IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 4 October 2024, accepted 18 October 2024, date of publication 23 October 2024, date of current version 1 November 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3485593

RESEARCH ARTICLE

DeepImageDroid: A Hybrid Framework Leveraging Visual Transformers and Convolutional Neural Networks for Robust Android Malware Detection

COLLINS CHIMEZIE OBIDIAGHA, MOHAMED RAHOUTI, (Member, IEEE), AND THAIER HAYAJNEH, (Senior Member, IEEE)
Department of Computer and Information Science, Fordham University, New York City, NY 10023, USA
Corresponding author: Thaier Hayajneh (thayajneh@fordham.edu)

ABSTRACT As the leading mobile operating system, Android powers critical infrastructure and personal devices across sectors such as finance and healthcare and a wide range of user scenarios. However, its open-source nature presents considerable security challenges. Exploiting vulnerabilities in native and custom permissions, malicious API calls, intents, signatures, and manifests, threat actors can gain access to sensitive data and device control. The continuously evolving landscape of Android malware necessitates robust and generalized detection methods. While traditional machine learning (ML) models have been employed to address this issue, they have limitations. Focusing on single datasets can impede both generalization and effective detection. Further, the dynamic nature of malware renders many traditional methods inadequate for providing comprehensive and real-time protection. This paper addresses this critical need by proposing DeepImageDroid, an advanced and efficient deep learning (DL) framework for Android malware detection. DeepImageDroid harnesses the combined power of Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs), utilizing three diverse Android malware datasets. This hybrid approach significantly improves detection accuracy and model generalization compared to existing solutions. By employing the weighted average ensemble method, DeepImageDroid achieved a remarkable 96% accuracy.

INDEX TERMS Android, CNN, deep learning, ensemble, hybrid, malware, security, ViTs.

I. INTRODUCTION

The Android operating system (OS) continues to dominate a wide range of devices, and its significance on a global scale cannot be understated. Thus, it is crucial to implement measures that mitigate the vulnerabilities inherent in Android devices and enhance the security ecosystem. Android's most prominent application is in mobile devices, including smartphones, wearable technology, and even medical and health-related equipment.

The open-source nature of Android has propelled it to the forefront of the smart device market, boasting over 2 billion active users and a commanding 74% market share. However, this widespread popularity also presents a considerable security challenge. The extensive user base makes Android a prime target for malware developers who exploit the platform's accessibility to create sophisticated threats. These malicious programs often aim to compromise the device or the sensitive data it stores. They can leverage system resources such as cameras, microphones, and location data or directly access private information like contacts, emails, and messages. Malware can also infiltrate legitimate applications by repackaging them with malicious code or requesting excessive permissions during updates, effectively transforming them into Trojan horses [1], [2].

Recent research underscores the significant threat malicious actors pose targeting Android's vast user base. Studies indicate that personal privacy theft is a primary objective

The associate editor coordinating the review of this manuscript and approving it for publication was Bing Li.

in Android malware attacks, with nearly half of all such malware functioning as multifunctional Trojans designed to steal user data. This threat is exacerbated by the sheer volume of applications available on the official Google Play store, which exceeded 3 million by the end of 2020 [2], [3]. Several factors contribute to Android's vulnerability, including:

- Environmentally accessible methods: Android applications can access various sensors and functionalities within the device, offering potential attack vectors.
- Coarse-grained permission control: The permission system can be exploited due to its limited granularity, granting potentially broader access than necessary.
- Third-party code execution: The ability to execute code from external sources introduces additional security risks [3].

These vulnerabilities collectively create a vast attack surface for malicious actors, jeopardizing the integrity and security of Android apps. The alarming statistics emphasize the severity of the issue, with over 3.25 million infected apps discovered in 2016 alone, translating to roughly one new malicious app every 10 seconds [3], [4]. Thus, it is imperative to address these security challenges to protect users and maintain the integrity of the Android ecosystem.

Further, Android malware detection faces a continuous battle against the ever-evolving tactics of malicious actors. Malware authors employ sophisticated obfuscation techniques such as encryption, polymorphism, and metamorphic methods, making their creations increasingly difficult to detect [4], [5], [6]. This challenge is exacerbated by recent malware programs that mimic popular apps, incorporating subtle similarities to evade human detection [4].

