix offers a comprehensive view of the model's prediction outcomes and serves as the foundation for deriving metrics like accuracy, precision, and recall.

ROC curve. The Receiver Operating Characteristic (ROC) curve is a graphical representation of a classifier's performance across different discrimination thresholds. It plots the true positive rate (recall) against the false positive rate (1 - specificity) at various threshold settings. The curve shows the trade-off between true positive rate and false positive rate, and the area under the ROC curve (AUC) is the summary measure of a classifier's performance. A greater AUC value signifies improved classification performance. These assessments extend to multi-class classification challenges, such as those tackled through the One-vs-All or One-vs-One methods. Both methods enable the evaluation of a multi-class classifier through ROC curves but involve converting the multi-class issue into several binary classification tasks.

Resource Consumption Measurements. Additionally, certain researchers prioritize the efficacy of their proposed models. They offer metrics such as *training time*, *classification time*, and *CPU Usage* to highlight the time and resources expended by high-performing models, demonstrating their strengths and weaknesses. It is crucial to provide context to the quantitative outcomes by considering the hardware and software setup used for training and testing.

Merely assessing accuracy does not offer a comprehensive evaluation of a model. Therefore, considering precision, recall, and the F1-score as metrics is crucial for a thorough evaluation. Moreover, a confusion matrix aids in pinpointing how a

Table 6: Metrics used to evaluate classification models and reference articles

Metric	Ref.
Accuracy	[217] [72] [159] [236] [90] [58] [117] [198] [52] [200] [229] [227] [93] [78] [112] [152] [134] [171] [21] [237] [42] [6] [108] [195] [215] [146] [127] [154] [212] [29] [100] [211] [250] [63] [180] [80] [216] [176] [221] [76] [97] [192] [57] [15][204] [168] [24] [129] [2] [199] [222] [11] [228] [223] [101] [26] [67] [91] [79] [161] [251] [118] [143] [48] [253] [174] [39] [37] [172] [61] [140] [183] [113] [102][125] [109] [190] [149] [7] [247] [243] [65] [120]
Precision	[72] [159] [90] [52] [152] [229] [227] [78] [237] [108] [195] [146] [154] [212] [29] [211] [180] [80] [216] [176] [97] [192] [57] [204] [168] [24] [15][129] [222] [11] [228] [223] [101] [26] [67] [161] [251] [118] [143][253] [162][162] [174][37] [169] [172] [61] [140] [183] [102] [125] [109] [190] [149] [7] [247] [243] [65] [120]
Recall	[72] [159] [90] [52] [229] [37] [199][2] [227] [78] [152] [237] [108] [195] [146] [154] [212] [29] [211] [180] [80] [216] [176] [97] [192] [57][204] [168] [24] [15] [129] [222] [11] [228] [223] [101] [26] [67] [161] [251] [118] [143] [253][162] [174] [169] [172] [61] [140] [183] [102] [125] [109] [190] [149] [7] [247] [243] [65] [120]
F1-score	[72] [159] [90] [229] [227] [78] [152] [237] [108] [195] [146] [29] [211] [63] [180] [80] [216] [176] [76] [97] [57] [204] [15] [168] [24] [129] [2] [199] [222] [11] [228] [223] [101] [26] [67] [91] [251] [143][48] [253] [162][174] [169] [172] [61] [183][113] [102] [125] [109] [190] [149] [7] [247] [243][65] [120]
Confusion Matrix	[52] [78] [152] [6] [154] [212] [29] [211] [180] [216] [76] [97] [192] [57] [204] [168] [24] [15] [129] [2] [199] [222] [11] [228] [223] [26] [67] [161] [251] [118] [143] [253] [174] [37] [169] [61] [183][113] [109] [65]
ROC Curves	[217] [236] [6] [108] [146] [212] [180] [176] [15] [67] [118] [253] [169] [61] [109] [238] [120]
Time & Resource	[132] [97] [20] [122] [72] [159] [227] [78] [153] [152] [21] [195] [154] [212] [29] [59] [211] [250] [180] [216] [221] [57] [168] [199] [11] [229] [223] [101] [67] [91] [161] [253] [253] [48] [65]
Others (MCC, JI, FMI, etc.)	[9] [243] [7] [238] [241] [145]

model misclassifies samples within a specific class, providing an immediate overview of the classifier's discriminatory behavior. Lastly, ROC curves serve in binary classification directly or are adapted for multi-class scenarios to showcase a model's capability to differentiate between different groups. Table 6 showcases the chosen evaluation metrics for each paper concerning their models. Observe that, the "Other" row represents papers that employed extra pertinent metrics related to the visualization aspect of the problem.

5.6 Model Robustness and Adaptation

This section highlights adversarial attacks targeting visualization-based malware detectors. Researchers have demonstrated that ML and DL are susceptible to Adversarial Examples (AEs) [201, 235]. Adversarial attacks aim to either misclassify input into a different class from the legitimate source class or intentionally misclassify samples from any source class into a specifically chosen target class. The adversarial domain has been structured using a taxonomy [163] designed for multi-class Deep Learning classifiers. Adversaries can be classified according to their level of knowledge about the targeted model needed to carry out attacks: *(i)* white-box attack, *(ii)* black-box attack, and *(iii)* grey-box attack. A white-box attack requires complete access to the model. In a black-box attack, the attacker has limited or no visibility into the internal parameters or architecture of the targeted machine-learning model. In a grey-box attack, the attacker possesses limited information about the model, including architecture, training data, and gradients. An Adversarial Example (AE) pertains to an altered sample from the initial dataset, intentionally designed with slight modifications to trick Machine Learning-based malware detectors [201]. The adversarial image x_{adv} is generated using equation 4.

$$x_{adv} = x_{or} + \delta \tag{4}$$

where a small perturbation called δ is applied to the original image x_{org}. Various approaches can be employed to create adversarial malware images. One method involves implementing adversarial perturbations [224], which entail carefully crafting imperceptible modifications to deceive image classification algorithms. Gradient-based techniques leverage gradients to generate adversarial examples, employing methods such as the Fast Gradient Sign Method (FGSM) [83]. Generative Adversarial Networks (GANs) can also train models that generate realistic, misleading images. Researchers generally classify malware adversarial attacks into two categories: Attacks in the problem domain and attacks in the feature domain [131, 182]. In problem-domain attacks, adversarial attacks are designed to fool the model by manipulating the input data itself. This can be done by adding small, imperceptible perturbations to the input data that cause the model to make a different prediction. For example, an attacker could add a small amount of noise to an image. Conversely, feature-domain attacks, on the other hand, are designed to fool the model by manipulating the intermediate features that the model extracts from the input data. This can be done by adding or removing small amounts of activation to the model's neurons. These attacks frequently utilize gradient-based techniques in the image domain, such as DeepFool [148] and FGSM [83]. It's important to recognize that mapping from the feature space back to the original sample might not always succeed, possibly leading to creating samples that cannot be executed.

Targeted adversarial attacks aim to influence the model to predict a particular chosen target class. These adversarial examples are carefully crafted by subtly altering the data to guide the model's predictions toward the desired target class. Conversely, untargeted adversarial attacks pursue a broader goal of causing misclassification, not restricted to a specific target class. For example, they might create a function F that modifies input X' in a way that leads model M to misclassify X' as any class other than its original one. The challenge here is to ensure that the differences between X and X' remain imperceptible to human observers. In contrast, targeted attacks seek to bias model M toward a particular target class Y. Attackers devise the function F to produce a modified version of any normal input, prompting M to predict class Y.

In [133], Liu et al. introduce a framework known as ATMPA, the first white-box adversarial attack method to undermine visualization-based malware detectors. ATMPA starts by transforming the malware sample into a grayscale image representing its binary texture, and then altering the associated adversarial example using subtle perturbations produced with the help of FGSM and C&W techniques. However, a significant limitation of ATMPA is that the resulting adversarial grayscale image disrupts the original malware structure, rendering it unsuitable for practical PE malware detection in real-world scenarios. Khormali et al. introduce COPYCAT [112], an adversarial attack that makes use of existing generic adversarial attacks (e.g., FGSM, C&W, DeepFool, PGD, etc.) similar to ATMPA targeting visualization-based malware detectors that utilize CNNs. Subsequently, COPYCAT takes a different approach by attaching the adversarial image to the tail of the original malware image rather than integrating it directly into it. In contrast, Park et al. in [164] proposed an alternative adversarial attack technique that generates an adversarial image of malware using existing off-the-shelf adversarial attacks. Subsequently, the adversarial malware alignment obfuscation (AMAO) algorithm injects minimal `NOP` instructions into the original executable malware. This process aims to align the executables with the previously generated adversarial image, ultimately evading visualization-based malware detectors. The authors introduce AMGmal [245], an Adaptive Mask-Guided adversarial attack designed to evade malware detection with minimal perturbation while preserving malware functionality. They achieve this by leveraging GradCAM++ for saliency detection to identify crucial bytes in malware binaries and modifying only slack areas in the PE format. The approach employs three adaptive mask transformation strategies—Dilation, Erosion, and Heuristic—to balance evasion effectiveness and perturbation minimization. Experimental results show AMGmal reduces perturbation from 3.34% to 0.90% on Malimg and from 2.32% to 0.73% on SOREL-20M, achieving evasion rates of 68.09% and 64.75%. The method ensures functionality preservation and adaptability to other attack models but relies on slack area availability and accurate saliency detection. They suggest enhancing saliency precision, exploring black-box attacks, and developing robust dynamic and behavior-based detection techniques to counter adversarial malware evasion in the future.

