MALVIS: A LARGE-SCALE IMAGE-BASED FRAMEWORK AND DATASET FOR ADVANCING ANDROID MALWARE CLASSIFICATION

Saleh J. Makkawy
Department of Electrical and Computer Engineering
University of Delaware
Newark, DE 19711
salehmak@udel.edu

Michael J. De Lucia
Department of Electrical and Computer Engineering
University of Delaware
Newark, DE 19711
mdelucia@udel.edu

Kenneth E. Barner
Department of Electrical and Computer Engineering
University of Delaware
Newark, DE 19711
barner@udel.edu

June 9, 2025

ABSTRACT

As technology advances, developers continually create innovative solutions to enhance smartphone security. However, the rapid spread of Android malware poses significant threats to devices and sensitive data. The Android Operating System (OS) 's open-source nature and Software Development Kit (SDK) availability mainly contribute to this alarming growth. Conventional malware detection methods, such as signature-based, static, and dynamic analysis, face challenges in detecting obfuscated techniques such as encryption, packing, and compression in malware. Although developers have created several visualization techniques for malware detection using deep learning (DL), they often fail to identify the critical malicious features of malware accurately. This research introduces MalVis, a unified visualization framework that integrates entropy and N-gram analysis to emphasize meaningful structural and anomalous operational patterns within the malware bytecode. By addressing significant limitations of existing visualization methods, such as insufficient feature representation, limited interpretability, small dataset sizes, and restricted data access, MalVis delivers enhanced detection capabilities, particularly for obfuscated and previously unseen (zero-day) malware. The framework leverages the MalVis dataset introduced in this work, a publicly available large-scale dataset comprising more than 1.3 million visual representations in nine malware classes and one benign class. A comprehensive comparative evaluation was performed against existing state-of-the-art visualization techniques using leading convolutional neural network (CNN) architectures, MobileNet-V2, DenseNet201, ResNet50, and Inception-V3. To further boost classification performance and mitigate overfitting, the outputs of these models were combined using eight distinct ensemble strategies. To reduce the problem of an imbalanced class distribution in the multiclass dataset, we implemented an undersampling technique to ensure balanced learning across all types of malware. MalVis achieved superior results, with 95.19% accuracy, 90.81% F1-score, 92.58% precision, 89.10% recall, 87.58% Matthews Correlation Coefficient (MCC), and 98.06% Receiver Operating Characteristic Area Under Curve (ROC-AUC). These findings emphasize the effectiveness of MalVis in providing interpretable, accurate representation features for malware detection and classification, offering a valuable resource for both research and real-world security applications.

1 Introduction

Smartphones are proliferating, with projections indicating that they will exceed 7 billion by 2025, and nearly 70% using the Android operating system [1] [2]. Due to their compact designs, these mobile devices have become indispensable, facilitating tasks like email management, banking transactions, and the storage of sensitive health information. However, the widespread adoption of smartphones has also drawn the attention of hackers [3], exacerbated by the open-source nature of Android's OS and its SDK. This vulnerability has facilitated the way for various forms of malware, including viruses [4], [5], worms [6], adware [7, 8], spyware [9], ransomware [10], rootkits [11], trojans [12], keyloggers [13], botnets [14], and mobileware [15]. Consequently, developing a robust defense system capable of identifying and mitigating this wide range of threats is crucial. While traditional detection methods like signature-based [16, 17], dynamic analysis [18], and static analysis [19] remain dominant, they often struggle with modern evasion techniques such as code obfuscation, encryption, polymorphism, and packing. As a result, there has been a growing interest in using advanced Deep Learning (DL) techniques to analyze malware and detect these suspicious behaviors.

S everal studies have explored the transformation of binary code or bytecode into image representations to leverage the capabilities of Deep Neural Network (DNN) for malware detection. However, these approaches frequently fail to capture semantic context, structural anomalies, and obfuscation pattern features critical for accurate and robust classification.

Designing effective detection systems, particularly those utilizing visual representations, requires a thorough understanding of the structural composition of Android applications. The following section presents an overview of the Android Package Kit (APK) file structure, emphasizing its core components relevant to malware analysis.

1.1 Overview of the Android APK file structure

The APK is a compressed file that the Android OS uses to distribute and install applications, consisting of core files and folders such as the application bytecode, assets, resources, and a manifest file, as presented in Fig. 1.

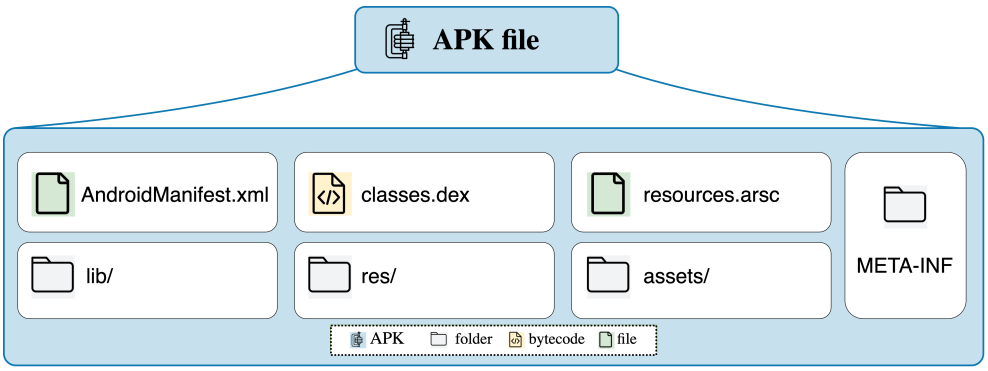


Figure 1: An illustration of the structure of an Android APK file, highlighting key components such as application bytecode, assets, resources, and the manifest file.

Visualization approaches mainly focus on analyzing the *AndroidManifest.xml* and *Classes.dex* files, as these components provide critical insight into the structure and behavior of Android applications. The *AndroidManifest.xml* file specifies essential metadata, including the application's components, package name, and requested permissions. In contrast, the *Classes.dex* file contains the executable bytecode intended for the Android Dalvik Virtual Machine (DVM), making it a vital source for behavioral analysis. This research focuses on encoding and analyzing the *Classes.dex* file, given its importance in capturing malicious behavior. Our proposed approach employs CNN models to detect hidden Android malware threats by transforming bytecode into visual representations, focusing on highlighting anomalous operational or structural patterns indicative of obfuscation or malicious intent.

1.2 signature-based Analysis

The signature-based detection method is widely used to recognize and detect malware. A software signature is a unique identifier that cannot be replicated and is typically generated using hash algorithms such as RSA, MD5, SHA1, SHA-256, and SHA-512 [16, 20]. The detection engine generates a signature for the software and compares it with a blocklist database of signatures stored locally or in the cloud. Typically, these blacklisted databases are proprietary assets of vendors, with restricted access granted to licensed users. A significant limitation of this method is the need for

the detection engine to continuously refresh its blocklist database, which can lead to potential gaps in identifying new zero-day malware [21]. As technology evolves, malware creators continue to find new techniques to evade detection, such as code alteration, function modification, file repacking, data encoding, or null byte injection, all to generate new signatures capable of evading security defenses [17, 22–25].

1.3 dynamic analysis

Dynamic analysis is a pivotal method for malware detection, which involves observing and understanding software behavior during execution within a controlled and contained environment, such as a Sandbox or Virtual Machine (VM). This technique is effective in detecting abnormal actions, such as invoking suspicious system calls [26], examining network traffic [27], altering memory [28], and detecting errors in Logcat that invoke suspicious services from the OS [29].

However, this method requires accessing or monitoring users' sensitive information, which can be impractical when managing highly confidential data [30]. Despite its promising outcomes, acquiring an extensive dataset of labeled training data for optimal performance is often both time-consuming and costly [31]. Security researchers are increasingly shifting to the visualization of malware based on static analysis images, which allows instant scanning of malware images to overcome the challenges posed by new malware [32–34]. Unlike dynamic analysis methods that require days or weeks to monitor suspicious behavior in an application, these image representations can be harmless, do not require manual feature engineering, and resist typical obfuscation techniques employed by adversaries [35].

1.4 static analysis

Static analysis is a technique used to evaluate applications without executing them or observing their execution behavior, which can often be demanding and time-consuming. By not requiring execution, static analysis provides a unique assessment mode comparable to behavioral analysis techniques. One of the primary advantages of this malware detection method is its cost-effectiveness, as it minimizes the need for additional hardware or extensive computational resources beyond the actual analysis tool itself [19].

Despite its advantages, this approach has notable limitations. Specifically, it largely depends on identifying already known malware patterns, which challenges its effectiveness in generalizing and detecting evolving zero-day malware. Research efforts focus on improving the detection of suspicious activities using advanced methodologies such as machine learning (ML) and convolutional neural network (CNN) [36, 37] to mitigate this limitation. These innovations aim to improve the adaptability and robustness of static analysis against evolving threats.

1.5 Contributions

Our novel Android malware visualization framework uniquely integrates critical semantic and structural features extracted from executable bytecode and transforms them into RGB representations. Unlike previous techniques, MalVis enhances interpretability and classification accuracy while maintaining resilience against obfuscation.

Our contributions include the following:

- **MalVis Dataset:** Introducing MalVis, the largest Android malware visualization dataset with over 1.3 million images across ten classes, including nine malware types and benign software that is accessible to the research community[1]. Scripts for generating these various visualization methods are publicly available on GitHub at the link[2].
- **Enhanced Visualization Framework:** Developing an advanced MalVis framework that enhances malware visualization by incorporating an entropy encoder with an N-gram technique. This approach utilizes the three RGB channels to effectively capture a broader range of malware characteristics, including encryption, compression, packing, and structural irregularities. This improves the precision of malware pattern detection in the visualizations.
- **Enhanced Multiclass Labeling:** Implementing an improved multiclass labeling approach using results from Euphony [38] and VirusTotal [39] allows precise classification and analysis of malware behavior, enhancing targeted threat identification and classification.
- **Robust Detection Model:** Evaluation of the performance of the MalVis framework on several SOTA visualization methods using advanced deep CNN architectures such as MobileNet-V2, DenseNet201, ResNet50,

[1] https://www.mal-vis.org
[2] https://github.com/makkawysaleh/MalVis

and Inception-V3, combined with several ensemble techniques, to further improve detection accuracy and generalization. The results showed that the MalVis framework achieved superior performance compared to others.