Traditional approaches, which rely on feature extraction and machine learning (ML) classifiers, often fall short in this dynamic landscape. While these methods can achieve high accuracy, they remain vulnerable to attackers who add benign features like pop-up messages or logging, thereby tricking the classifier into categorizing malicious apps as benign [4]. Moreover, traditional algorithms struggle to learn complex patterns in high-dimensional data, limiting their effectiveness.

The effectiveness of these models is also heavily dependent on the quality and relevance of the training data. Models trained on a specific dataset often become obsolete as new Android apps are developed and software evolves [6]. This necessitates constant retraining and adaptation, which poses a significant challenge for real-world deployment. To address these issues, developing more robust and adaptable malware detection methods is essential to keep pace with the rapid evolution of threats in the Android ecosystem.

Our approach to addressing the challenges in Android malware detection leverages the power of Deep Neural Networks (DNNs), a leading force in the ML landscape. Unlike traditional methods that require manual feature engineering, DNNs automatically extract features directly from raw data. This capability allows them to learn intricate representations with minimal prior knowledge, enhancing their ability to detect sophisticated and evolving malware threats [6], [7], [8].

The inherent capability of DNNs proves invaluable in the fight against malware, allowing them to detect even subtle changes employed by malicious actors. By training on large datasets, our model can effectively discern the true nature of an application despite the presence of obfuscating benign features. Previous research has demonstrated the effectiveness of DNN-based image classifiers in identifying malware, and our approach builds upon this success. Moreover, DNNs offer the exciting potential to detect never-before-seen malware variants, a crucial advantage in the ever-evolving malware landscape [4]. Motivated by the growing threat of Android malware and the limitations of traditional detection methods, we introduce DeepImageDroid, a novel framework for malware detection and classification. This framework leverages the power of deep learning (DL) to achieve robust and generalized performance.

DeepImageDroid's key strength lies in its diverse training data, utilizing three distinct datasets encompassing a wide range of malware and benign samples. This varied data exposure equips the model to learn comprehensive representations and generalize effectively to unseen threats. DeepImageDroid employs a hybrid approach, combining two robust DL architectures: Visual Transformer (ViT) and Convolutional Neural Network (CNN). ViT excels at capturing long-range dependencies within data, while CNNs effectively learn local patterns. By combining these complementary strengths, DeepImageDroid gains a deeper understanding of both global and local features, leading to more accurate and robust malware identification. In our experimentation, the CNN model achieved an impressive 96% accuracy in malware detection, while the ViT architecture closely followed with 95% accuracy. Recognizing the potential synergy between these architectures, we combined them through a weighted average ensemble method, resulting in DeepImageDroid achieving an outstanding 96% accuracy. This significant improvement underscores the effectiveness of our hybrid approach, paving the way for a more robust and generalizable solution to Android malware detection.

The key contributions of this paper can be outlined as follows:

- We curate three diverse Android malware binary datasets and convert them into grayscale images for comprehensive data representation, enabling the identification of subtle malicious patterns.
- We develop a resilient ViT model tailored for Android malware detection, demonstrating ViT's effectiveness in analyzing spatial relationships and long-range dependencies.
- We propose DeepImageDroid, a novel hybrid model combining CNNs and ViTs, achieving an impressive accuracy of 96% using the weighted average ensemble method and leveraging the unique strengths of each DL architecture.

- We leverage diverse training data from three distinct datasets to equip the model with comprehensive representations, enabling effective generalization to unseen threats.
- We transform large datasets of binary malware and benign features into grayscale images, facilitating smoother processing and valuable insights for DL models.
- We significantly improve malware detection accuracy and generalization compared to existing solutions, underscoring the hybrid approach's effectiveness in combating Android malware using grayscale images.

Our comprehensive approach, efficient data utilization, novel model introduction, and superior accuracy set a new benchmark for Android malware detection using DL. It opens doors for further exploration of visual representations and hybrid models, paving the way for more robust and generalizable solutions against the ever-evolving threat of Android malware.