The authors of [105] utilized Generative Adversarial Networks (GANs) to improve the robustness of its malware detection model against adversarial attacks. This process involves generating adversarial examples that closely resemble original malware images by perturbing them with a generator network, while a discriminator network learns to distinguish between real and adversarial samples. These generated adversarial examples are then incorporated into the training dataset alongside the original images, leveraging a similarity metric (Structural Similarity Index Measure) to ensure their resemblance to the originals. This adversarial retraining enhances the model's ability to correctly classify both legitimate and perturbed images, ultimately improving its resilience against attacks, as evidenced by more concentrated activation patterns in heatmaps following retraining. In [14], the authors evaluated deep learning models under white-box adversarial attacks using noise perturbations like Gaussian and Salt & Pepper. The models exhibited minor accuracy drops on MalImg (4.1%) and BIG2015 (11.9%), but performance on the Malhub data set dropped sharply (up to 98.5%) due to data imbalance. These results highlight that even simple adversarial noise can significantly affect detection reliability, especially in imbalanced settings, emphasizing the importance of incorporating adversarial robustness into malware detection systems. The study [9] employs adversarial training using the FGSM to enhance the resilience of the FASNet model against malicious attacks. Adversarial images are produced through the addition of carefully calculated perturbations to the original malware images, making them difficult to classify correctly. As the model undergoes training, it learns to classify both the original and adversarial images, which enhance its capability to handle deliberately misleading inputs. This exposure ensures that the model can accurately distinguish between benign and adversarial samples, thereby bolstering its robustness against adversarial attacks in real-world environments.

Attacks on models. Machine learning (ML) has enhanced malware detection by enabling the identification of both known and novel threats, it simultaneously opens new attack surfaces. Adversaries increasingly exploit these vulnerabilities through sophisticated techniques to subvert and manipulate ML-based detection systems. Recent work in[128] presents a novel backdoor attack targeting ML-based Android malware detectors, exposing critical vulnerabilities in these systems. Unlike traditional adversarial attacks, their approach operates without access to the training data, enhancing its realism and threat level. The authors inject adversarial Android applications containing stealthy, trigger-based perturbations selected using a Genetic Algorithm into the training pipeline. These samples are mislabeled as benign and modified at the APK level to preserve functionality while evading detection. Evaluated on four prominent detectors—Drebin, MaMaDroid, DroidCat, and DroidAPIMiner—the attack achieves evasion rates of up to 99%, with minimal impact on overall model accuracy. The study identifies feature interdependencies as a limitation and underscores the need for robust defenses via improved feature selection and labeling.

Another effort in this direction is the Jigsaw Puzzle (JP) attack proposed in [239], which improves the stealth of backdoor strategies by selectively targeting specific malware families. Unlike traditional attacks that poison entire classes, JP activates only when a unique trigger aligns with the latent features of the attacker's malware. The authors design the attack in a clean-label setting without controlling training or labels, enhancing realism. JP achieves high evasion on attacker-owned samples while preserving accuracy and maintaining low false positives. Evaluations on a large Android dataset show JP evades advanced defenses like MNTD by violating their broad-spectrum backdoor assumptions. While the study focuses solely on Android malware and uses fixed hyperparameters across families, it highlights the urgent need for adaptive defenses and calls for further research to generalize selective backdoor strategies to broader classification contexts. In [246], the authors present a framework that exploits third-party crowdsourced threat intelligence, allowing adversaries to inject poisoned data without altering labels. They embed universal adversarial triggers in unused PE header bytes, preserving functionality while manipulating predictions. Using heuristic search, they craft triggers that misclassify malware as benign. Experiments show high attack success and evasion of defenses with minimal impact on clean sample accuracy. Despite relying on soft labels and fixed regions, the method exposes weaknesses in static detection systems. The study emphasizes the need for research into sample-specific triggers, adaptability to dynamic detectors, and stronger defenses against clean-label backdoor attacks.

In [184], the authors highlight that the widespread use of crowdsourced threat intelligence by security vendors creates a practical avenue for injecting poisoned data into training pipelines. They demonstrate the vulnerability of feature-based malware classifiers and propose a model-agnostic, explainable AI approach using SHAP to embed stealthy backdoors via influential features. The method achieves high attack success across diverse datasets and models while remaining difficult to detect due to benign sample diversity. The study highlights the dual-use risk of explainable AI in adversarial contexts, assuming knowledge of the victim model's feature space and showing variability across datasets. It calls for research into generalizable attacks and robust defenses, especially against stealthy clean-label threats in hybrid detection systems.

6 Interpretability

This section focuses on the vital realm of interpretable techniques for detecting image-based malware. Explainability holds immense significance in ML, addressing the need for clarity and responsibility in automated decision-making across diverse sectors like healthcare, finance, and autonomous systems [144]. Legislations like GDPR's Article 22 underscore the right for individuals to comprehend the logic and implications behind automated decisions, driving the call for transparency [220]. In cybersecurity, this transparency is non-negotiable, as ambiguity within security systems can lead to grave consequences. Understanding the 'why' behind these systems' decisions is crucial to prevent vulnerabilities and malicious activities. While some Machine Learning models naturally explain their outputs, Neural Networks—a prevalent model class—lack inherent transparency. This ambiguity has created a need for methods that make these models more understandable. This isn't just about trust and fairness but also about ethically using these systems.

The researchers in [234] applied a CNN with attention to identifying critical regions for classification, facilitating the extraction of distinctive byte sequences unique to the malware family. Even without prior knowledge, this method offers significant insights to human analysts. Moreover, examining the connections in attention maps between malware samples belonging to the same family assists analysts in pinpointing specific positions unique to targeted malware families. In their work [103], the authors investigate how the Vision Transformer (ViT) exploits its strengths to offer a robust understanding of complex patterns in malware images. The attention map produced during detection identifies crucial elements such as class and method names. Real-world dataset validation demonstrates an 80.27% accuracy in malware detection. Furthermore, they computed an "interpretability score" [225], surpassing other interpretable Machine Learning (ML) methods like Drebin, LIME, and XMal. This underscores its potential contribution to explainable artificial intelligence within cybersecurity.

The authors of [97] presented an interpretable approach to detect malware in the Android environment. They employed the Grad-CAM algorithm to facilitate the visual debugging of models and gain insights into the specific areas that receive focus when predicting the maliciousness of Android application images. Hamad et al. in [150] developed a pre-trained Inception-v3 transfer learning model to analyze malware in IoT devices. They used Grad-CAM to generate visual explanations through cumulative heatmaps, providing insights into the learned features of the CNN models. Additionally, t-SNE was employed to assess feature density within the proposed CNN models. In [43], the authors have utilized LIME to generate super-pixel (patches of pixels) plots for malware images. An interpretable approach using an ensemble of eight CNN-based pre-trained models is proposed in [130]. They have also conducted experiments to enhance accuracy by leveraging the resultant interpretations. Furthermore, the authors of [71] suggested that explainable AI models demonstrate resilience against adversarial attacks. They highlighted the potential of these models in detecting adversarial inputs or samples by generating distinctive explanations using the Shapley Additive Explanations (SHAP) for perturbed samples. They generated SHAP Signatures for the internal layers of a Deep Neural Network classifier, which helped assess features' relevance and importance in distinguishing between normal and adversarial inputs. The authors of [105] use SHAP to attribute individual feature contributions to model predictions, helping identify key influences on outcomes. Grad-CAM generates heatmaps to highlight important image regions, showing which areas most impact classifications. Additionally, t-SNE visualizes high-dimensional data in two dimensions, allowing the examination of clusters and patterns among different malware families.

The quantitative evaluation of interpretation quality is paramount. Metrics such as Fidelity [87], stability[68], and Robustness[68] are three well-established measures used to assess the quality of explanations. Fidelity evaluates individual explanation accuracy, while robustness quantifies dissimilarity between explanations from different families. Stability assesses similarity among explanations from the same interpretation method on identical models.

7 Applications and Real-World Deployment Contexts

Visualization-based malware detection has found applications across platforms including Windows, Android, and IoT environments such as smart factories and autonomous vehicles [166, 170, 150, 173]. These methods typically convert binary or behavioral data into grayscale images for classification using convolutional neural networks (CNNs). Recent work shows that embedding ML-based image analysis directly into IoT and vehicle platforms can enable low-latency malware detection. For example, Patel et al.[166] continuously monitor an autonomous vehicle's network traffic, convert malware binaries into gray-scale images, and apply a ResNet50V2 CNN to classify them. In a smart-factory IIoT case, Kim and Lee deploy a three-tier edge/cloud architecture (on-device, edge server, and cloud) that runs a CNN on visualization images of malware (from the Malimg dataset) and reaches 98.9% accuracy. Prathiba et al.[170] take a complementary approach by using blockchain to label trustworthy AV nodes: onboard sensors detect malicious code and isolate infected vehicles, and only vetted vehicle IDs are recorded on-chain. This Blockchain-enabled scheme achieves 0.99 F1 for malware detection. ViT4Mal[173] demonstrates partial deployment of a lightweight vision transformer on a Xilinx FPGA, achieving real-time malware detection on edge devices with minimal latency and high accuracy, though decoder execution remains offloaded to a host CPU. In practice these systems span Windows executables, Android apps, and general IoT devices, leveraging CNNs on binary-to-image conversions and integrating with SOC or SIEM infrastructures. The blockchain logs and on-device inference bolster trust and privacy, but real-world deployment must still tackle issues of cross-device scalability, heterogeneous malware datasets, and the interpretability of deep models.

8 Lesson Learned

This section presents a comprehensive discussion of the insights gained from the analysis conducted in this study. The main findings derived from our survey are outlined below:

- Relying on outdated datasets risks poor model generalization to new malware variants, underscoring the necessity of using up-to-date datasets that reflect current threats for robust detection.
- Research largely focuses on static feature-based image generation, but combining it with dynamic visualization—despite its computational cost—offers more profound insights and improves the accuracy of malware analysis.
- The standard method found in the literature to generate images from malware is the bytecode gray scale method and its variants. This method interprets the bytes that form the malware file as grayscale values for a pixel. One can use the technique with the sample's opcodes with minor adaptation.
- Images can also be created by using dynamic features of the analyzed malware. Authors have experimented mainly with API calls, system calls, and network traffic. The methods used to transform these features in images vary a lot among authors, and the problem is often less explored;
- Researchers experimented with different methods to reorder and reinterpret the bytes to convey more useful information in an image. In particular, Markov images help classifiers to identify repeated patterns inside the code, SFC is used to maintain the vicinity of each byte of a sample, and hashing schemes are another way to maintain the locality of each byte. All these methods have proven to increase the accuracy of the analyzed classifiers. However, the literature needs a proper, well-defined benchmark dataset and environment with fixed parameters on which models can be objectively tested and compared.
- Entropy can also be used to represent a sample uniquely. The use of Entropy is interesting because it also conveys information about obfuscated malware samples.
- Various methods exist for extracting features from images, generally categorized into extracting global features and local features.