This research builds on our previous work, "Improving Android Malware Detection Using a Bytecode-to-Image Encoding Framework" [37] to detect anomalous structural and malicious features in Android malware. Although traditional detection techniques such as signature-based, static, and dynamic analysis remain prevalent, visualization-based approaches have gained popularity due to their speed and ability to highlight malicious patterns. However, existing methods often rely on simplistic byte-to-color mappings based on byte location in the file, which overlook semantic features and abnormal structure traits of malware. Additionally, they struggle against obfuscation, encryption, and packing. MalVis offers a richer and more interpretable representation that enhances the classification's robustness and addresses existing methods' limitations.

This paper is organized as follows. Section 1 introduces the background of Android malware threats and provides an overview of the Android APK file structure, followed by traditional detection approaches and a summary of our key contributions. Section 2 reviews related work, including the motivation behind visual-based detection, limitations of existing malware image datasets, and prior grayscale and RGB encoding techniques. Section 3 details our proposed MalVis framework, including the data generation process, bytecode-to-image transformation, and an in-depth analysis of entropy and N-gram features through two distinct visualization approaches. Section 4 defines the performance evaluation metrics used to assess the model's effectiveness. In Section 5, we present experimental results across binary and multiclass classification tasks, evaluate the impact of data balancing using undersampling, and demonstrate performance improvements through ensemble modeling. Finally, Section 6 concludes the paper and outlines future research directions.

2 Related Works

This section describes the motivation to adopt visual representations in malware analysis. Next, we review Android malware visualization datasets that benchmark image-based malware detection. Finally, we discuss recent visualization-based detection techniques, focusing on grayscale and RGB encoding methods to detect malicious patterns.

2.1 Motivation

Deep Neural Networks, especially CNNs, have shown exceptional performance across domains such as vision, biomedical, and cybersecurity [40–43], primarily due to the availability of large, structured datasets. In malware detection, transforming code into image representations allows CNNs to identify visual patterns of malicious behavior, offering a scalable, non-executable, and efficient alternative to traditional analysis methods. These representations are significantly smaller than the raw executable files (Fig. 2), reducing storage needs and execution risks.

Our approach further benefits from transfer learning by leveraging pretrained CNNs trained on massive image datasets such as ImageNet, enabling effective pattern recognition in malware with minimal domain-specific training. Despite these advantages, progress is hindered by limited access to large-scale, interpretable, and public malware visualization datasets. Addressing this gap is essential for advancing robust, explainable, and reproducible malware detection research.

2.2 Existing Malware Image Datasets

We highlight two widely known Android malware datasets: AndroZoo [44] and Drebin [45], both commonly used in Android malware detection research. However, because our MalVis approach focuses on transforming bytecode into images for visualization, we primarily compare MalVis with other existing image-based datasets in this section.

Scott Freitas *et al.* [46] introduced the MalNet database, a substantial contribution to the field with over 1.2 million malware images spanning 47 types and 696 families. While their direct byte-to-location color mapping method is innovative within the Android application structure, our dataset, MalVis, offers enhancements in the form of over 1.3 million images, with a particular focus on addressing malware obfuscation techniques. This focus improves the effectiveness of Android malware detection, as discussed in more detail by Makkawy *et al.* [37].

Virus-MNIST, proposed by David A. *et al.* [4], is a large publicly available malware image dataset. The dataset includes 51,880 grayscale images of malware, classified into nine virus classes and one benign class, all formatted as $[32 \times 32]$ images. The dataset represents the malware classification problem, like the famous MNIST dataset used for handwriting recognition. Malware images are generated by converting the first 1,024 bytes of Portable Executable (PE) files into $[32 \times 32]$ grayscale images. Although Virus-MNIST introduces a significant step towards standardizing malware image

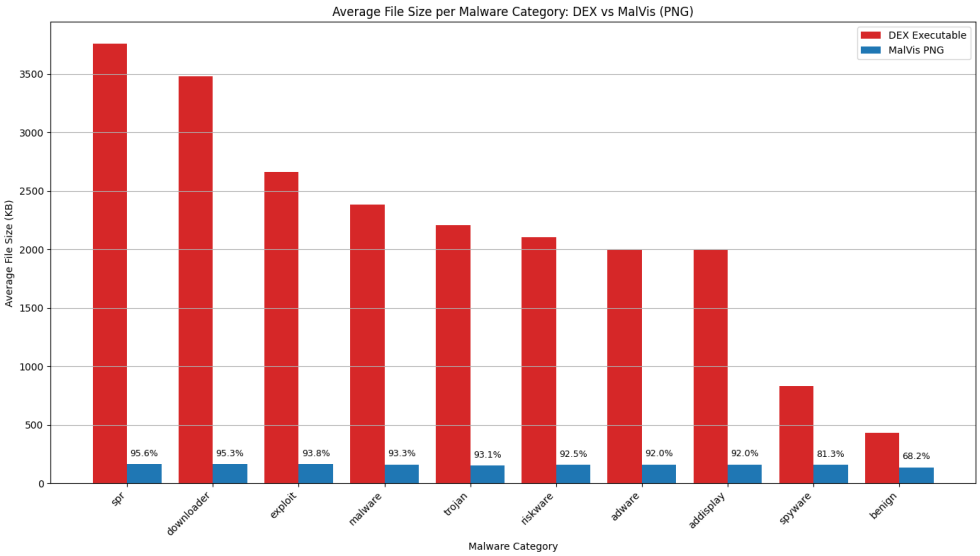


Figure 2: Comparison of average file sizes in DEX executables vs malVis PNG representations across malware categories (with size reduction percentages)

datasets, its representation of malware using only the first 1,024 bytes may result in a lack of capturing the complete characteristics of the malware [47].

L. Nataraj *et al.* [48] presents the MalImg dataset, which offers a straightforward and efficient malware visualization and classification method. Using image processing techniques, this approach classifies malware samples based on their similarity to specific malware types, utilizing standard image features. MalImg achieves a notable classification accuracy of 98% on a dataset comprising 9,458 samples across 25 distinct malware types. However, with this limited dataset size, there is a possibility that the model is overfitting to the specific characteristics of these samples.

Table 1 summarizes public and private image-based malware datasets, including MalVis, MALNET-IMAGE, and Virus-MNIST, providing details on the number of classes and dataset size. Despite existing visualization contributions, these methods face some limitations. As observed by Kunwar *et al.* [49], MalNet encodes malware bytecode based on byte location in the executable file, lacks resilience to obfuscation, and does not identify suspicious behaviors. Virus-MNIST [4] uses only the first 1,024 bytes of PE files, limiting its representational scope. In contrast, MalVis combines entropy and N-gram analysis to generate color-encoded RGB representations that emphasize abnormal structures, encryption, packing, and compression behaviors. Our experiments show that this richer visual encoding enhances model interpretability and improves classification performance. A detailed discussion of the MalVis dataset and its visualization approach is presented in Section 3.

Table 1: Summary of image-based malware datasets detailing the number of classes, dataset sizes, and availability.

Dataset	# Classes	Dataset Size	Public	Private
MalVis	10	1,300,822	✓	
MalNet [46]	696	1,262,024	✓	
AndroDex [50]	180	24,746	✓	
Virus-MNIST [4]	10	51,880	✓	
Malimg [51]	25	9,458	✓	
Microsoft [52]	9	108,000		✓
IVMD-2013 [36]	2	37,000		✓
AdvAndMal [53]	12	5,560		✓
Halil-2020 [54]	2	29,100		✓

2.3 Visualization Strategies for Malware Detection

As image-based malware detection has become a powerful paradigm for analyzing Android applications, it bypasses manual feature engineering through automated visual pattern recognition. Current approaches primarily focus on two types of representations:

2.3.1 Grayscale Image Encoding

The foundational work by Nataraj *et al.* [48] established grayscale conversion by mapping binary bytes to pixel values in the range (0–255), revealing structural patterns in malware families. Modern implementation includes DexRay by Nadia Daoudi *et al.* [55]; it converts DEX bytecode into 1D grayscale vectors (1×128×128) for CNN classification, achieving 96% F1-score while resisting obfuscation. Despite its highest accuracy, their approach resulted in smaller-size grayscale images that could be affected by more data loss in the representations. In another instance, Wang *et al.* [56] developed a novel scheme that combines static and dynamic analysis with CNN for efficient malware detection and classification. Their method integrates a Convolutional Block Attention Module (CBAM) with CNN to detect malware similarities using grayscale images from the MalImg and Microsoft datasets. However, their experiment was conducted on a relatively small dataset of approximately 20,000 samples covering 25 types of malware, which could be affected by model overfitting.

2.3.2 RGB Image Encoding

Advanced malware variants often exhibit more sophisticated patterns and behaviors that are difficult to capture using standard grayscale or single-channel representations. Additional color channels are required to encode these complex characteristics, enabling richer feature representation and enhancing the model's ability to detect subtle malicious traits. Asim *et al.* [57] introduced a technique that transforms APK files into lightweight RGB images utilizing a predefined dictionary and an intelligent mapping mechanism. Their method converts the *AndroidManifest.xml* permissions into ASCII values, which are then aggregated and encoded into a single color value.

While this approach facilitates the image-based representation of APK features in RGB channels, it suffers from significant information loss due to the summation of ASCII values, which flattens each permission into a single numerical value. This reduction hinders the model's ability to capture detailed permission information, thus limiting its effectiveness in capturing nuanced malicious behaviors.

Progress in this field is hindered by the scarcity of publicly available visualized malware datasets [58] and the need for robust methodologies to capture malware patterns and behaviors [46] effectively. The MalVis dataset aims to tackle these challenges by providing comprehensive representations of malware, which convert abnormal operational and structural patterns in bytecode into visual forms. In addition, it includes multiclass labels for accurate malware classification and analysis, thereby improving targeted threat identification.

3 Methodology

This section describes the methodology for collecting and constructing the MalVis dataset, including data generation and construction, MalVis bytecode-to-image visualization, CNN architectures with experiment settings, and environment setup.