The rest of this article is structured as follows: Section II summarizes the state-of-the-art related to our work. Next, Section III provides background information on Android malware and DNNs with grayscale images. In Section IV, we present the details of our methodology, while Section V expands on our experiments and results. Last, Section VI concludes this work.

II. RELATED WORK

Our research focuses on detecting and classifying Android malware utilizing DL techniques. Specifically, we leverage CNN and ViT models and subsequently develop an ensemble model that integrates these approaches. In this section, we review several relevant studies that employ CNN, ViT, and ensemble methods for Android malware detection and delineate how our work differentiates itself from these existing efforts.

A. CONVOLUTIONAL NEURAL NETWORKS (CNNs)

Many researchers have utilized CNNs for malware detection using grayscale or RGB images, employing various techniques and hyperparameter tuning to achieve optimal results. Kinkkead et al. [9] investigated the significance of CNNs in identifying critical locations in an Android app's opcode sequence contributing to malware detection. They identified essential locations within opcode sequences and compared these with those identified by the state-of-the-art explainability method LIME using the Drebin benchmark dataset. The study revealed that the locations identified as most malicious by their CNN closely aligned with those highlighted by LIME, instilling confidence in CNN's ability to focus on patterns of malicious opcodes in Android apps.

Building on using CNNs, Vu and Jung [10] proposed AdMat, a framework that treats Android applications as images by creating an adjacency matrix for each application. These matrices serve as ''input images'' for a CNN model, enabling it to discern between benign and malicious apps and identify specific malware families. AdMat demonstrated adaptability across various training ratios, achieving an average detection rate of 98.26% across different malware datasets and successfully recognizing over 97% of different malware families even with limited training data. Similarly, Nicheporuk et al. [11] introduced a CNN mixed-data model that utilizes API method calls and a set of permissions associated with Android applications, represented using Word2vec technology and binary features, respectively. This novel approach leverages the strengths of CNNs and integrates multiple data sources to enhance the efficacy of Android malware detection. Ganesh et al. [12] focused on investigating permission patterns using CNNs, demonstrating a robust performance with a 93% accuracy rate in identifying and categorizing malicious applications within a dataset of 2500 Android applications. This DL-based method showcases the potential for enhanced mobile security through accurate malware detection.

Expanding on the innovative uses of CNNs, Xiao and Yang [13] employ an innovative approach to address challenges in existing malware detection techniques, particularly data obfuscation and limited code coverage. Their method uses CNNs to learn features directly from Dalvik bytecode, overcoming obfuscation limitations and achieving comprehensive code coverage. This approach facilitates the automatic extraction of meaningful features, enabling the model to distinguish between benign and malicious applications with an average detection time of 0.22 seconds and an overall accuracy surpassing 93%.

Similarly, Zhang et al. [14] introduce DeepClassifyDroid, a DL-based system for Android malware detection. It employs a three-step feature extraction, embedding, and detection approach using CNNs. Evaluated against various ML methods, DeepClassifyDroid achieved a 97.4% detection rate with minimal false alarms, demonstrating superior efficiency by being 10 times faster than linear-SVM and 80 times faster than kNN. Addressing computational demands, Hasegawa and Iyatomi [15] present a lightweight method that analyzes a minimal portion of the APK file through a 1-D CNN, achieving an accuracy of 95.40% to 97.04% using 10-fold cross-validation on datasets of 5,000 malware and 2,000 goodware samples.

B. VISUAL TRANSFORMERS (ViTs)

While ViT architectures have gained traction in various domains, their application for Android malware detection and classification still needs to be explored. Our review suggests that our work is among the few that specifically adapt ViT for Android malware. Researchers have explored ViT variants for broader malware detection tasks, though not tailored to Android's unique challenges. Rahali and Akhloufi [16] introduced MalBERT, a model built upon BERT, for automating the detection of malicious software through static analysis of Android application source code. MalBERT utilizes preprocessed features to classify malware into representative

categories, leveraging transformer architectures beyond NLP. The results from MalBERT are promising, highlighting the high performance of transformer-based models in malware detection.