- The combination of local and global features has been proven more and more as the best-performing one
- The extraction of features through a Neural Network, particularly a CNN and its variants, has been proven better than any other feature extractors that do not use domain-specific information. Deep learning seems to be the way forward if we do not consider handcrafted features generated by domain experts.
- There is no way to objectively compare different feature extractors because different authors usually use different classifiers in their research, and so the final results are not comparable. At the feature level, few authors have pushed to interpret the extracted features, and the only explanations are related to inherently interpretable algorithms used to extract those features.
- Limited research has been conducted on interpretability in the context of DL malware image visualization, with CAM explanations emerging as the most promising avenues. Further experimentation in this domain is imperative, particularly in light of the growing demand for explanations driven by GDPR and similar laws requirements and other pressing security robustness concerns.

9 Challenges and Future Directions

The cybersecurity research community has recently voiced concerns about the efficacy of a visualization-based approach in malware classification. Research on such approaches is gaining momentum. To contribute to the literature, we comprehensively investigated various steps and procedures in the visualization-based malware detection domain. This section delves into potential research challenges and future directions as visualization-based methods in malware detection become more prevalent.

Obfuscated data. By utilizing visualization techniques, analysts can better understand the behavior and structure of obfuscated code, enabling them to identify malicious patterns and enhance the detection process. Commonly employed obfuscation techniques include injecting dead code, reshuffling subroutines, rearranging code, etc. [242, 185]. Texture-based methods in malware detection enhance resilience against obfuscation techniques [157]. Rather than focusing on code-level modifications, these methods analyze image texture patterns and visual characteristics, including color distribution, edges, and local patterns. They exhibit resilience to code obfuscation, packing, and encryption if they do not introduce significant visual alterations. These methods also demonstrate robustness to code-level modifications by effectively capturing stable high-level visual patterns. By performing statistical analysis on textures, such as histograms and co-occurrence matrices, these methods offer a robust representation that captures higher-level information. However, one significant challenge with many visualization approaches is their reliance on computing texture similarity. While these approaches effectively tackle code obfuscation issues, they demand substantial computational resources for extracting intricate texture features from malware images, including LBP, GIST, DSIFT, and GLCM. Analyzing specific malware obfuscations poses significant challenges due to their potential utilization of diverse packing techniques and resolutions. Consequently, examining them solely based on textual features becomes a difficult task[215]. In [57], the authors introduce a semi-supervised approach that uses grayscale images to visually analyze executables, aiming to detect and classify obfuscated malware. Meanwhile, Vasan et al.[211] tackle the challenge of obfuscated malware classification by employing a fine-tuned CNN architecture. This architecture aims to extract resilient features by incorporating translation invariance, capturing distinct representations of obfuscated patterns, and adapting to evolving obfuscation techniques. Furthermore, in [212], the same authors initially tested their model on MalImg and subsequently on packed and salted samples. The model achieved an accuracy of 98.11% against packed malware, a common obfuscation technique, and 97.59% against salted malware. Salting malware, observed in other studies like [232], involves inserting benign samples, commonly found on recent Windows PCs, into each malware family. Researchers in [62] proposed a method to classify obfuscated malware and identify obfuscation types in Android IoT applications by converting bytecode into Markov images and applying a CNN. Although this approach improves robustness against obfuscation, its effectiveness depends on image quality, remains susceptible to concept drift, and may not address all obfuscation techniques. They recommend integrating dynamic analysis and implementing continuous learning to improve detection performance.

Sustainability. Concept drift is a key challenge in sustainable malware detection, as the statistical properties of malware evolve over time, reducing the effectiveness of ML models. Traditional models trained on historical datasets become outdated and struggle to classify new malware accurately. While some researchers have addressed this issue, no systematic methodology has been established. A trade-off exists between using fixed benchmark datasets (e.g., Big2015, MalImg) and more current datasets. Solutions include testing models on benchmark datasets for comparability before evaluating them on newer data or periodically updating benchmark datasets with new samples. To mitigate concept drift, continuous model updates, adaptive learning techniques, ensemble methods, and the integration of new features and data sources are essential for maintaining effective malware classification. Unfortunately, the literature often overlooks a direct analysis of concept drift, as evident in Table 5. This oversight is rooted in the fact that benchmark datasets typically remain static over time. Consequently, evaluations of the presented models stand as singular snapshots, subject to judgment based on the future evolution of malware over an extended period. Malware visualization approaches have seen a limited exploration of concept drift, with only a few works addressing the issue in the context of malware detection using alternative features. Transcend [106] introduces a framework that innovatively identifies aging classification models and issues an early warning before the model consistently makes poor decisions due to outdated training. The framework achieves this by employing statistical comparisons of samples encountered during deployment with those utilized for model training, thereby establishing metrics for prediction quality. In[22], the authors revisit the conformal evaluator and Transcend [106], aiming to establish their internal workings on a solid theoretical foundation and identify optimal operational settings. They also introduce Transcendent, a framework for drift detection with computational efficiency. DroidSpan in[34] introduces a behavior profile capturing the distribution of sensitive accesses in Android apps. This profile enables DroidSpan to demonstrate superior sustainability, consistently separating benign and malicious apps over eight years. The study also deals with the crucial role of features that endure over time in achieving sustainable learning-based malware detection. Another approach in[85] is to detect and mitigate concept drift in Android malware detection and demonstrate its effectiveness over a seven-year data set. The method also minimizes retraining with short-term datasets, evaluates the impact of timestamps, and characterizes concept drift in Android malware by analyzing essential features across time horizons. While sustainability is not explicitly addressed in the majority of papers, numerous authors examine the robustness of their models by testing them on diverse datasets. This approach, when combined with a reflection on the evolution of malware and model performance following the introduction of newer samples, can establish the foundation for a future study on sustainability [132].

Outdated Datasets. While many papers excel in malware detection or classification tasks using visualization-based approaches, these models are typically assessed using outdated datasets, posing a risk to the model's generalization ability for zero-day malware. While widely used datasets exist, they frequently encompass brief timeframes that might not sufficiently represent malware evolution or possess unevenly distributed samples across various families. Moreover, researchers frequently incorporate samples from various public repositories to either balance or augment the datasets. This practice, however, hampers the efficiency of method comparisons and reproducibility, as the samples vary across each research paper. These factors can significantly impact the performance of detection approaches. Hence, the creation of current and precise datasets that mirror ongoing trends becomes imperative, and we highlight this as a priority for future endeavors.

Computational Costs. The majority of research focuses on techniques for generating static-feature-based images, particularly in greyscale. Although dynamic information visualization holds considerable potential for malware analysis, it comes with the drawback of being computationally intensive. Further efforts to apply hybrid approaches to images could enhance the robustness of these techniques, but this area still requires exploration.

Benchmark Dataset for Feature Fusion Strategies. Though all feature extraction techniques enhance classifier accuracy, the existing literature lacks a well-defined benchmark dataset specifically tailored for applying these techniques. Moreover, a consistent environment with fixed parameters enables unbiased testing and model comparisons. Further exploration into combining local and global features is necessary. Additionally, investigating the fusion of various feature extractors and Deep Learning models warrants attention.

Resource-Constrained IoT devices. In many instances, techniques for analyzing malware are crafted to enhance detection accuracy, often overlooking constraints such as memory footprint, power consumption, and network resources. Regrettably, these operational limitations give rise to notable performance bottlenecks in the context of IoT systems, resulting in a degradation of detection accuracy. For example, complex convolution and pooling operations slow down these techniques when used on edge devices. Extensive research is necessary for visualization-based approaches, which have proven effective for resource-constrained devices like IoT systems.

Interpretability. Using deep models is limited by their lack of interpretability, as they essentially function as black boxes. However, understanding the reasoning behind predictive models is vital, especially in cybersecurity, where analysts must comprehend the security-related decisions made by algorithms. While explainability techniques currently exist to illuminate predictions from deep architectures, there is a need to explore more of these techniques to improve the interpretability of malware predictions through visualization-based malware detection approaches. This is crucial because very few works in this area involve creating methods that clearly explain why specific visual patterns indicate obfuscated behavior. Further research in this direction has the potential to benefit the fields of malware detection and analysis significantly.

Adversarial Attacks. In a broad context, adversaries can target image classifiers by modifying pixel values to create adversarial images. Concerning malware, any alterations to a malware file must not compromise its functionality. The content within executable files is highly sensitive, and even a minor change can completely alter the malware functionality or render the file inoperable. Despite this constraint, adversaries in the visualization-based malware domain have mainly limited themselves to perturbation types, which struggle to maintain the functionality of modified files. Moreover, there are few available mechanisms to verify post-perturbation functionality. Consequently, future research should focus on developing tools that can automatically and efficiently verify the functionality of malware after undergoing perturbations.

Reproducibility. The issue of result replicability is a well-known concern within the research community, and this study's domain is no exception. Daoudi et al. [55] have addressed the critical challenge of reproducibility in Android image-based malware detection. The difficulty in replicating results, even with similar setups, highlights the need for researchers to consider this aspect carefully when aiming to make meaningful and testable contributions to the field. In this assessment, we're determining if the examined papers offer crucial details that can benefit fellow researchers, encompassing specifics on the model's structure, parameters, software components, and more. Guaranteeing thorough documentation of this data is vital to improve the ability to replicate research outcomes and drive progress in image-based malware detection.

10 Conclusion

The proliferation of malware, or malicious software, poses an ongoing and evolving challenge in cybersecurity. In response to the increasing threat landscape, cybersecurity defenses continually evolve to mitigate the impact of malware. Visualization-based detection approaches have emerged as a crucial component of these defense strategies. Adopting visualization-based approaches in malware detection enhances traditional methods by leveraging pictorial representations to uncover intricate relationships and anomalies in large datasets. This paper comprehensively reviews techniques that employ visualization for malware detection. We introduce fundamental knowledge essential for understanding visualization-based methods. Our review categorizes and provides insight into state-of-the-art works, outlining various steps adopted for visualization-based approaches and offering comprehensive descriptions of malware datasets represented as images. This survey also underscores the vulnerability of image-based detection systems to effective adversarial attacks, as attackers aim to deceive these systems. In addition, our survey also discusses the obfuscation detection and sustainability of visualization-based approaches. We began by analyzing 248 works and ultimately included 102 papers published from 2018 to 2025 in our study to reveal the promising future of applying image-based methods to malware detection. As a result, we highlight the lessons learned and suggest future research directions based on current challenges in the field.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Vasundhara Acharya, Vinayakumar Ravi, and Nazeeruddin Mohammad. Efficientnet-based convolutional neural networks for malware classification. In *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pages 1–6, 2021.