3.1 Data Generation and Construction

The MalVis generation process uses a subset of the AndroZoo dataset [44], a key resource in Android research encompassing 24,743,375 applications collected from platforms like the Google Play Store. The binary classification dataset consists of 49,150 malware and 135,324 benign samples. The multiclass malware dataset utilizes Euphony [38] to categorize malware into 289 distinct classes. For training purposes, we focus on the nine largest classes to enhance labeling accuracy and reduce false positives, avoiding samples with multiple labels. We also verify these samples using Virustotal [39] to ensure reliability. The refined dataset primarily included malware samples, along with an addition of 135,324 benign samples sourced from AndroZoo. Figure 3 presents the distribution of nine malware and the benign class within 1,300,822 application visualizations.

3.2 MalVis bytecode-to-image visualization

The MalVis bytecode-to-image visualization process begins by extracting Dalvik Executable (DEX) files from Android APKs using AndroGuard [59], a well-known reverse engineering tool. This step yields the `classes.dex` files, as

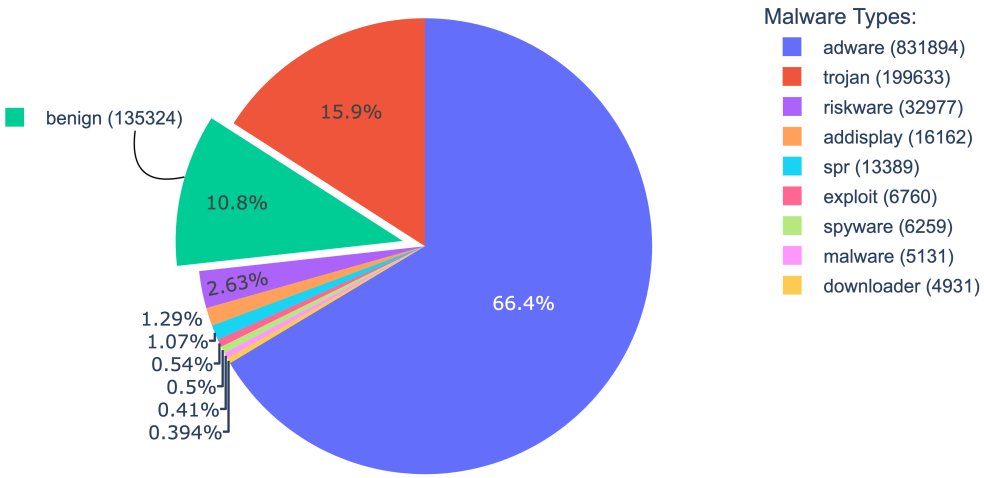


Figure 3: Distribution of malware types and benign in MalVis.

illustrated in Fig. 9④. These `.dex` files consist of byte values in the range of `0x00` to `0xFF`. These values are first converted into a one-dimensional array of unsigned integers, where each value is between 0 and 255. These integers correspond directly to the pixel color intensities. The 1D array is reshaped into a two-dimensional grayscale image with a fixed width and height of 256 pixels to visualize the bytecode. This transformation employs Nearest-Neighbor Interpolation (NNI) and the Pillow library in Python, ensuring consistent image dimensions while preserving the original byte sequence structure.

Shannon entropy is applied to the executable dex file using a 32-byte sliding window to determine the red and blue channels. These channels are defined by distinct formulas motivated by [60], as described in our earlier paper [37]. Each formula utilizes Shannon entropy 1 differently to highlight regions of considerable randomness, which may indicate encryption or obfuscation. Note

$$H(X) = -\sum_{i=1}^{N} P(x_i)\log_2(P(x_i)), \tag{1}$$

where

- $H(X)$: Represents the Shannon entropy of the random variable X, which measures the uncertainty or randomness in the 32-byte sequence.
- $P(x_i)$: The probability of observing the specific i^{th} outcome or byte value x_i. It is calculated as the frequency of x_i in the 32-byte sequence divided by the total number of bytes (32).
- N: Denotes the total number of unique outcomes for the random variable X. For a single byte, $N = 256$ (corresponding to values $\{0, 1, 2, ..., 255\}$).
- x_i: Refers to a specific byte value in the range $\{0, 1, 2, ..., 255\}$. In the context of a sliding window of 32 bytes, it represents the i^{th} byte in the sequence.
- $\log_2$: The logarithm to base two is commonly used in entropy calculations to express the result in units of bits.

Given the varied types of malware introduced by the MalVis dataset, we have explored techniques to improve the recognition of these variations. Our analysis focuses on extending our earlier approach [37] from two color channels (red and blue) to three channels by encoding additional feature into the green channel of RGB images using two primary encoding methods:

- **Classbyte Encoding:** We adopt the Classbyte encoder introduced by Duc-Ly *et al.* [34], which maps semantic features of bytecode to varying intensities of the green channel. We selected this method due to its effectiveness and comparable performance to our previously employed entropy-based encoding for binary classification tasks.
- **N-gram Structural Encoding:** We incorporate N-gram representations derived from byte sequences to capture the malware bytecode's underlying structural patterns and contextual dependencies. This technique,

commonly used in malware detection research [61, 62], enriches the green channel with statistical features that reflect code regularities and anomalies, thereby enhancing the capability to distinguish between different malware types.

The following subsections discuss the implications of these methods for advancing malware visualization.

3.2.1 Approach MalVis-A

This approach uses the Classbyte representation, which performs similarly to the entropy encoder in binary classification. It translates the features identified by the four Classbyte colors into four distinct shades of green in the green channel, as illustrated in Fig. 4 ①. The method highlights sections of bytecode containing both clear-text printable and non-printable ASCII characters and null byte areas, as illustrated in Fig. 4 ②. These distinctions assist in analyzing the bytecode to determine whether it has been encrypted or injected with null bytes to evade malware detection.

The previously generated red and blue channels are combined with the newly constructed green channel, resulting in MalVis-A RGB images, as shown in Fig. 4 ③. Unfortunately, this approach did not yield the desired improvement in the accuracy of multiclass classification. Further results of the analysis and evaluation of this approach are presented in Section 5.

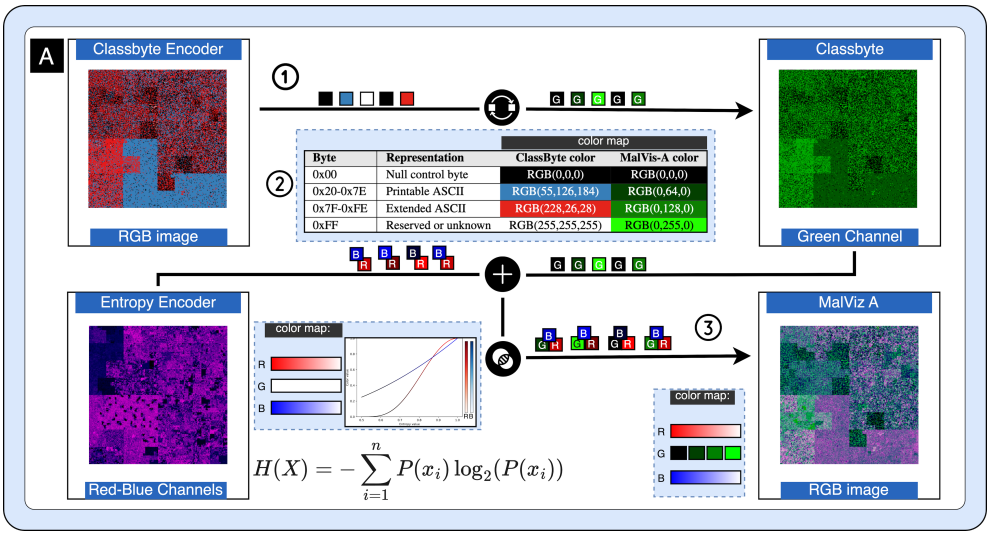


Figure 4: Overview of constructing the MalVis-A visualization method, resulting in RGB image representations using the Classbyte encoding in the green channel and encoding entropy in the red and blue channels.

3.2.2 Approach MalVis-B

This approach utilizes the N-gram method, which has been extensively studied for malware anomaly detection. The approach is particularly relevant for Android applications, which are often written in Java and Kotlin, thus inheriting the programmatic structure. Abnormalities are detected when the byte sequences differ from the typical bytecode structure using the green channel depicted in Fig. 5 ②. One of the key goals of MalVis is to bridge the gap between raw visualization and interpretability. Unlike prior methods that map byte values to color values without semantic linkage, our framework encodes interpretable attributes: entropy highlights encrypted or compressed code regions. At the same time, N-gram transitions emphasize structural irregularities in bytecode. This mapping allows security analysts and researchers to visually associate distinct color patterns with specific malware behaviors, such as repacking functions or obfuscation.

Figures (6-8) further clarify this concept. For example, the script in Fig. 6 below depicts a simple Java code for a *'for'* loop before it is compiled into bytecode. The bytecode often reveals specific patterns that represent the underlying syntax and structure of the program. The keyword *'for'* indicates the presence of a loop followed by an initialization statement, a condition, and an increment surrounded by braces $(\cdots)$, while curly braces $\{\cdots\}$ denote the loop's body.

Running "javac For_Loop.java" compiles the code into a bytecode file named "For_Loop.class", which the DVM uses to execute the program, as demonstrated in Fig. 7.

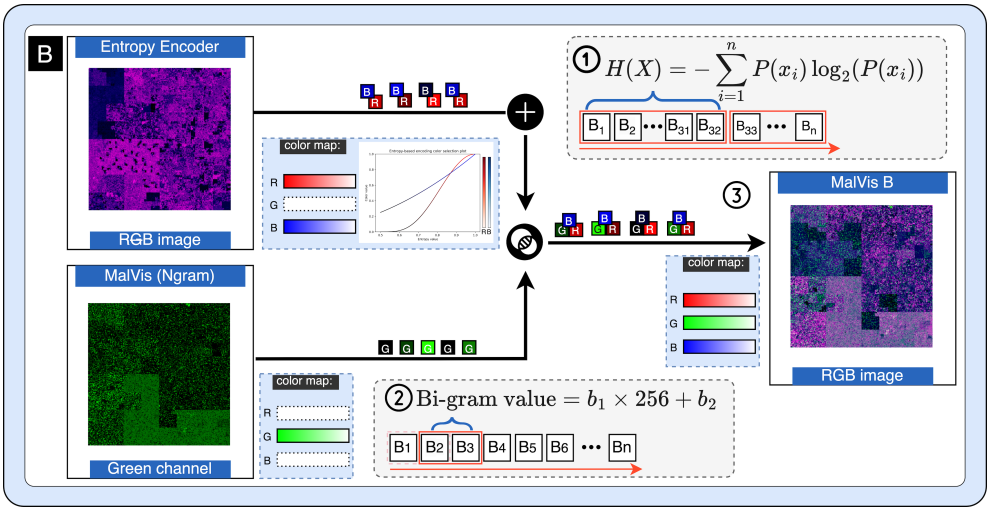


Figure 5: Overview of constructing the MalVis-B visualization method using the N-gram encoding in the green RGB channel.

```
// For_Loop.java
public class Main {
  public static void main(String[] args) {
    for (int i = 0; i < 5; i++) {
        System.out.println(i);
    }
}
```

Figure 6: An example of a simple for-loop written in Java.