[3] Georgios Agrafiotis, Eftychia Makri, Ioannis Flionis, Antonios Lalas, Konstantinos Votis, and Dimitrios Tzovaras. Image-based neural network models for malware traffic classification using pcap to picture conversion. In *Proceedings of the 17th International Conference on Availability, Reliability and Security*, ARES '22, New York, NY, USA, 2022. Association for Computing Machinery.

[4] Mansour Ahmadi, Dmitry Ulyanov, Stanislav Semenov, Mikhail Trofimov, and Giorgio Giacinto. Novel feature extraction, selection and fusion for effective malware family classification. In *Proceedings of the sixth ACM conference on data and application security and privacy*, pages 183–194, 2016.

[5] Mumtaz Ahmed, Neda Afreen, Muneeb Ahmed, Mustafa Sameer, and Jameel Ahamed. An inception v3 approach for malware classification using machine learning and transfer learning. *International Journal of Intelligent Networks*, 4:11–18, 2023.

[6] S. Akarsh, K. Simran, Prabaharan Poornachandran, Vijay Krishna Menon, and K.P. Soman. Deep learning framework and visualization for malware classification. In *2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS)*, pages 1059–1063, 2019.

[7] Inzamamul Alam, Md Samiullah, SM Asaduzzaman, Upama Kabir, AM Aahad, and Simon S Woo. Miracle: Malware image recognition and classification by layered extraction. *Data Mining and Knowledge Discovery*, 39(1):10, 2025.

[8] Tibra Alsmadi and Nour Alqudah. A survey on malware detection techniques. In *2021 International Conference on Information Technology (ICIT)*, pages 371–376, 2021.

[9] Namrata Govind Ambekar, Sonali Samal, N Nandini Devi, and Surmila Thokchom. Fasnet: Federated adversarial siamese networks for robust malware image classification. *Journal of Parallel and Distributed Computing*, 198:105039, 2025.

[10] Hybrid Analysis. Hybrid sandbox. `https://www.hybrid-analysis.com`, 2023.

[11] V Anandhi, P Vinod, and Varun G Menon. Malware visualization and detection using densenets. *Personal and Ubiquitous Computing*, pages 1–17, 2021.

[12] H. S. Anderson and P. Roth. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *ArXiv e-prints*, April 2018.

[13] Omer Aslan Aslan and Refik Samet. A comprehensive review on malware detection approaches. *IEEE Access*, 8:6249–6271, 2020.

[14] KA Asmitha, Vinod Puthuvath, KA Rafidha Rehiman, and SL Ananth. Deep learning vs. adversarial noise: a battle in malware image analysis. *Cluster Computing*, 27(7):9191–9220, 2024.

[15] Pooja Bagane, Susheel George Joseph, Abhishek Singh, Anurag Shrivastava, B. Prabha, and Amit Shrivastava. Classification of malware using deep learning techniques. In *2021 9th International Conference on Cyber and IT Service Management (CITSM)*, pages 1–7, 2021.

[16] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[17] Yude Bai, Zhenchang Xing, Xiaohong Li, Zhiyong Feng, and Duoyuan Ma. Unsuccessful story about few shot malware family classification and siamese network to the rescue. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 1560–1571, 2020.

[18] Khaled Bakour and Halil Murat Ünver. Deepvisdroid: android malware detection by hybridizing image-based features with deep learning techniques. *Neural Computing and Applications*, 33:11499–11516, 2021.

[19] Khaled Bakour and Halil Murat Ünver. Visdroid: Android malware classification based on local and global image features, bag of visual words and machine learning techniques. *Neural Computing and Applications*, 33:3133–3153, 2021.

[20] Halit Bakır and Rezan Bakır. Droidencoder: Malware detection using auto-encoder based feature extractor and machine learning algorithms. *Computers and Electrical Engineering*, 110:108804, 2023.

[21] Irina Baptista, Stavros Shiaeles, and Nicholas Kolokotronis. A novel malware detection system based on machine learning and binary visualization. In *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6, 2019.

[22] Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. Transcending transcend: Revisiting malware classification in the presence of concept drift. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 805–823. IEEE, 2022.

[23] Pedro H Barros, Eduarda TC Chagas, Leonardo B Oliveira, Fabiane Queiroz, and Heitor S Ramos. Malware-smell: A zero-shot learning strategy for detecting zero-day vulnerabilities. *Computers & Security*, 120:102785, 2022.

[24] Ahmed Bensaoud and Jugal Kalita. Deep multi-task learning for malware image classification. *Journal of Information Security and Applications*, 64:103057, 2022.

[25] Aaditya Vikram Saravana Bhavan, Srujana Golla, Yuktha Poral, Alan S Paul, Prasad B Honnavalli, and S Supreetha. Android malware detection: A comprehensive review. *Research Advances in Network Technologies*, pages 41–82, 2024.

[26] Prima Bouchaib and Mohammed Bouhorma. Transfer learning and smote algorithm for image-based malware classification. In *Proceedings of the 4th International Conference on Networking, Information Systems &; Security*, NISS2021, page 1–6, New York, NY, USA, 2021. Association for Computing Machinery.

[27] Kevin W. Bowyer, Nitesh V. Chawla, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *CoRR*, abs/1106.1813, 2011.

[28] Ahmet Selman Bozkir, Ahmet Ogulcan Cankaya, and Murat Aydos. Utilization and comparision of convolutional neural networks in malware recognition. In *2019 27th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4. IEEE, 2019.

[29] Ahmet Selman Bozkir, Ersan Tahillioglu, Murat Aydos, and Ilker Kara. Catch them alive: A malware detection approach through memory forensics, manifold learning and computer vision. *Computers & Security*, 103:102166, 2021.

[30] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[31] A.Z. Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*, pages 21–29, 1997.

[32] Matteo Brosolo, Vinod Puthuvath, Asmitha Ka, Rafidha Rehiman, and Mauro Conti. Sok: Visualization-based malware detection techniques. In *Proceedings of the 19th International Conference on Availability, Reliability and Security*, pages 1–13, 2024.

[33] Roland Burks, Kazi Aminul Islam, Yan Lu, and Jiang Li. Data augmentation with generative models for improved malware detection: A comparative study. In *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pages 0660–0665. IEEE, 2019.

[34] Haipeng Cai. Assessing and improving malware detection sustainability through app evolution studies. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 29(2):1–28, 2020.

[35] Rosangela Casolare, Carlo De Dominicis, Giacomo Iadarola, Fabio Martinelli, Francesco Mercaldo, and Antonella Santone. Dynamic mobile malware detection through system call-based image representation. *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.*, 12(1):44–63, 2021.

[36] Aykut Çayır, Uğur Ünal, and Hasan Dağ. Random capsnet forest model for imbalanced malware type classification task. *Computers & Security*, 102:102133, 2021.

[37] Rajasekhar Chaganti, Vinayakumar Ravi, and Tuan D. Pham. Image-based malware representation approach with efficientnet convolutional neural networks for effective malware classification. *Journal of Information Security and Applications*, 69:103306, 2022.

[38] Rajasekhar Chaganti, Vinayakumar Ravi, and Tuan D. Pham. A multi-view feature fusion approach for effective malware classification using deep learning. *Journal of Information Security and Applications*, 72:103402, 2023.

[39] Yuhan Chai, Jing Qiu, Lihua Yin, Lejun Zhang, Brij B. Gupta, and Zhihong Tian. From data and model levels: Improve the performance of few-shot malware classification. *IEEE Transactions on Network and Service Management*, 19(4):4248–4261, 2022.

[40] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, STOC '02, page 380–388, New York, NY, USA, 2002. Association for Computing Machinery.

[41] Narendrakumar Mangilal Chayal, Ankur Saxena, and Rijwan Khan. A review on spreading and forensics analysis of windows-based ransomware. *Annals of Data Science*, 11(5):1503–1524, 2024.

[42] Chia-Mei Chen, Shi-Hao Wang, Dan-Wei Wen, Gu-Hsin Lai, and Ming-Kung Sun. Applying convolutional neural network for malware detection. In *2019 IEEE 10th International Conference on Awareness Science and Technology (iCAST)*, pages 1–5, 2019.

[43] Li Chen. Deep transfer learning for static malware classification. *arXiv preprint arXiv:1812.07606*, 2018.

[44] Shi Chen, Ying Liu, Wei Hu, Jianyi Liu, Yating Gao, and Bingjie Lin. Malicious code family classification method based on vision transformer. In *2022 IEEE 10th International Conference on Information, Communication and Networks (ICICN)*, pages 704–709, 2022.

[45] Xu Chen, Jon Andersen, Z. Morley Mao, Michael Bailey, and Jose Nazario. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, pages 177–186, 2008.

[46] Sunoh Choi, Sungwook Jang, Youngsoo Kim, and Jonghyun Kim. Malware detection using malware image and deep learning. In *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 1193–1195. IEEE, 2017.

[47] François Chollet et al. Keras. `https://keras.io`, 2015.

[48] Mauro Conti, Shubham Khandhar, and P Vinod. A few-shot malware classification approach for unknown family recognition using malware feature visualization. *Computers & Security*, 122:102887, 2022.

[49] A. Cortesi. binvis.io: Visual analysis of binary files. `http://binvis.io/#/`, 2016. Accessed: 2023-5-15.

[50] Andrew Corum, Donovan Jenkins, and Jun Zheng. Robust pdf malware detection with image visualization and processing techniques. In *2019 2nd International Conference on Data Intelligence and Security (ICDIS)*, pages 108–114. IEEE, 2019.

[51] cuckoosandbox.org. Cuckoo sandbox. `https://cuckoosandbox.org`, 2023.