As illustrated by the MalVis-B algorithm in Fig. 10, we implemented the Bi-gram method on the raw bytes using a two-byte window to recognize anomalies in the bytecode's operational structure. Using Bi-gram on the bytecode represents the transition between instructions as listed in Fig. 8.

The Bi-gram method can detect obfuscated code by identifying irregular Bi-gram patterns. A two-byte window size allows for detecting simple structural patterns, while larger windows capture more complex ones. However, this decision involves a trade-off that requires more computational resources and time to generate over 1.3 million images. Given limited resources, this study used a two-byte window size. Further research could optimize the window size for improved performance.

The Bi-gram formula is

$$\text{Bi-gram value} = b_1 \times 2^8 + b_2, \tag{2}$$

which takes two consecutive bytes, b_1 and b_2, to compute the Bi-gram value. The multiplication of the first byte b_1 by $2^8 = 256$ shifts it to higher-order in the combined value, which is then added to the b_2 value, as shown in line 16 of the Algorithm, Figure 10. The resulting Bi-gram value represents the degree level of a green pixel and is normalized to the range $[0, 1]$ by dividing by the maximum possible value of $(256 \times 256) - 1 = 65,535$ as described by

$$g = \frac{\text{Bi-gram value}}{(256 \times 256) - 1} = \frac{b_1 \times 256 + b_2}{65,535}. \tag{3}$$

Finally, if the byte is the last in the file, it is reset to 0, as detailed in lines 18 and 19 of the Algorithm, Fig. 10. Hence, MalVis presents a conceptually innovative visualization design that maps meaningful malware properties to distinct visual domains. Consequently, MalVis is more effective and better aligned with the objectives of explainable malware classification. This approach has demonstrated improved accuracy in the context of multiclass MalVis, and both representation techniques are evaluated in Section 5.

```
// For_Loop.class
iconst_0          // Push 0 into the stack.
istore_1          // Store 0 to local variable 1.
LoopStart:
iload_1           // Load variable 1 (i) into the stack.
iconst_5          // Push 5 into the stack.
if_icmpge EndLoop // Compare i and 5, jump to EndLoop if i >= 5.
getstatic         // Access System.out.
iload_1           // Load variable 1 (i) into the stack.
invokevirtual     // Call System.out.println(i).
iinc 1, 1         // Increment variable 1 (i++).
goto LoopStart    // Jump to the start of the loop.
EndLoop:
```

Figure 7: Translation of a Java for-loop into its equivalent JVM instructions in bytecode form after compilation.

```
iconst_0 istore_1 // Initialization of i to 0.
istore_1 iload_1 // Load the stored value of i.
iconst_5 if_icmpge //Conditional jump.
getstatic iloa_1 // Prepare to print the value of i.
invokevirtual iinc // After printing, increment i.
iinc goto // Jump back to the start of the loop.
```

Figure 8: Representation of the employed Bi-gram approach on the Java instructions capturing the semantic transition of these instructions.

Algorithm 1 MalVis-B Visualization Algorithm

Input: Data array $data$ of bytecode, symbol map $symbol\_map$, index x
Output: RGB values in the range [0, 255]
$e \leftarrow \text{Entropy}(data, 32, x, \text{len}(symbol\_map))$ ▷ Calculate entropy using a window size of 32 bytes
function CURVE(v)
    $f \leftarrow (4v - 4v^2)^4$
    $f \leftarrow \max(f, 0)$
    return f
end function
if $e > 0.5$ **then**
    $r \leftarrow \text{curve}(e - 0.5)$ ▷ Red component is determined by the scaled entropy value
else
    $r \leftarrow 0$ ▷ If entropy is less than or equal to 0.5, set red component to 0
end if
$b \leftarrow e^2$ ▷ Blue component is proportional to the square of entropy
if $x < \text{len}(data) - 1$ **then**
    $n\_gram\_value \leftarrow (data[x] \ll 8) + data[x + 1]$ ▷ Compute 2-byte n-gram value
    $g \leftarrow \frac{n\_gram\ value}{65535}$ ▷ Normalize n-gram value to [0, 1] for green component
else
    $g \leftarrow 0$ ▷ If at the last byte, the green component is set to 0
end if
return $[\text{int}(255 \cdot r), \text{int}(255 \cdot g), \text{int}(255 \cdot b)]$ ▷ Return RGB values scaled to the range [0, 255]

Figure 10: Algorithm illustrating the generation of RGB image representation in the MalVis-B approach, utilizing entropy for the red and blue channels and N-gram for the green channel.

3.3 The Impact of Entropy and N-gram on MalVis representations experiments

In this section, we further investigate the sensitivity and interpretability of MalVis visualizations. We conducted a controlled analysis by applying targeted transformations to a benign Android application, specifically WhatsApp *Classes.dex*. The goal was to investigate and quantify the impact of encryption and unstructured operations in bytecode changes on the RGB image representations generated by our framework.

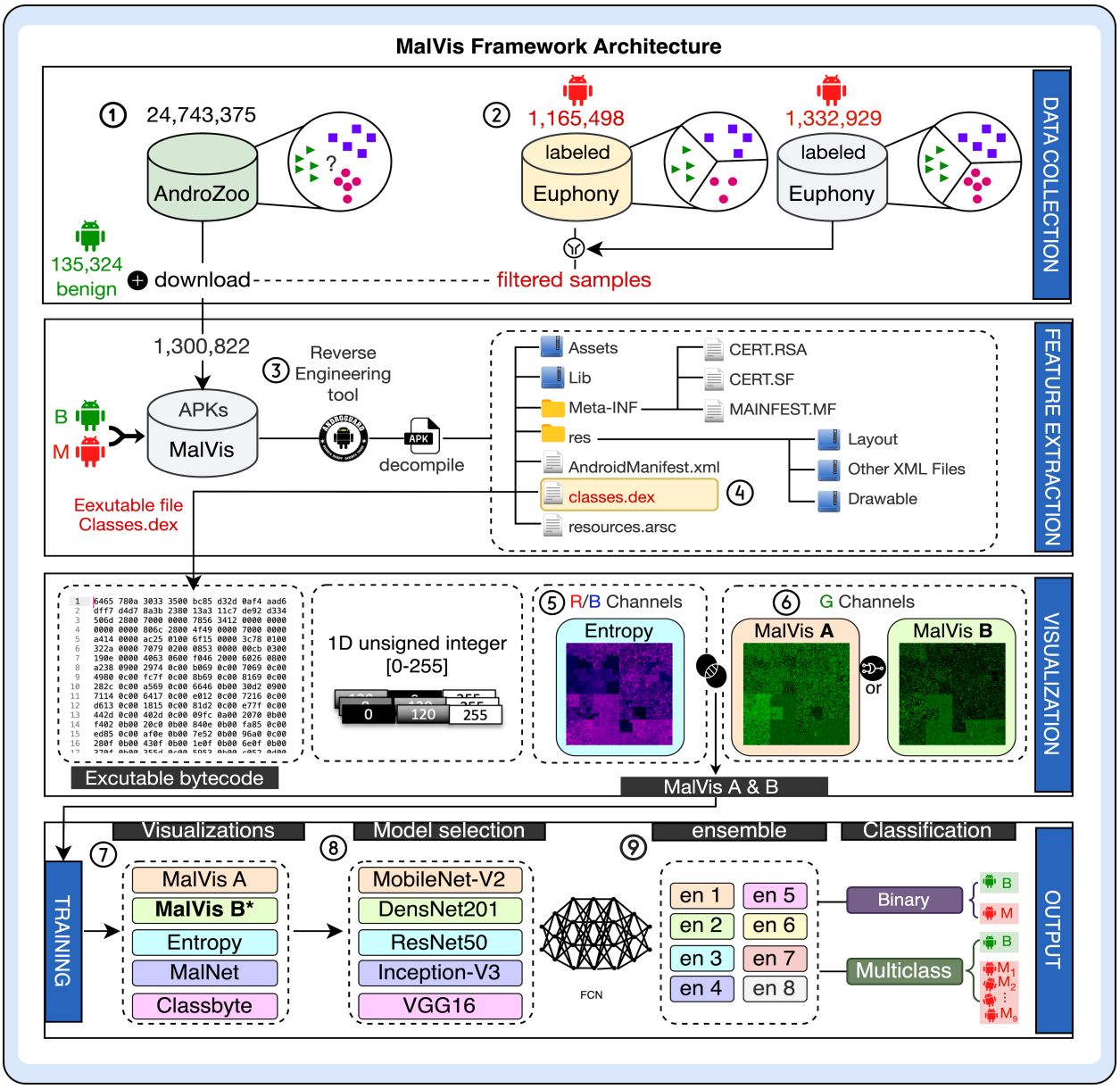


Figure 9: A schematic illustration of the proposed framework architecture is organized into four distinct rows. The first row details the data collection and labeling process. The second row focuses on feature extraction. The third row constructs and generates RGB images using entropy and N-gram methods. The bottom row describes the training process, including visualization techniques, CNN models, ensemble methods, and both binary and multiclass classifications.

3.3.1 Obfuscation Detection captured by Entropy in Red and Blue Channels

In this experiment, we applied `AES-256` encryption in Electronic Codebook (ECB) mode to the initial 30% of the `Classes.dex` bytecode. This encryption caused a noticeable entropy shift, particularly affecting image representations' red and blue channels. Entropy, which quantifies randomness over 32-byte windows, increased significantly in high-entropy areas, leading to brighter pixel intensities. This effect simulates obfuscation techniques that malware creators use to evade detection. As a result, the red and blue channels in Fig. 11 display brighter pixels in the top-left region, highlighting the encrypted sections in the representation.