[52] Zhihua Cui, Fei Xue, Xingjuan Cai, Yang Cao, Gai-ge Wang, and Jinjun Chen. Detection of malicious code variants based on deep learning. *IEEE Transactions on Industrial Informatics*, 14(7):3187–3196, 2018.

[53] Y. Le Cun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, and D. Henderson. *Handwritten Digit Recognition with a Back-Propagation Network*, pages 396—-404. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[54] Yusheng Dai, Hui Li, Yekui Qian, and Xidong Lu. A malware classification method based on memory dump grayscale image. *Digital Investigation*, 27:30–37, 2018.

[55] Kevin Daoudi, Nadia andAllix, Tegawendé F. Bissyandé, and Jacques Klein. Lessons learnt on reproducibility in machine learning based android malware detection. *Empirical Software Engineering*, 26:1573–7616, 2021.

[56] Nadia Daoudi, Jordan Samhi, Abdoul Kader Kabore, Kevin Allix, Tegawendé F Bissyandé, and Jacques Klein. Dexray: a simple, yet effective deep learning approach to android malware detection based on image representation of bytecode. In *Deployable Machine Learning for Security Defense: Second International Workshop, MLHat 2021, Virtual Event, August 15, 2021, Proceedings 2*, pages 81–106. Springer, 2021.

[57] Abdulbasit Darem, Jemal Abawajy, Aaisha Makkar, Asma Alhashmi, and Sultan Alanazi. Visualization and deep-learning-based malware variant detection using opcode-level features. *Future Generation Computer Systems*, 125:314–323, 2021.

[58] Fauzi Mohd Darus, Noor Azurati Ahmad Salleh, and Aswami Fadillah Mohd Ariffin. Android malware detection using machine learning on image patterns. In *2018 Cyber Resilience Conference (CRC)*, pages 1–2, 2018.

[59] Venkata Salini Priyamvada Davuluru, Barath Narayanan Narayanan, and Eric J Balster. Convolutional neural networks as classification tools and feature extractors for distinguishing malware programs. In *2019 IEEE National Aerospace and Electronics Conference (NAECON)*, pages 273–278. IEEE, 2019.

[60] Fatemeh Deldar and Mahdi Abadi. Deep learning for zero-day malware detection and classification: A survey. *ACM Comput. Surv.*, jun 2023. Just Accepted.

[61] Huaxin Deng, Chun Guo, Guowei Shen, Yunhe Cui, and Yuan Ping. Mctvd: A malware classification method based on three-channel visualization and deep learning. *Computers & Security*, 126:103084, 2023.

[62] KA Dhanya, P Vinod, Suleiman Y Yerima, Abul Bashar, Anwin David, T Abhiram, Alan Antony, Ashil K Shavanas, and Gireesh Kumar. Obfuscated malware detection in iot android applications using markov images and cnn. *IEEE Systems Journal*, 17(2):2756–2766, 2023.

[63] Putu Sukma Dharmalaksana, Teddy Mantoro, Lutfil Khakim, and Muchlis Nurseno. Improved malware detection results using visualization-based detection techniques ant convolutional neural network. In *2022 IEEE 8th International Conference on Computing, Engineering and Design (ICCED)*, pages 1–5, 2022.

[64] Mirabelle Dib, Sadegh Torabi, Elias Bou-Harb, and Chadi Assi. A multi-dimensional deep learning framework for iot malware classification and family attribution. *IEEE Transactions on Network and Service Management*, 18(2):1165–1177, 2021.

[65] Huiyao Dong and Igor Kotenko. Image-based malware analysis for enhanced iot security in smart cities. *Internet of Things*, 27:101258, 2024.

[66] Gianni D'Angelo, Massimo Ficco, and Francesco Palmieri. Malware detection in mobile environments based on autoencoders and api-images. *Journal of Parallel and Distributed Computing*, 137:26–33, 2020.

[67] Olorunjube James Falana, Adesina Simon Sodiya, Saidat Adebukola Onashoga, and Biodun Surajudeen Badmus. Mal-detect: An intelligent visualization approach for malware detection. *Journal of King Saud University - Computer and Information Sciences*, 34(5):1968–1983, 2022.

[68] Ming Fan, Wenying Wei, Xiaofei Xie, Yang Liu, Xiaohong Guan, and Ting Liu. Can we trust your explanations? sanity checks for interpreters in android malware analysis. *IEEE Transactions on Information Forensics and Security*, 16:838–853, 2020.

[69] Fergus Fei-Fei. Perona, 2006 fei-fei l., fergus r., perona p. *One-shot learning of object categories, IEEE Trans. Pattern Anal. Mach. Intell*, 28(4):594–611, 2006.

[70] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, page 3–14, New York, NY, USA, 2011. Association for Computing Machinery.

[71] Gil Fidel, Ron Bitton, and Asaf Shabtai. When explainability meets adversarial learning: Detecting adversarial examples using shap signatures. In *2020 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2020.

[72] Jianwen Fu, Jingfeng Xue, Yong Wang, Zhenyan Liu, and Chun Shan. Malware visualization for fine-grained classification. *IEEE Access*, 6:14510–14523, 2018.

[73] ANYRUN FZCO. Any.run. `https://any.run`, 2023.

[74] Tan Gao, Lan Zhao, Xudong Li, and Wen Chen. Malware detection based on semi-supervised learning with malware visualization. *Math. Biosci. Eng*, 18(5):5995–6011, 2021.

[75] Migran N. Gevorkyan, Anastasia V. Demidova, Tatiana S. Demidova, and Anton A. Sobolev. Review and comparative analysis of machine learning libraries for machine learning. *Discrete and Continuous Models and Applied Computational Science*, 27(4):305–315, 2019.

[76] Daniel Gibert, Carles Mateu, and Jordi Planes. Hydra: A multimodal deep learning framework for malware classification. *Computers & Security*, 95:101873, 2020.

[77] Daniel Gibert, Carles Mateu, and Jordi Planes. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, 153:102526, 2020.

[78] Daniel Gibert, Carles Mateu, Jordi Planes, and Ramon Vicens. Using convolutional neural networks for classification of malware represented as images. *Journal of Computer Virology and Hacking Techniques*, 15:15–28, 2019.

[79] Daniel Gibert, Jordi Planes, Carles Mateu, and Quan Le. Fusing feature engineering and deep learning: A case study for malware classification. *Expert Systems with Applications*, 207:117957, 2022.

[80] Jin Ho Go, Tony Jan, Manoranjan Mohanty, Om Prakash Patel, Deepak Puthal, and Mukesh Prasad. Visualization approach for malware classification with resnext. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–7, 2020.

[81] Mahshid Gohari, Sattar Hashemi, and Lida Abdi. Android malware detection and classification based on network traffic using deep learning. In *2021 7th International Conference on Web Research (ICWR)*, pages 71–77. IEEE, 2021.

[82] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks (2014). *arXiv preprint arXiv:1406.2661*, 1406, 2014.

[83] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[84] Simona E Grigorescu, Nicolai Petkov, and Peter Kruizinga. Comparison of texture features based on gabor filters. *IEEE Transactions on Image processing*, 11(10):1160–1167, 2002.

[85] Alejandro Guerra-Manzanares, Marcin Luckner, and Hayretdin Bahsi. Android malware concept drift using system calls: detection, characterization and challenges. *Expert Systems with Applications*, 206:117200, 2022.

[86] Jingcai Guo, Yuanyuan Xu, Wenchao Xu, Yufeng Zhan, Yuxia Sun, and Song Guo. Mdenet: Multi-modal dual-embedding networks for malware open-set recognition, 2023.

[87] Wenbo Guo, Dongliang Mu, Jun Xu, Purui Su, Gang Wang, and Xinyu Xing. Lemna: Explaining deep learning based security applications. In *proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 364–379, 2018.

[88] Robert M Haralick, Karthikeyan Shanmugam, and Its' Hak Dinstein. Textural features for image classification. *IEEE Transactions on systems, man, and cybernetics*, pages 610–621, 1973.

[89] Chihiro Hasegawa and Hitoshi Iyatomi. One-dimensional convolutional neural networks for android malware detection. In *2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA)*, pages 99–102. IEEE, 2018.

[90] Hashem Hashemi and Ali Hamzeh. Visual malware detection using local malicious pattern. *Journal of Computer Virology and Hacking Techniques*, 15:1–14, 2019.

[91] Hashem Hashemi, Mohammad Ebrahim Samie, and Ali Hamzeh. Ifmd: image fusion for malware detection. *Journal of Computer Virology and Hacking Techniques*, 19(2):271–286, 2023.

[92] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[93] Shou-Ching Hsiao, Da-Yu Kao, Zi-Yuan Liu, and Raylin Tso. Malware image classification using one-shot learning with siamese networks. *Procedia Computer Science*, 159:1863–1871, 2019. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 23rd International Conference KES2019.

[94] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[95] Xiang Huang, Li Ma, Wenyin Yang, and Yong Zhong. A method for windows malware detection based on deep learning. *Journal of Signal Processing Systems*, 93:265–273, 2021.

[96] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[97] Giacomo Iadarola, Fabio Martinelli, Francesco Mercaldo, and Antonella Santone. Towards an interpretable deep learning model for mobile malware detection and family identification. *Computers & Security*, 105:102198, 2021.

[98] Paul Irolla and Alexandre Dey. The duplication issue within the drebin dataset. *Journal of Computer Virology and Hacking Techniques*, 14(3):245–249, 2018.

[99] Marc Jacob. Microsoft digital defense report. `https://news.microsoft.com/en-cee/2024/11/29/microsoft-digital-defense-report-600-million-cyberattacks-per-day-around-the-globe-2/`, 2025. Accessed: 2025-5-3.

[100] Mugdha Jain, William Andreopoulos, and Mark Stamp. Convolutional neural networks and extreme learning machines for malware classification. *Journal of Computer Virology and Hacking Techniques*, 16:pages 229–244, 2020.

[101] Yifei Jian, Hongbo Kuang, Chenglong Ren, Zicheng Ma, and Haizhou Wang. A novel framework for image-based malware detection with a deep neural network. *Computers & Security*, 109:102400, 2021.

[102] Jiaqi Jiang and Yunchun Zhang. A pyramid stripe pooling-based convolutional neural network for malware detection and classification. *Journal of Ambient Intelligence and Humanized Computing*, 14(3):2785–2796, Mar 2023.

[103] Jeonggeun Jo, Jaeik Cho, and Jongsub Moon. A malware detection and extraction method for the related information using the vit attention mechanism on android operating system. *Applied Sciences*, 13(11):6839, 2023.

[104] joesandbox.com. Joe sandbox. `https://www.joesandbox.com/#windows`, 2023.

[105] Jahez Abraham Johny, KA Asmitha, P Vinod, G Radhamani, KA Rafidha Rehiman, and Mauro Conti. Deep learning fusion for effective malware detection: leveraging visual features. *Cluster Computing*, 28(2):135, 2025.

[106] Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. Transcend: Detecting concept drift in malware classification models. In *26th USENIX security symposium (USENIX security 17)*, pages 625–642, 2017.

[107] 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.

[108] Evanson Mwangi Karanja, Shedden Masupe, and Mandu Gasennelwe Jeffrey. Analysis of internet of things malware using image texture features and machine learning techniques. *Internet of Things*, 9:100153, 2020.

[109] ElMouatez Billah Karbab, Mourad Debbabi, and Abdelouahid Derhab. Swiftr: Cross-platform ransomware fingerprinting using hierarchical neural networks on hybrid features. *Expert Systems with Applications*, 225:120017, 2023.

[110] Kevin Kerr. Trustwave's 2025 cybersecurity predictions: Ai-powered attacks, critical infrastructure risks, and regulatory challenges, 2025. Accessed: 2025-05-03.

[111] David Sean Keyes, Beiqi Li, Gurdip Kaur, Arash Habibi Lashkari, Francois Gagnon, and Frédéric Massicotte. Entroplyzer: Android malware classification and characterization using entropy analysis of dynamic characteristics. In *2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS)*, pages 1–12. IEEE, 2021.

[112] Aminollah Khormali, Ahmed Abusnaina, Songqing Chen, DaeHun Nyang, and Aziz Mohaisen. Copycat: practical adversarial attacks on visualization-based malware detection. *arXiv preprint arXiv:1909.09735*, 2019.

[113] Jeongwoo Kim, Joon-Young Paik, and Eun-Sun Cho. Attention-based cross-modal cnn using non-disassembled files for malware classification. *IEEE Access*, 11:22889–22903, 2023.

[114] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[115] Ajit Kumar, K Pramod Sagar, KS Kuppusamy, and G Aghila. Machine learning based malware classification for android applications using multimodal image representations. In *2016 10th international conference on intelligent systems and control (ISCO)*, pages 1–6. IEEE, 2016.

[116] Nitish Kumar and Toshanlal Meenpal. Texture-based malware family classification. In *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–6. IEEE, 2019.

[117] Rajesh Kumar, Zhang Xiaosong, Riaz Ullah Khan, Ijaz Ahad, and Jay Kumar. Malicious code detection based on image processing using deep learning. In *Proceedings of the 2018 International Conference on Computing and Artificial Intelligence*, ICCAI 2018, page 81–85, New York, NY, USA, 2018. Association for Computing Machinery.

[118] Sanjeev Kumar and B. Janet. Dtmic: Deep transfer learning for malware image classification. *Journal of Information Security and Applications*, 64:103063, 2022.

[119] Sanjeev Kumar, B Janet, and Subramanian Neelakantan. Identification of malware families using stacking of textural features and machine learning. *Expert Systems with Applications*, 208:118073, 2022.

[120] Sanjeev Kumar, B Janet, and Subramanian Neelakantan. Imcnn: Intelligent malware classification using deep convolution neural networks as transfer learning and ensemble learning in honeypot enabled organizational network. *Computer Communications*, 216:16–33, 2024.

[121] Malicia Lab. Avclass. `https://github.com/malicialab/avclass`, 2025. Accessed: 2023-5-15.

[122] Nada Lachtar, Duha Ibdah, Hamza Khan, and Anys Bacha. Ransomshield: A visualization approach to defending mobile systems against ransomware. *ACM Trans. Priv. Secur.*, 26(3), mar 2023.

[123] Sumit S Lad, Amol C Adamuthe, et al. Malware classification with improved convolutional neural network model. *International Journal of Computer Network & Information Security*, 12(6):30–43, 2020.

[124] Quan Le, Oisín Boydell, Brian Mac Namee, and Mark Scanlon. Deep learning at the shallow end: Malware classification for non-domain experts. *Digital Investigation*, 26:S118–S126, 2018.

[125] Hyunjong Lee, Sooin Kim, Dongheon Baek, Donghoon Kim, and Doosung Hwang. Robust iot malware detection and classification using opcode category features on machine learning. *IEEE Access*, 11:18855–18867, 2023.

[126] Jongkwan Lee and Jongdeog Lee. A classification system for visualized malware based on multiple autoencoder models. *IEEE Access*, 9:144786–144795, 2021.

[127] Ahmed Lekssays, Bouchaib Falah, and Sameer Abufardeh. A novel approach for android malware detection and classification using convolutional neural networks. In *15th International Conference on Software Technologies*, pages 606–614, 01 2020.

[128] Chaoran Li, Xiao Chen, Derui Wang, Sheng Wen, Muhammad Ejaz Ahmed, Seyit Camtepe, and Yang Xiang. Backdoor attack on machine learning based android malware detectors. *IEEE Transactions on dependable and secure computing*, 19(5):3357–3370, 2021.

[129] Qi Li, Jiaxin Mi, Weishi Li, Junfeng Wang, and Mingyu Cheng. Cnn-based malware variants detection method for internet of things. *IEEE Internet of Things Journal*, 8(23):16946–16962, 2021.

[130] Yuzhou Lin and Xiaolin Chang. Towards interpretable ensemble learning for image-based malware detection. *arXiv preprint arXiv:2101.04889*, 2021.

[131] Xiang Ling, Lingfei Wu, Jiangyu Zhang, Zhenqing Qu, Wei Deng, Xiang Chen, Yaguan Qian, Chunming Wu, Shouling Ji, Tianyue Luo, et al. Adversarial attacks against windows pe malware detection: A survey of the state-of-the-art. *Computers & Security*, page 103134, 2023.

[132] Xinbo Liu, Yaping Lin, He Li, and Jiliang Zhang. A novel method for malware detection on ml-based visualization technique. *Computers & Security*, 89:101682, 2020.

[133] Xinbo Liu, Jiliang Zhang, Yaping Lin, and He Li. Atmpa: attacking machine learning-based malware visualization detection methods via adversarial examples. In *Proceedings of the International Symposium on Quality of Service*, pages 1–10, 2019.

[134] Ya-shu Liu, Yu-Kun Lai, Zhi-Hai Wang, and Han-Bing Yan. A new learning approach to malware classification using discriminative feature extraction. *IEEE Access*, 7:13015–13023, 2019.

[135] D.G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer vision*, volume 2, pages 1150–1157 vol.2, 1999.

[136] Yan Lu and Jiang Li. Generative adversarial network for improving deep learning based malware classification. In *2019 Winter Simulation Conference (WSC)*, pages 584–593. IEEE, 2019.

[137] Jhu-Sin Luo and Dan Chia-Tien Lo. Binary malware image classification using machine learning with local binary pattern. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 4664–4667. IEEE, 2017.

[138] Ahmad M., Ahmed Ebada, and Aya Zoghby. A survey on visualization-based malware detection. *Journal of Cyber Security*, 4, 11 2022.

[139] Gopinath M. and Sibi Chakkaravarthy Sethuraman. A comprehensive survey on deep learning based malware detection techniques. *Computer Science Review*, 47:100529, 2023.

[140] Zhidong Ma, Zhiwei Zhang, Chengliang Liu, Tianzhu Hu, Hongjun Li, and Baoquan Ren. Visualizable malware detection based on multi-dimension dynamic behaviors. In *2022 International Conference on Networking and Network Applications (NaNA)*, pages 247–252, 2022.

[141] Arash Mahboubi, Seyit Camtepe, and Keyvan Ansari. Stochastic modeling of iot botnet spread: A short survey on mobile malware spread modeling. *IEEE Access*, 8:228818–228830, 2020.

[142] Aziz Makandar and Anita Patrot. Malware analysis and classification using artificial neural network. In *2015 International conference on trends in automation, communications and computing technology (I-TACT-15)*, pages 1–6. IEEE, 2015.

[143] Abhishek Mallik, Anavi Khetarpal, and Sanjay Kumar. Conrec: malware classification using convolutional recurrence. *Journal of Computer Virology and Hacking Techniques*, 18:297–313, 2022.

[144] Sherin Mary Mathews. Explainable artificial intelligence applications in nlp, biomedical, and malware classification: A literature review. In Kohei Arai, Rahul Bhatia, and Supriya Kapoor, editors, *Intelligent Computing*, pages 1269–1292, Cham, 2019. Springer International Publishing.

[145] Zhaoyi Meng, Jiale Zhang, Jiaqi Guo, Wansen Wang, Wenchao Huang, Jie Cui, Hong Zhong, and Yan Xiong. Detecting android malware by visualizing app behaviors from multiple complementary views. *IEEE Transactions on Information Forensics and Security*, 2025.

[146] Francesco Mercaldo and Antonella Santone. Deep learning for image-based mobile malware detection. *Journal of Computer Virology and Hacking Techniques*, 16:157 – 171, 2020.

[147] Tajuddin Manhar Mohammed, Lakshmanan Nataraj, Satish Chikkagoudar, Shivkumar Chandrasekaran, and BS Manjunath. Malware detection using frequency domain-based image visualization and deep learning. *arXiv preprint arXiv:2101.10578*, 2021.

[148] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[149] Caio C. Moreira, Davi C. Moreira, and Claudomiro de S. de Sales Jr. Improving ransomware detection based on portable executable header using xception convolutional neural network. *Computers & Security*, 130:103265, 2023.

[150] Hamad Naeem, Bandar M Alshammari, and Farhan Ullah. Explainable artificial Intelligence-Based IoT device malware detection mechanism using image visualization and Fine-Tuned CNN-Based transfer learning model. *Comput Intell Neurosci*, 2022:7671967, July 2022.