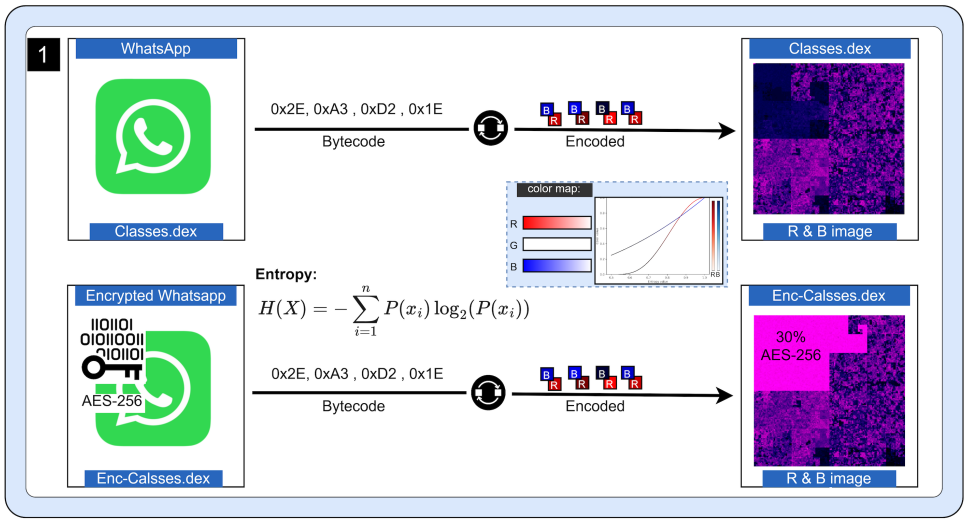


Figure 11: The impact of 30% AES-256 encryption on *Classes.dex* file captured by the entropy encoder in the red and blue channels of MalVis representations.

3.3.2 Unstructured bytecode Insertion captured by N-gram in Green Channel

In this experiment, we examined the structural sensitivity of the green channel by injecting random, unstructured operations into the initial 30% of the `Classes.dex` file. This action disrupted the byte sequence, causing noticeable distortions in the N-gram values, significantly impacting the green channel. MalVis-B, which utilizes bi-gram formulas to detect abnormal operational patterns, recorded these disturbances as increased bi-gram values, resulting in brighter pixel values within green-channel textures, as depicted in Fig. 12. These deviations were apparent when visualized next to an unchanged sample, highlighting the effectiveness of the green channel in detecting structural anomalies.

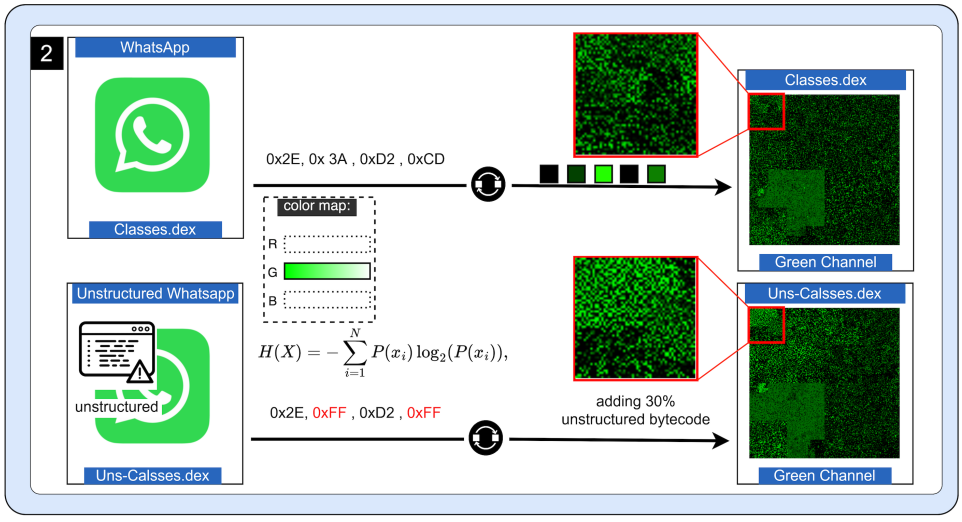


Figure 12: The impact of injecting 30% randomized unstructured operations to *Classes.dex* file captured by the N-gram encoder in the green channels of MalVis-B representations.

3.4 Model Architecture and Experiment Settings

To assess the effectiveness of our proposed approach, we used a selection of well-recognized CNN models, including MobileNetV2, ResNet50, DenseNet201, VGG16, and InceptionV3. These models were applied to our generated

visualizations and baseline comparison methods. The employed CNN models have proven highly effective in malware detection because they capture intricate patterns and features within image data [46, 63]. To ensure consistency across the CNN models, all images were resized to 224×224 pixels using nearest neighbor interpolation to align with the input dimensions required by the models. The dataset was partitioned into 80% for training, 10% for validation, and 10% for testing. The batch size 64 was chosen based on empirical experimentation, as it provided an optimal trade-off between training speed and memory consumption on our GPU setup. The training was conducted over 50 epochs and carefully monitored to mitigate overfitting. This setup allowed for consistent and accurate assessments of the models' performance across different visualization techniques.

3.5 Environment Setup

MalVis visualization and model training were generated using an Ubuntu Server 22.04 LTS OS with x86 64 architecture. The hardware setup included a 16-core AMD Ryzen Threadripper PRO 5955WX processor, 128 GB of DDR4 RAM at 3200MHz, and an NVIDIA RTX A6000 graphics card. The system was configured within a controlled environment to ensure accurate results and minimize external influences.

4 Performance Measures

To ensure fairness when comparing the visualization methods and evaluating our proposed approaches alongside the baseline methods presented in Table 2, we employed accuracy, precision, recall, ROC-AUC, and MCC as validation metrics in the binary classification context. Similarly, the same metrics were employed for consistent evaluation in multiclass classification, as demonstrated in Table 3. The accuracy (4) indicates the percentage of instances correctly identified among the entire set of samples. The F1-score (5) provides a harmonic mean of the model's precision and recall, accounting for false positives and false negatives. Precision (6) refers to the proportion of true positives in relation to all positive predictions made. Recall (7) denotes the fraction of actual positives correctly identified by the model. ROC-AUC measures the area under the receiver operating characteristic curve, highlighting the balance between sensitivity and specificity. The MCC (8) serves as a metric to assess classification performance, factoring in true and false positives and true and false negatives.

Accordingly,

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}, \tag{4}$$

$$\text{F1-score} = 2 \times \frac{P \times R}{P + R}, \tag{5}$$

$$\text{Precision} = \frac{TP}{TP + FP}, \tag{6}$$

$$\text{Recall} = \frac{TP}{TP + FN}, \tag{7}$$

and

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}, \tag{8}$$

where TP, TN, FP, and FN represent true positive, true negative, false positive, and false negative, respectively.

5 Results

This section provides a comparative analysis of the performance of the newly introduced visualizations, MalVis-A and MalVis-B, compared to baseline methods, namely MalNet [46] and Classbyte, as detailed in the following subsections:

5.1 Evaluation of MalVis-A and MalVis-B Performance Compared to Other Methods on a Binary Classification Dataset

All methods used the same settings and were trained on identical subsets of training data to ensure a fair comparison. As shown in Table 2, the MalVis-A approach, which combines Classbyte and Entropy, did not improve classification performance as expected. Instead, it disrupted the patterns captured by the entropy encoder, as shown in Fig. 13. Encoding the four colors of the Classbyte method into a single green channel erases and replaces previously detected

Table 2: Comparison of visualization approaches using different CNN models. Abbreviations in the table include MNv2 (MobileNet-V2), DN201 (DenseNet201), RN50 (ResNet50), and INC-V3 (Inception-V3). Bold values highlight the highest score for each metric within the respective model.

Approaches	Models	Accuracy	F1-score	Precsion	Recall	MCC	R-AUC
Classsbyte Encoder [34]	**MNv2**	91.60%	85.42%	79.43%	92.39%	80.02%	96.91%
	DN201	94.38%	89.39%	89.93%	88.85%	85.57%	97.57%
	RN50	93.02%	86.51%	89.22%	83.97%	81.88%	96.38%
	INC-V3	94.38%	89.03%	92.73%	85.62%	85.38%	97.75%
	VGG16	93.18%	87.33%	86.46%	88.22%	82.68%	96.17%
MalNet Encoder [46]	**MNv2**	92.78%	85.63%	91.16%	80.74%	81.10%	97.11%
	DN201	89.66%	83.16%	74.46%	90.11%	77.27%	96.91%
	RN50	86.34%	67.60%	91.92%	53.46%	63.22%	94.69%
	INC-V3	94.82%	90.01%	92.69%	87.47%	86.58%	97.84%
	VGG16	93.87%	88.02%	91.80%	84.54%	84.04%	97.38%
Entropy-based [37]	**MNv2**	93.85%	88.10%	90.88%	85.50%	84.03%	97.57%
	DN201	**95.32%**	**91.30%**	90.54%	**92.07%**	**88.10%**	**98.25%**
	RN50	93.14%	86.50%	90.96%	82.47%	82.10%	97.11%
	INC-V3	94.94%	90.43%	91.12%	89.75%	86.99%	97.93%
	VGG16	93.59%	87.00%	**94.64%**	80.49%	83.25%	97.45%
MalVis-A	**MNv2**	91.97%	84.05%	89.22%	79.45%	78.94%	96.76%
	DN201	94.95%	90.28%	92.60%	88.08%	86.92%	98.15%
	RN50	93.64%	87.47%	92.03%	83.34%	83.40%	97.38%
	INC-V3	94.18%	88.63%	92.50%	85.07%	84.87%	97.88%
	VGG16	92.81%	86.38%	87.26%	85.52%	81.50%	96.89%
MalVis-B*	**MNv2**	**95.04%**	**90.57%**	**91.76%**	**89.42%**	**87.22%**	**98.14%**
	DN201	95.22%	90.91%	**92.16%**	89.69%	87.68%	98.19%
	RN50	**95.08%**	**90.60%**	**92.35%**	**88.91%**	**87.30%**	**98.18%**
	INC-V3	**95.19%**	**90.81%**	**92.58%**	**89.10%**	**87.58%**	**98.06%**
	VGG16	**94.60%**	**89.89%**	89.68%	**90.11%**	**86.21%**	**97.90%**

*The results denote the optimal proposed method, referred to as MalVis-B.

patterns. In contrast, the proposed MalVis-B method, which utilizes entropy and N-gram visualization, outperformed other methods in most CNN models except for DenseNet201.