[151] Hamad Naeem, Amjad Alsirhani, Mohammed Mujib Alshahrani, and Abdullah Alomari. Android device malware classification framework using multistep image feature extraction and multihead deep neural ensemble. *Traitement du Signal*, 39(3), 2022.

[152] Hamad Naeem, Bing Guo, Muhammad Rashid Naeem, Farhan Ullah, Hamza Aldabbas, and Muhammad Sufyan Javed. Identification of malicious code variants based on image visualization. *Computers & Electrical Engineering*, 76:225–237, 2019.

[153] Hamad Naeem, Bing Guo, Muhammad Rashid Naeem, and Danish Vasan. Visual malware classification using local and global malicious pattern. *Journal of Computers*, 30(6):73–83, 2019.

[154] Hamad Naeem, Farhan Ullah, Muhammad Rashid Naeem, Shehzad Khalid, Danish Vasan, Sohail Jabbar, and Saqib Saeed. Malware detection in industrial internet of things based on hybrid image visualization and deep learning model. *Ad Hoc Networks*, 105:102154, 2020.

[155] Shivarti Naik and Amita Dessai. Malware classification approaches using machine learning techniques: A review. In *2021 5th International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques (ICEECCOT)*, pages 111–117, 2021.

[156] Antonio Nappa, M Zubair Rafique, and Juan Caballero. The malicia dataset: identification and analysis of drive-by download operations. *International Journal of Information Security*, 14, 02 2014.

[157] 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.

[158] Quoc-Dung Ngo, Huy-Trung Nguyen, Van-Hoang Le, and Doan-Hieu Nguyen. A survey of iot malware and detection methods based on static features. *ICT Express*, 6(4):280–286, 2020.

[159] Sang Ni, Quan Qian, and Rui Zhang. Malware identification using visualization images and deep learning. *Computers & Security*, 77:871–885, 2018.

[160] Timo Ojala, Matti Pietikainen, and David Harwood. Performance evaluation of texture measures with classification based on kullback discrimination of distributions. In *Proceedings of 12th international conference on pattern recognition*, volume 1, pages 582–585. IEEE, 1994.

[161] Stephen O'Shaughnessy and Stephen Sheridan. Image-based malware classification hybrid framework based on space-filling curves. *Computers & Security*, 116:102660, 2022.

[162] Cornelius Paardekooper, Nasimul Noman, Raymond Chiong, and Vijay Varadharajan. Designing deep convolutional neural networks using a genetic algorithm for image-based malware classification. In *2022 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2022.

[163] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*, pages 372–387. IEEE, 2016.

[164] Daniel Park, Haidar Khan, and Bülent Yener. Generation & evaluation of adversarial examples for malware obfuscation. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 1283–1290. IEEE, 2019.

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

[166] Dev Patel, Dhairya Jadav, Rajesh Gupta, Nilesh Kumar Jadav, Sudeep Tanwar, Bassem Ouni, and Mohsen Guizani. Deep learning and blockchain-based framework to detect malware in autonomous vehicles. In *2022 International Wireless Communications and Mobile Computing (IWCMC)*, pages 278–283. IEEE, 2022.

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

[168] Anson Pinhero, Anupama M L, Vinod P, C.A. Visaggio, Aneesh N, Abhijith S, and AnanthaKrishnan S. Malware detection employed by visualization and deep neural network. *Computers & Security*, 105:102247, 2021.

[169] Handhika Yanuar Pratama and Jeckson Sidabutar. Malware classification and visualization using efficientnet and b2img algorithm. In *2022 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pages 75–80, 2022.

[170] Sahaya Beni Prathiba, Pranav Murali, Rajalakshmi Shenbaga Moorthy, Deepak Kumar Anandhan, Arikumar K Selvaraj, and Joel JPC Rodrigues. A blockchain-powered malicious node detection in internet of autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 2024.

[171] Yanchen Qiao, Qingshan Jiang, Zhenchao Jiang, and Liang Gu. A multi-channel visualization method for malware classification based on deep learning. In *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 757–762, 2019.

[172] Lingfeng Qiu, Shuo Wang, Jian Wang, Yifei Wang, and Wei Huang. Malware classification based on a lightweight architecture of cnn: Malshufflenet. In *2022 3rd International Conference on Computer Vision, Image and Deep Learning & International Conference on Computer Engineering and Applications (CVIDL & ICCEA)*, pages 1047–1050, 2022.

[173] Akshara Ravi, Vivek Chaturvedi, and Muhammad Shafique. Vit4mal: Lightweight vision transformer for malware detection on edge devices. *ACM Transactions on Embedded Computing Systems*, 22(5s):1–26, 2023.

[174] Vinayakumar Ravi and Rajasekhar Chaganti. Efficientnet deep learning meta-classifier approach for image-based android malware detection. *Multimedia Tools and Applications*, 82:24891–24917, 2023.

[175] John Reiser. MS Windows NT kernel description. `https://github.com/upx/upx`, 2025.

[176] Zhongru Ren, Haomin Wu, Qian Ning, Iftikhar Hussain, and Bingcai Chen. End-to-end malware detection for android iot devices using deep learning. *Ad Hoc Networks*, 101:102098, 2020.

[177] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. Automatic analysis of malware behavior using machine learning. *J. Comput. Secur.*, 19(4):639–668, dec 2011.

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

[179] Candong Rong, Gaopeng Gou, Chengshang Hou, Zhen Li, Gang Xiong, and Li Guo. Umvd-fsl: unseen malware variants detection using few-shot learning. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.

[180] S. Abijah Roseline, S. Geetha, Seifedine Kadry, and Yunyoung Nam. Intelligent vision-based malware detection and classification using deep random forest paradigm. *IEEE Access*, 8:206303–206324, 2020.

[181] S Abijah Roseline, AD Sasisri, S Geetha, and C Balasubramanian. Towards efficient malware detection and classification using multilayered random forest ensemble technique. In *2019 International Carnahan Conference on Security Technology (ICCST)*, pages 1–6. IEEE, 2019.

[182] Ishai Rosenberg, Asaf Shabtai, Yuval Elovici, and Lior Rokach. Adversarial machine learning attacks and defense methods in the cyber security domain. *ACM Computing Surveys (CSUR)*, 54(5):1–36, 2021.

[183] Furqan Rustam, Imran Ashraf, Anca Delia Jurcut, Ali Kashif Bashir, and Yousaf Bin Zikria. Malware detection using image representation of malware data and transfer learning. *Journal of Parallel and Distributed Computing*, 172:32–50, 2023.

[184] Giorgio Severi, Jim Meyer, Scott Coull, and Alina Oprea. {Explanation-Guided} backdoor poisoning attacks against malware classifiers. In *30th USENIX security symposium (USENIX security 21)*, pages 1487–1504, 2021.

[185] Asaf Shabtai, Robert Moskovitch, Yuval Elovici, and Chanan Glezer. Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. *information security technical report*, 14(1):16–29, 2009.

[186] Syed Shakir Hameed Shah, Norziana Jamil, and Atta ur Rehman Khan. Performance comparison of visualization-based malware detection and classification techniques. In *2022 17th International Conference on Emerging Technologies (ICET)*, pages 200–205, 2022.

[187] Claude Elwood Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 1948.

[188] Osho Sharma, Akashdeep Sharma, and Arvind Kalia. Windows and iot malware visualization and classification with deep cnn and xception cnn using markov images. *Journal of Intelligent Information Systems*, pages 1–27, 2022.

[189] Osho Sharma, Akashdeep Sharma, and Arvind Kalia. Migan: Gan for facilitating malware image synthesis with improved malware classification on novel dataset. *Expert Systems with Applications*, 241:122678, 2024.

[190] Kamran Shaukat, Suhuai Luo, and Vijay Varadharajan. A novel deep learning-based approach for malware detection. *Engineering Applications of Artificial Intelligence*, 122:106030, 2023.

[191] Limin Shen, Jiayin Feng, Zhen Chen, Zhongkui Sun, Dongkui Liang, Hui Li, and Yuying Wang. Self-attention based convolutional-lstm for android malware detection using network traffics grayscale image. *Applied Intelligence*, 53(1):683–705, 2023.

[192] Jaiteg Singh, Deepak Thakur, Tanya Gera, Babar Shah, Tamer Abuhmed, and Farman Ali. Classification and analysis of android malware images using feature fusion technique. *IEEE Access*, 9:90102–90117, 2021.

[193] Shruti Singh, Divya Srivastava, and Suneeta Agarwal. Glcm and its application in pattern recognition. In *2017 5th International Symposium on Computational and Business Intelligence (ISCBI)*, pages 20–25, 2017.

[194] Hispasec Sistemas. Virus total. `https://www.virustotal.com`, 2023.

[195] Shiva Darshan S.L and Jaidhar C.D. Windows malware detector using convolutional neural network based on visualization images. *IEEE Transactions on Emerging Topics in Computing*, 9(2):1057–1069, 2021.

[196] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28, 2015.

[197] I. Stančin and A. Jović. An overview and comparison of free python libraries for data mining and big data analysis. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 977–982, 2019.

[198] Jiawei Su, Danilo Vargas Vasconcellos, Sanjiva Prasad, Daniele Sgandurra, Yaokai Feng, and Kouichi Sakurai. Lightweight classification of iot malware based on image recognition. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, volume 02, pages 664–669, 2018.

[199] Sudhakar and Sushil Kumar. Mcft-cnn: Malware classification with fine-tune convolution neural networks using traditional and transfer learning in internet of things. *Future Generation Computer Systems*, 125:334–351, 2021.

[200] Guosong Sun and Quan Qian. Deep learning and visualization for identifying malware families. *IEEE Transactions on Dependable and Secure Computing*, 18(1):283–295, 2021.

[201] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[202] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019.

[203] Mingdong Tang and Quan Qian. Dynamic api call sequence visualization for malware classification. *IET Information Security*, 13, 07 2019.

[204] Adem Tekerek and Muhammed Mutlu Yapici. A novel malware classification and augmentation model based on convolutional neural network. *Computers & Security*, 112:102515, 2022.

[205] Virus Total. Virus total global statistics. `https://www.virustotal.com/gui/stats`, 2023.

[206] Turker Tuncer, Fatih Ertam, and Sengul Dogan. Automated malware recognition method based on local neighborhood binary pattern. *Multimedia Tools and Applications*, 79:27815–27832, 2020.