Although DenseNet201 did not show significant improvements across all metrics, it demonstrated superior precision, highlighting the model's effectiveness in accurately distinguishing true positives from false positives. The observed limitations in the remaining metrics are attributed to the highly imbalanced dataset outlined in Section 3.1, where the prevalence of benign samples significantly exceeds that of malware. This imbalanced dataset was further addressed in the multiclass dataset evaluation detailed in Section 5.3.

These experiments demonstrate that existing methods, including *Classbyte* and *MalNet*, provide limited semantic and structural variation, resulting in suboptimal performance for malware classification tasks. In contrast, *MalVis-B* outperforms these approaches by integrating both entropy and N-gram patterns, producing meaningful visual representations that more effectively expose obfuscation, encryption, and other malicious behaviors. Notably, our earlier

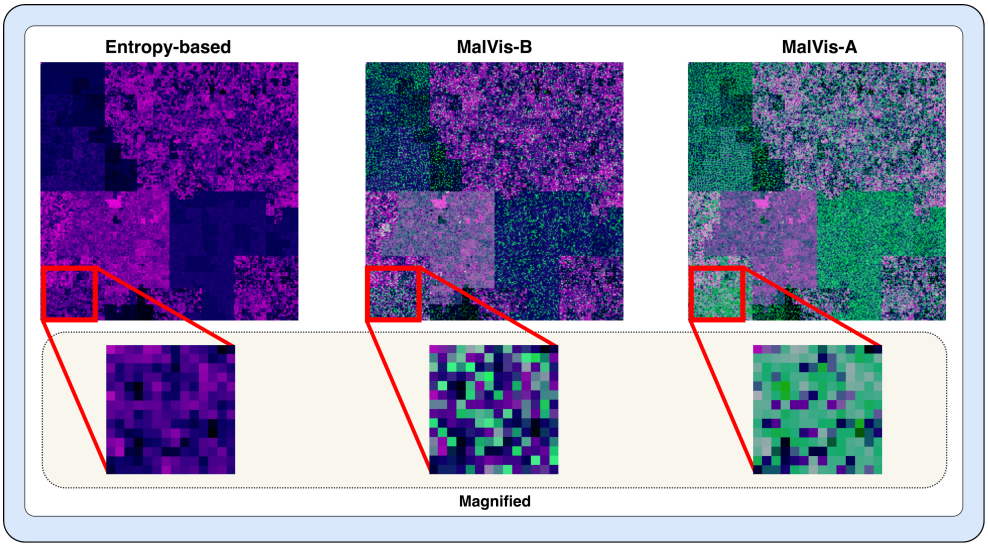


Figure 13: Figure showing the disruption caused by MalVis-A encoding of classbyte in the green channel, which impacts patterns captured by the entropy encoder compared to the representations by MalVis-B.

method relying solely on entropy [37] did not achieve comparable performance, underscoring the value of combining multiple feature types. This highlights the need for enhanced visualization techniques that improve both interpretability and classification accuracy. Accordingly, *MalVis-B* was selected for the subsequent advanced multiclass classification experiments to better distinguish between diverse malware types.

5.2 Evaluation of MalVis-B Performance on Imbalanced Multiclass Dataset

The evaluation of the MalVis-B representation in the imbalanced multiclass malware classification task, presented in Table 3, demonstrated that the ResNet50 model achieved the highest performance. It achieved an overall accuracy of 94.03%, F1-score of 83.54%, and Precision of 83.34%, surpassing the performance of state-of-the-art multiclass malware classification approaches [46]. The analysis of the confusion matrix, presented in Fig. 14, $\boxed{A}$ to $\boxed{E}$ reveals significant challenges in differentiating between the majority and minority classes within the imbalanced multiclass dataset. The darker column for the adware class suggests a bias due to its higher frequency in the training set, as shown in Fig. 3.

A deeper inspection of the confusion matrix Fig. 14 reveals frequent misclassification between Adware, Trojan, and Spyware classes. These malware types often share similar bytecode structures and use comparable obfuscation techniques, leading to visually overlapping patterns in the entropy and N-gram channels. For example, packed adware and spyware samples may exhibit high-entropy values with irregular n-gram sequences, which confound the classifier. These findings highlight the need for refined feature selection and possibly more semantic augmentation in future visualization efforts. This highlights the effect of class imbalance, leading to biased decision boundaries that favor the majority class at the expense of consistent performance across all classes.

Various strategies can address this imbalance, such as oversampling minority classes, undersampling majority classes, applying class weighting in the loss function, and using ensemble methods [64, 65]. The following sections cover applying undersampling to majority classes and discuss the evaluation of eight different ensemble methods in detail.

5.3 Evaluation of MalVis-B Performance using Undersampling on a Balanced Multiclass Dataset

We applied undersampling to the majority classes to address the imbalanced class distribution, creating a more balanced dataset. Although oversampling minority classes is the most effective data balancing method [64], we opted for undersampling due to limited computational resources and the time constraints associated with training the oversampled method. Table 4 presents the evaluation results for undersampling with MalVis-B. The confusion matrix in Fig. 15, models $\boxed{B}$ to $\boxed{F}$, highlights improved differentiation between majority and minority classes. Despite a 15-20% reduction in overall performance relative to the results of the imbalanced dataset (Table 3), we discuss ensemble methods to boost model performance in the following section.

Table 3: Performance results of different models on the MalVis-B imbalanced multiclass dataset.

Models	MalVis-B Multi-Class					
	A	F1	P	R	MCC	ROC-AUC
MNv2	83.27%	83.05%	82.89%	83.27%	67.49%	93.67%
DN201	82.81%	81.81%	81.74%	82.81%	65.27%	**95.29%**
RN50	**84.03%**	**83.54%**	**83.34%**	**84.03%**	**68.53%**	94.03%
INC-V3	80.18%	78.67%	78.74%	80.18%	59.95%	93.39%
VGG16	82.56%	81.99%	81.72%	82.56%	65.42%	91.87%

Table 4: Performance results after undersampling approach on the large-MalVis dataset

Models	MalVis-B Multi-Class					
	A	F1	P	R	MCC	ROC-AUC
MNv2	61.71%	61.86%	62.16%	61.71%	57.47%	89.36%
DN201	**66.29%**	**66.06%**	**65.98%**	**66.29%**	**62.57%**	**91.24%**
RN50	65.00%	64.81%	64.77%	65.00%	61.12%	90.58%
INC-V3	64.43%	64.37%	64.43%	64.43%	60.48%	90.35%
VGG16	60.70%	60.23%	60.02%	60.70%	56.36%	89.35%

5.4 Evaluation of MalVis-B Performance using Ensemble Models on a Balanced Multiclass Dataset

To address the performance impact caused by the undersampling approach, we explored the application of various ensemble methods. The aim was to take advantage of the combined strengths of all CNN models, which enhanced both the models' performance and robustness. The ensemble methods implemented and evaluated include:

- **Average Voting:** Combines predictions by averaging the probabilities of all CNN models.
- **Majority Voting:** Determines the final class by selecting the most predicted by individual models.
- **Weighted Voting:** Assigns different weights to CNN models based on their prediction accuracy. We preserve the ranking performance of the models and assign weights corresponding to their place in the ranking.
- **Min Confidence Voting:** Only consider a model's prediction when it meets the minimum required confidence level. In our implementation, a confidence threshold of 60% was selected.
- **Soft Voting:** Uses the predicted class probabilities to decide the final output.
- **Median Voting:** Determines decisions by selecting the median of predicted class probabilities.
- **Rank-Based Voting:** Ranks predictions from models and aggregates ranks to select a class.
- **Stacking Ensemble:** Trains a new model to integrate the predictions of the base model and improve performance.

In Table 5, the Min Confidence Voting ensemble achieved the highest performance across all evaluation metrics except for ROC-AUC. These results signify superior performance compared to the results in the unbalanced dataset shown in Table 3. The confusion matrix in Fig. 15 in box $\boxed{A}$ illustrates that the Min Confidence Voting ensemble demonstrated enhanced performance by producing a more pronounced diagonal shape. This indicates an improved ability to accurately detect the more challenging classes compared to the CNN models shown in boxes $\boxed{B}$ to $\boxed{F}$ after undersampling. Moreover, the Stacking ensemble achieved the highest ROC-AUC metric, attributable to its ability to integrate predictions from multiple models, thereby leveraging their strengths to improve overall performance in distinguishing different classes.

These findings underline the effectiveness of ensemble methods, particularly Min Confidence Voting and Stacking, in handling multiclass classification challenges on the Large-MalVis dataset.

Table 5: Performance results of different ensemble methods on the Large-MalVis multiclass dataset after undersampling evaluation.

Ensemble Methods	MalVis-B Multi-Class				
	A	F1	P	R	ROC-AUC
Average Voting	66.25%	65.15%	65.36%	66.25%	81.49%
Majority Voting	63.15%	61.40%	63.70%	63.15%	79.79%
Weighted Voting	64.79%	63.15%	63.90%	64.79%	80.72%
Min Confidence Voting	**88.65%**	**86.32%**	**89.02%**	**88.65%**	86.41%
Soft Voting	66.25%	65.15%	65.36%	66.25%	81.49%
Median Voting	64.51%	63.49%	64.39%	64.51%	80.54%
Rank-Based Voting	63.23%	63.12%	64.34%	63.23%	79.82%
Stacking Ensemble	83.61%	83.33%	83.50%	83.61%	**90.99%**

6 Conclusions

This research establishes the critical importance of visualizing Android malware to safeguard user data and smartphone security. We introduced MalVis, the largest publicly available image-based dataset for Android malware, containing over 1.3 million samples. To complement this resource, we developed a novel visualization framework that transforms bytecode into RGB images by integrating entropy and N-gram encoding techniques. This method effectively captures encryption, compression, structural, and operational anomaly patterns within the malware.

Through extensive evaluation, MalVis consistently outperformed existing visualization-based detection approaches, achieving 95.19% accuracy, 90.81% F1-score, 92.58% precision, 89.10% recall, 87.58% Matthews Correlation Coefficient, and a 98.06% ROC-AUC. Beyond its strong performance, MalVis delivers a conceptually innovative framework that links visual representations to the semantic characteristics of malware, enhancing interpretability and classification robustness. This dataset and framework provide a valuable foundation for advancing research in malware classification, adversarial resilience, and explainable threat detection.

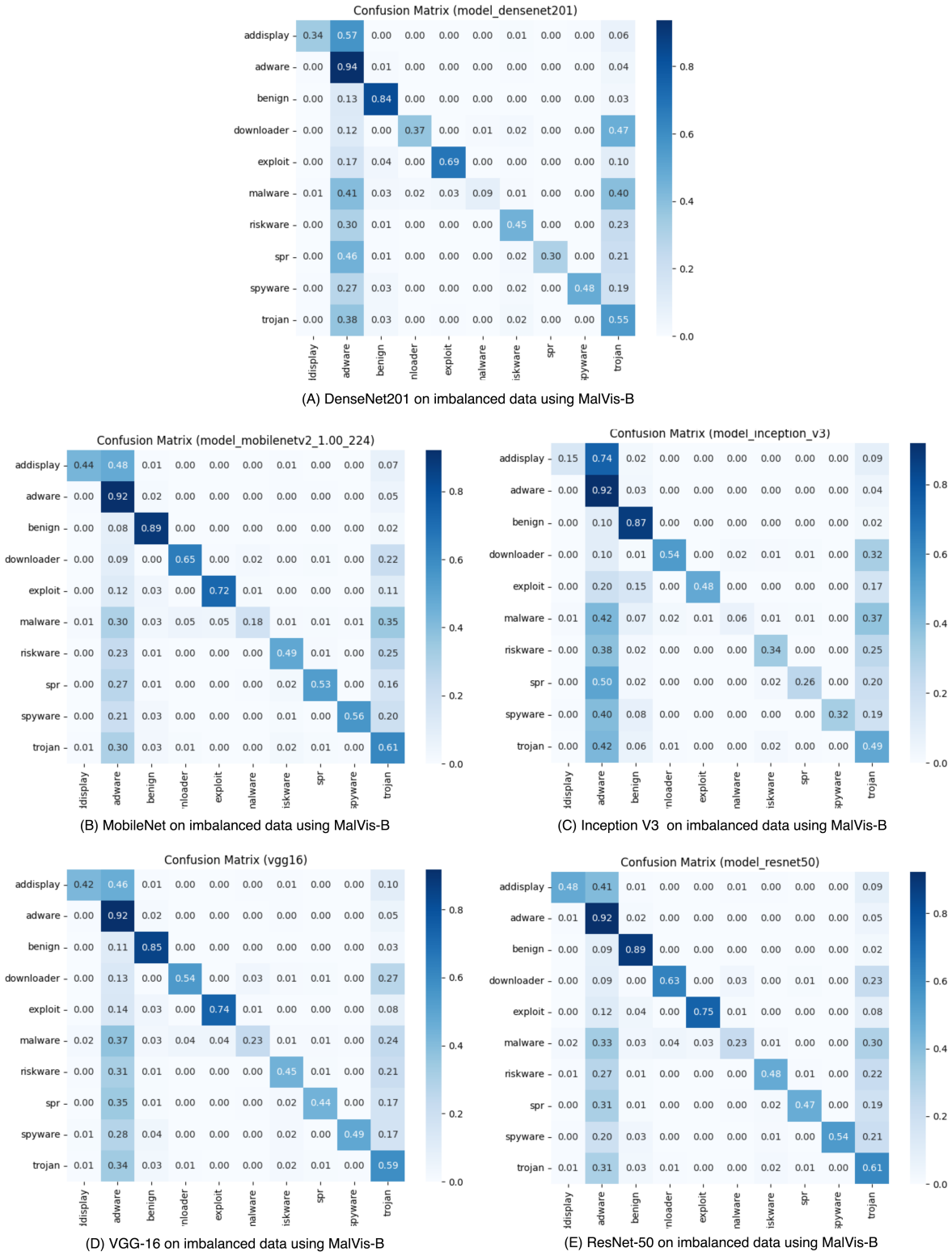
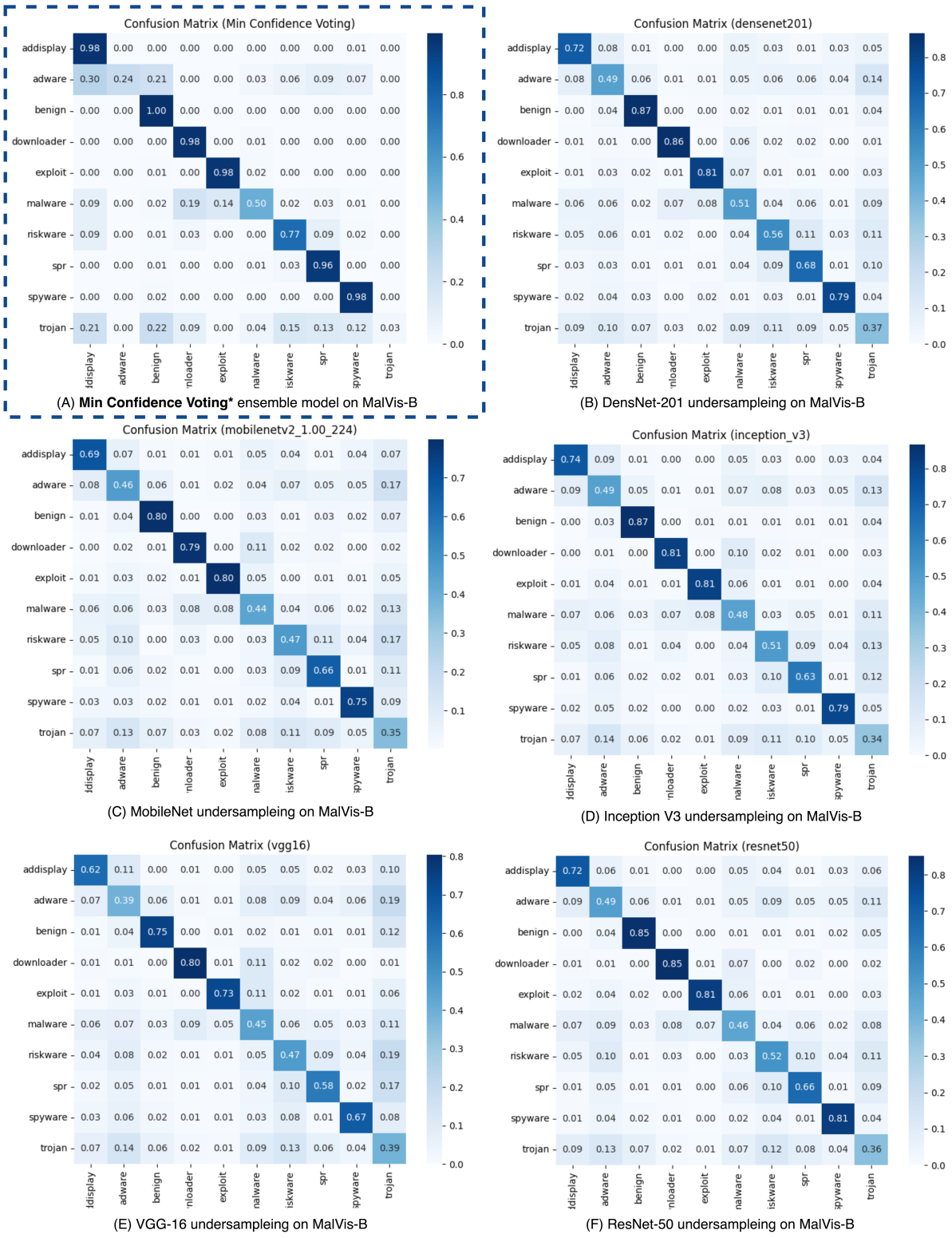


Figure 14: Confusion matrices of CNN models trained on the imbalanced multiclass MalVis dataset with the MalVis-B approach.



(A) **Min Confidence Voting*** ensemble model on MalVis-B

(B) DensNet-201 undersampleing on MalVis-B

(C) MobileNet undersampleing on MalVis-B

(D) Inception V3 undersampleing on MalVis-B

(E) VGG-16 undersampleing on MalVis-B

(F) ResNet-50 undersampleing on MalVis-B

Figure 15: Confusion matrices for CNN models trained on a balanced multiclass MalVis dataset and the optimal ensemble method.

References

[1] Ahmed Sherif. Market share of mobile operating systems worldwide from 2009 to 2024, by quarter. Sep 23, 2024.

[2] Statista. Smartphone operating system share by age group in the u.s. as of december 2023, 2024. Accessed: January 31, 2024.

[3] Verizon Business. Mobile security index (msi) report 2023: Security threats and attacks. https://www.verizon.com/business/resources/reports/mobile-security-index/, 2023. Accessed on 10th January 2024.

[4] David Noever and Samantha E Miller Noever. Virus-mnist: A benchmark malware dataset. *arXiv preprint arXiv:2103.00602*, 2021.

[5] Pu Wang, Marta C González, Cesar A Hidalgo, and Albert-László Barabási. Understanding the spreading patterns of mobile phone viruses. *Science*, 324(5930):1071–1076, 2009.

[6] Darrell M Kienzle and Matthew C Elder. Recent worms: a survey and trends. In *Proceedings of the 2003 ACM workshop on Rapid Malcode*, pages 1–10, 2003.

[7] Seyhmus Yilmaz and Sultan Zavrak. Adware: a review. *International Journal of Computer Science and Information Technologies*, 6(6):5599–5604, 2015.

[8] Supraja Suresh, Fabio Di Troia, Katerina Potika, and Mark Stamp. An analysis of android adware. *Journal of Computer Virology and Hacking Techniques*, 15:147–160, 2019.

[9] Martin Boldt, Bengt Carlsson, and Andreas Jacobsson. Exploring spyware effects. In *Nordsec 2004*, 2004.

[10] Azad Ali. Ransomware: A research and a personal case study of dealing with this nasty malware. *Issues in Informing Science and Information Technology*, 14:087–099, 2017.

[11] Lynn Erla Beegle. Rootkits and their effects on information security. *Information Systems Security*, 16(3):164–176, 2007.

[12] ZHU Zhenfang. Study on computer trojan horse virus and its prevention. *International Journal of Engineering and Applied Sciences*, 2(8):257840, 2015.