[207] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. Survey of machine learning techniques for malware analysis. *Computers & Security*, 81:123–147, 2019.

[208] P Umesh. Image processing in python. *CSI Communications*, 23, 2012.

[209] Halil Murat Ünver and Khaled Bakour. Android malware detection based on image-based features and machine learning techniques. *SN Applied Sciences*, 2:1–15, 2020.

[210] Tuan Van Dao, Hiroshi Sato, and Masao Kubo. An attention mechanism for combination of cnn and vae for image-based malware classification. *IEEE Access*, 10:85127–85136, 2022.

[211] 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.

[212] Danish Vasan, Mamoun Alazab, Sobia Wassan, Babak Safaei, and Qin Zheng. Image-based malware classification using ensemble of cnn architectures (imcec). *Computers & Security*, 92:101748, 2020.

[213] Danish Vasan, Mohammad Hammoudeh, and Mamoun Alazab. Broad learning: A gpu-free image-based malware classification. *Applied Soft Computing*, 154:111401, 2024.

[214] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[215] Sitalakshmi Venkatraman, Mamoun Alazab, and R. Vinayakumar. A hybrid deep learning image-based analysis for effective malware detection. *Journal of Information Security and Applications*, 47:377–389, 2019.

[216] Vinita Verma, Sunil K Muttoo, and VB Singh. Multiclass malware classification via first-and second-order texture statistics. *Computers & Security*, 97:101895, 2020.

[217] R. Vinayakumar, Mamoun Alazab, K. P. Soman, Prabaharan Poornachandran, and Sitalakshmi Venkatraman. Robust intelligent malware detection using deep learning. *IEEE Access*, 7:46717–46738, 2019.

[218] R Vinayakumar, Prabaharan Poornachandran, and KP Soman. Scalable framework for cyber threat situational awareness based on domain name systems data analysis. *Big data in engineering applications*, pages 113–142, 2018.

[219] VirusTotal. Yara rules. `https://virustotal.github.io/yara`, 2025.

[220] Paul Voigt and Axel Von dem Bussche. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10(3152676):10–5555, 2017.

[221] Duc-Ly Vu, Trong-Kha Nguyen, Tam V. Nguyen, Tu N. Nguyen, Fabio Massacci, and Phu H. Phung. Hit4mal: Hybrid image transformation for malware classification. *Trans. Emerg. Telecommun. Technol.*, 31(11), nov 2020.

[222] Bidong Wang, Hui Liu, Xinli Han, and Dongliang Xuan. Image-based ransomware classification with classifier combination. In *Proceedings of the 3rd International Conference on Advanced Information Science and System*, AISS '21, New York, NY, USA, 2022. Association for Computing Machinery.

[223] Peng Wang, Zhijie Tang, and Junfeng Wang. A novel few-shot malware classification approach for unknown family recognition with multi-prototype modeling. *Computers & Security*, 106:102273, 2021.

[224] David Warde-Farley and Ian Goodfellow. 11 adversarial perturbations of deep neural networks. *Perturbations, Optimization, and Statistics*, 311:5, 2016.

[225] Bozhi Wu, Sen Chen, Cuiyun Gao, Lingling Fan, Yang Liu, Weiping Wen, and Michael R Lyu. Why an android app is classified as malware: Toward malware classification interpretation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30(2):1–29, 2021.

[226] Qixin Wu, Zheng Qin, Jinxin Zhang, Hui Yin, Guangyi Yang, and Kuangsheng Hu. Android malware detection using local binary pattern and principal component analysis. In *Data Science: Third International Conference of Pioneering Computer Scientists, Engineers and Educators, ICPCSEE 2017, Changsha, China, September 22–24, 2017, Proceedings, Part I*, pages 262–275. Springer, 2017.

[227] Guoqing Xiao, Jingning Li, Yuedan Chen, and Kenli Li. Malfcs: An effective malware classification framework with automated feature extraction based on deep convolutional neural networks. *Journal of Parallel and Distributed Computing*, 141:49–58, 2020.

[228] Mao Xiao, Chun Guo, Guowei Shen, Yunhe Cui, and Chaohui Jiang. Image-based malware classification using section distribution information. *Computers & Security*, 110:102420, 2021.

[229] Xusheng Xiao and Shao Yang. An image-inspired and cnn-based android malware detection approach. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1259–1261, 2019.

[230] Peng Xu, Claudia Eckert, and Apostolis Zarras. hybrid-flacon: Hybrid pattern malware detection and categorization with network traffic and program code. *arXiv preprint arXiv:2112.10035*, 2021.

[231] Bona Xuan, Jin Li, and Yafei Song. Bitcn-taefficientnet malware classification approach based on sequence and rgb fusion. *Computers & Security*, 139:103734, 2024.

[232] Sravani Yajamanam, Vikash Raja Samuel Selvin, Fabio Di Troia, and Mark Stamp. Deep learning versus gist descriptors for image-based malware classification. In *Icissp*, pages 553–561, 2018.

[233] Hiromu Yakura, Shinnosuke Shinozaki, Reon Nishimura, Yoshihiro Oyama, and Jun Sakuma. Malware analysis of imaged binary samples by convolutional neural network with attention mechanism. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, CODASPY '18, page 127–134, New York, NY, USA, 2018. Association for Computing Machinery.

[234] Hiromu Yakura, Shinnosuke Shinozaki, Reon Nishimura, Yoshihiro Oyama, and Jun Sakuma. Neural malware analysis with attention mechanism. *Computers & Security*, 87:101592, 2019.

[235] Senming Yan, Jing Ren, Wei Wang, Limin Sun, Wei Zhang, and Quan Yu. A survey of adversarial attack and defense methods for malware classification in cyber security. *IEEE Communications Surveys & Tutorials*, 25(1):467–496, 2022.

[236] Chun Yang, Yu Wen, Jianbin Guo, Haitao Song, Linfeng Li, Haoyang Che, and Dan Meng. A convolutional neural network based classifier for uncompressed malware samples. In *Proceedings of the 1st Workshop on Security-Oriented Designs of Computer Architectures and Processors*, SecArch'18, page 15–17, New York, NY, USA, 2018. Association for Computing Machinery.

[237] Hangfeng Yang, Shudong Li, Xiaobo Wu, Hui Lu, and Weihong Han. A novel solutions for malicious code detection and family clustering based on machine learning. *IEEE Access*, 7:148853–148860, 2019.

[238] Jin Yang, Huijia Liang, Hang Ren, Dongqing Jia, and Xin Wang. Sac: Collaborative learning of structure and content features for android malware detection framework. *Neurocomputing*, page 130053, 2025.

[239] Limin Yang, Zhi Chen, Jacopo Cortellazzi, Feargus Pendlebury, Kevin Tu, Fabio Pierazzi, Lorenzo Cavallaro, and Gang Wang. Jigsaw puzzle: Selective backdoor attack to subvert malware classifiers. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 719–736. IEEE, 2023.

[240] Limin Yang, Arridhana Ciptadi, Ihar Laziuk, Ali Ahmadzadeh, and Gang Wang. Bodmas: An open dataset for learning based temporal analysis of pe malware. In *4th Deep Learning and Security Workshop*, 2021.

[241] Shumian Yang, Jiarui Hu, Xin Li, Dawei Zhao, Lijuan Xu, Fuqiang Yu, and Chunhui Wang. A variant-sensitive malware detection method based on feature contrast enhancement. *IEEE Transactions on Computational Social Systems*, 2025.

[242] Ilsun You and Kangbin Yim. Malware obfuscation techniques: A brief survey. In *2010 International conference on broadband, wireless computing, communication and applications*, pages 297–300. IEEE, 2010.

[243] Yaoxiang Yu, Bo Cai, Kamran Aziz, Xinyan Wang, Jian Luo, Muhammad Shahid Iqbal, Prasun Chakrabarti, and Tulika Chakrabarti. Semantic lossless encoded image representation for malware classification. *Scientific Reports*, 15(1):7997, 2025.

[244] Baoguo Yuan, Junfeng Wang, Dong Liu, Wen Guo, Peng Wu, and Xuhua Bao. Byte-level malware classification based on markov images and deep learning. *Computers & Security*, 92:101740, 2020.

[245] Dazhi Zhan, Yexin Duan, Yue Hu, Lujia Yin, Zhisong Pan, and Shize Guo. Amgmal: Adaptive mask-guided adversarial attack against malware detection with minimal perturbation. *Computers & Security*, 127:103103, 2023.

[246] Dazhi Zhan, Kun Xu, Xin Liu, Tong Han, Zhisong Pan, and Shize Guo. Practical clean-label backdoor attack against static malware detection. *Computers & Security*, 150:104280, 2025.

[247] Dandan Zhang, Yafei Song, Qian Xiang, and Yang Wang. Imcmk-cnn: A lightweight convolutional neural network with multi-scale kernels for image-based malware classification. *Alexandria Engineering Journal*, 111:203–220, 2025.

[248] Wenhui Zhang, Nurbol Luktarhan, Chao Ding, and Bei Lu. Android malware detection using tcn with bytecode image. *Symmetry*, 13(7):1107, 2021.

[249] Jiawei Zhao, Rahat Masood, and Suranga Seneviratne. A review of computer vision methods in network security. *IEEE Communications Surveys & Tutorials*, 23(3):1838–1878, 2021.

[250] Yuntao Zhao, Wenjie Cui, Shengnan Geng, Bo Bo, Yongxin Feng, and Wenbo Zhang. A malware detection method of code texture visualization based on an improved faster rcnn combining transfer learning. *IEEE Access*, 8:166630–166641, 2020.

[251] Fangtian Zhong, Zekai Chen, Minghui Xu, Guoming Zhang, Dongxiao Yu, and Xiuzhen Cheng. Malware-on-the-brain: Illuminating malware byte codes with images for malware classification. *IEEE Transactions on Computers*, 72(2):438–451, 2022.

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

[253] Jinting Zhu, Julian Jang-Jaccard, Amardeep Singh, Ian Welch, Harith AI-Sahaf, and Seyit Camtepe. A few-shot meta-learning based siamese neural network using entropy features for ransomware classification. *Computers & Security*, 117:102691, 2022.