[13] Akashdeep Bhardwaj and Sam Goundar. Keyloggers: silent cyber security weapons. *Network Security*, 2020(2):14–19, 2020.

[14] Maryam Feily, Alireza Shahrestani, and Sureswaran Ramadass. A survey of botnet and botnet detection. In *2009 Third International Conference on Emerging Security Information, Systems and Technologies*, pages 268–273. IEEE, 2009.

[15] Nagababu Pachhala, S Jothilakshmi, and Bhanu Prakash Battula. A comprehensive survey on identification of malware types and malware classification using machine learning techniques. In *2021 2nd International Conference on Smart Electronics and Communication (ICOSEC)*, pages 1207–1214. IEEE, 2021.

[16] Shai Halevi and Hugo Krawczyk. Strengthening digital signatures via randomized hashing. In *Annual International Cryptology Conference*, pages 41–59. Springer, 2006.

[17] Gerardo Canfora, Andrea Di Sorbo, Francesco Mercaldo, and Corrado Aaron Visaggio. Obfuscation techniques against signature-based detection: a case study. In *2015 Mobile systems technologies workshop (MST)*, pages 21–26. IEEE, 2015.

[18] Rajan Thangaveloo, Wong Jing, Kang Leng Chiew, and Johari Abdullah. Datdroid: Dynamic analysis technique in android malware detection. *International Journal on Advanced Science, Engineering and Information Technology*, 10:536, 03 2020.

[19] Ya Pan, Xiuting Ge, Chunrong Fang, and Yong Fan. A systematic literature review of android malware detection using static analysis. *IEEE Access*, 8:116363–116379, 2020.

[20] Arshi Dhammi and Maninder Singh. Behavior analysis of malware using machine learning. In *2015 Eighth International Conference on Contemporary Computing (IC3)*, pages 481–486, 2015.

[21] Bertram Poettering and Simon Rastikian. Sequential digital signatures for cryptographic software-update authentication. In *European Symposium on Research in Computer Security*, pages 255–274. Springer, 2022.

[22] Rabia Tahir. A study on malware and malware detection techniques. *International Journal of Education and Management Engineering*, 8(2):20, 2018.

[23] Cuiying Gao, Minghui Cai, Shuijun Yin, Gaozhun Huang, Heng Li, Wei Yuan, and Xiapu Luo. Obfuscation-resilient android malware analysis based on complementary features. *IEEE Transactions on Information Forensics and Security*, 2023.

[24] Wael F Elsersy, Ali Feizollah, and Nor Badrul Anuar. The rise of obfuscated android malware and impacts on detection methods. *PeerJ Computer Science*, 8:e907, 2022.

[25] Dhilung Kirat and Giovanni Vigna. Malgene: Automatic extraction of malware analysis evasion signature. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 769–780, 2015.

[26] Anastasia Pereberina, Alexey Kostyushko, and Alexander Tormasov. An approach to dynamic malware analysis based on system and application code split. *Journal of Computer Virology and Hacking Techniques*, pages 1–11, 2022.

[27] Farhan Ullah, Shamsher Ullah, Gautam Srivastava, Jerry Chun-Wei Lin, and Yue Zhao. Nmal-droid: network-based android malware detection system using transfer learning and cnn-bigru ensemble. *Wireless Networks*, 30(6):6177–6198, 2024.

[28] Rami Sihwail, Khairuddin Omar, Khairul Akram Zainol Ariffin, and Sanad Al Afghani. Malware detection approach based on artifacts in memory image and dynamic analysis. *Applied Sciences*, 9(18):3680, 2019.

[29] Yousef Seyfari and Akbar Meimandi. A new approach to android malware detection using fuzzy logic-based simulated annealing and feature selection. *Multimedia Tools and Applications*, pages 1–25, 2023.

[30] Vera Orlova, Vyacheslav Goiko, Yulia Alexandrova, and Evgeny Petrov. Potential of the dynamic approach to data analysis. In *E3S Web of Conferences*, volume 258, page 07012. EDP Sciences, 2021.

[31] Taniya Bhatia and Rishabh Kaushal. Malware detection in android based on dynamic analysis. In *2017 International conference on cyber security and protection of digital services (Cyber security)*, pages 1–6. IEEE, 2017.

[32] Rinu Rani Jose and A Salim. Integrated static analysis for malware variants detection. In *Inventive Computation Technologies 4*, pages 622–629. Springer, 2020.

[33] Scott Freitas, Yuxiao Dong, Joshua Neil, and Duen Horng Chau. A large-scale database for graph representation learning. *arXiv preprint arXiv:2011.07682*, 2020.

[34] 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. *Transactions on Emerging Telecommunications Technologies*, 31(11):e3789, 2020.

[35] Thomas Sutter, Timo Kehrer, Marc Rennhard, Bernhard Tellenbach, and Jacques Klein. Dynamic security analysis on android: A systematic literature review. *IEEE Access*, 2024.

[36] Kesav Kancherla and Srinivas Mukkamala. Image visualization based malware detection. In *2013 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*, pages 40–44. IEEE, 2013.

[37] Saleh J Makkawy, Abdalrahman H Alblwi, Michael J De Lucia, and Kenneth E Barner. Improving android malware detection with entropy bytecode-to-image encoding framework. In *2024 33rd International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9. IEEE, 2024.

[38] Médéric Hurier, Guillermo Suarez-Tangil, Santanu Kumar Dash, Tegawendé F Bissyandé, Yves Le Traon, Jacques Klein, and Lorenzo Cavallaro. Euphony: harmonious unification of cacophonous anti-virus vendor labels for android malware. In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 425–435. IEEE Press, 2017.

[39] Virustotal - free online virus, malware, and url scanner. https://www.virustotal.com. Accessed: 2024-8-05.

[40] Anamika Dhillon and Gyanendra K Verma. Convolutional neural network: a review of models, methodologies and applications to object detection. *Progress in Artificial Intelligence*, 9(2):85–112, 2020.

[41] Min-Hyeok Sun, Seung-Hyun Kong, and Dong-Hee Paek. A survey on deep learning-based lane detection algorithms for camera and lidar. *IEEE Transactions on Intelligent Transportation Systems*, 2025.

[42] Abdalrahman Alblwi, Saleh Makkawy, and Kenneth E Barner. D-ddpm: Deep denoising diffusion probabilistic models for lesion segmentation and data generation in ultrasound imaging. *IEEE Access*, 2025.

[43] Heng Sun, Miaomiao Chen, Jian Weng, Zhiquan Liu, and Guanggang Geng. Anomaly detection for in-vehicle network using cnn-lstm with attention mechanism. *IEEE Transactions on Vehicular Technology*, 70(10):10880–10893, 2021.

[44] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th international conference on mining software repositories*, pages 468–471, 2016.

[45] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.

[46] Scott Freitas, Rahul Duggal, and Duen Horng Chau. Malnet: A large-scale image database of malicious software. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 3948–3952, 2022.

[47] Tina Rezaei, Farnoush Manavi, and Ali Hamzeh. A pe header-based method for malware detection using clustering and deep embedding techniques. *Journal of Information Security and Applications*, 60:102876, 2021.

[48] 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.

[49] Pradip Kunwar, Kshitiz Aryal, Maanak Gupta, Mahmoud Abdelsalam, and Elisa Bertino. Sok: Leveraging transformers for malware analysis. *arXiv preprint arXiv:2405.17190*, 2024.

[50] Sana Aurangzeb, Muhammad Aleem, Muhammad Taimoor Khan, George Loukas, and Georgia Sakellari. Androdex: Android dex images of obfuscated malware. *Scientific Data*, 11(1):212, 2024.

[51] 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.

[52] Alessandro Panconesi, Marian, Will Cukierski, and WWW BIG Cup Committee. Microsoft malware classification challenge (big 2015). https://kaggle.com/competitions/malware-classification, 2015. Kaggle.

[53] Chenyue Wang, Linlin Zhang, Kai Zhao, Xuhui Ding, and Xusheng Wang. Advandmal: Adversarial training for android malware detection and family classification. *Symmetry*, 13(6):1081, 2021.

[54] Halil Murat Ünver and Khaled Bakour. Android malware detection based on image-based features and machine learning techniques. *SN Applied Sciences*, 2(7):1299, 2020.

[55] 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.

[56] Changguang Wang, Ziqiu Zhao, Fangwei Wang, and Qingru Li. A novel malware detection and family classification scheme for iot based on deam and densenet. *Security and Communication Networks*, 2021(1):6658842, 2021.

[57] Asim Darwaish and Farid Naït-Abdesselam. Rgb-based android malware detection and classification using convolutional neural network. In *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pages 1–6. IEEE, 2020.

[58] Setia Juli Irzal Ismail, Budi Rahardjo, Tutun Juhana, Yasuo Musashi, et al. Malssl–self-supervised learning for accurate and label-efficient malware classification. *IEEE Access*, 2024.

[59] Anthony Desnos/ Google.com. Androguard tool by google. https://code.google.com/archive/p/androguard/, Feb 13, 2013 / 1st January 2024. Accessed: on 8th January 2024.

[60] Aldo Cortesi. (binvis) a library for drawing space-filling curves like the hilbert curve. https://github.com/cortesi/scurve, 2015.

[61] Muhammad Ali, Stavros Shiaeles, Gueltoum Bendiab, and Bogdan Ghita. Malgra: Machine learning and n-gram malware feature extraction and detection system. *Electronics*, 9(11):1777, 2020.

[62] Fangtian Zhong, Qin Hu, Yili Jiang, Jiaqi Huang, Cheng Zhang, and Dinghao Wu. Enhancing malware classification via self-similarity techniques. *IEEE Transactions on Information Forensics and Security*, 2024.

[63] Iman Almomani, Aala Alkhayer, and Walid El-Shafai. An automated vision-based deep learning model for efficient detection of android malware attacks. *IEEE Access*, 10:2700–2720, 2022.

[64] Roweida Mohammed, Jumanah Rawashdeh, and Malak Abdullah. Machine learning with oversampling and undersampling techniques: overview study and experimental results. In *2020 11th international conference on information and communication systems (ICICS)*, pages 243–248. IEEE, 2020.

[65] Anjana Gosain and Saanchi Sardana. Handling class imbalance problem using oversampling techniques: A review. In *2017 international conference on advances in computing, communications and informatics (ICACCI)*, pages 79–85. IEEE, 2017.