In another study, Jo et al. [17] introduce ViT-MalDetect, a ViT-based malware detection model focused on high detection accuracy and interpretability. The model leverages ViT's attention mechanisms to interpret and understand intricate patterns in application images robustly, achieving a detection accuracy of 80.27% on real-world datasets. Ravi et al. [18] present ViT4Mal, a lightweight ViT approach for malware detection on edge devices. By converting executable bytecode into images, ViT4Mal facilitates malware feature learning and employs a customized ViT for accurate detection within resource constraints. Extensive experiments show that ViT4Mal achieves comparable accuracy to state-of-the-art CNNs, reaching around 97%, without requiring deeper networks.

Further expanding using ViTs, Seneviratne et al. [4] introduce SHERLOCK, a cutting-edge malware detection model based on ViT architecture and self-supervised learning. SHERLOCK leverages image-based binary representation to distinguish malware from benign programs without needing labeled data. This model captures intricate patterns in Android applications, achieving a remarkable 97% accuracy for binary classification and outperforming state-of-the-art techniques in multi-class malware classification with macro-F1 scores of .497 and .491 for types and families, respectively. SHERLOCK demonstrates the potential of self-supervised learning and ViT architecture in advancing Android malware detection, positioning itself as a formidable tool in cybersecurity. This study highlights the promise of ViTs in improving Android malware detection in the cybersecurity landscape. Although ViTs are relatively unexplored in Android malware detection, our study is among the few utilizing them. This integration of CNNs and ViTs forms the basis of our robust hybrid model.

C. ENSEMBLE/HYBRID TECHNIQUE

Ensemble methods, which combine the strengths of diverse ML and DL architectures, offer another avenue for Android malware detection and classification. Our study leverages this approach, specifically employing a weighted average ensemble that integrates the unique capabilities of CNNs and ViTs.

Building on this concept, Wang et al. [19] propose a hybrid model to enhance the accuracy and efficiency of large-scale Android malware detection by combining a deep autoencoder (DAE) and a CNN. This approach addresses the challenges of high-dimensional features in Android applications. The model reconstructs high-dimensional features and employs multiple CNNs. In their serial CNN architecture (CNN-S), they use the ReLU activation function for increased sparsity and incorporate dropout to prevent overfitting. Combining convolutional and pooling layers with a fully connected layer enhances feature extraction capabilities. To expedite training, they introduce a deep autoencoder as a pre-training method for the CNN. The hybrid model, DAE-CNN, learns flexible patterns more efficiently. Experimental evaluations on 10,000 benign and 13,000 malicious apps show improvements: CNN-S achieves a 5% accuracy improvement over traditional ML methods like SVM, while DAE-CNN reduces training time by 83% compared to CNN-S.

Similarly, Pei et al. [20] proposed MalNet, a stacked ensemble model based on CNN and LSTM for classifying malicious files. Designed to address the complexities of Android malware detection and family attribution, MalNet leverages Graph Convolutional Networks (GCNs) to model high-level graphical semantics and identify semantic and sequential patterns. An Independently Recurrent Neural Network (IndRNN) also decodes deep semantic information, extracting features independently while utilizing remote dependent information between nodes. Experimental results on multiple benchmark datasets highlight MalNet's superiority over other state-of-the-art techniques, showcasing its robustness and performance.

In another study, Lin and Chang [21] introduce a Selective Deep Ensemble Learning-based (SDEL) detector, leveraging ensemble learning for enhanced detection accuracy. They designed an Ensemble Deep Taylor Decomposition (EDTD) approach to provide pixel-level explanations for the SDEL detector's outputs. To augment model interpretability, they introduced a novel Interpretable Dropout approach (IDrop) and trained on the SDEL detector with insights from the EDTD explanation. Experimental results demonstrate the superior explanation quality of EDTD compared to previous methods in image-based malware detection. Furthermore, IEMD achieves an impressive detection accuracy of up to 99.87%, maintaining interpretability with high-quality prediction results.

Further contributing to the field, Pierazzi et al. [22] conducted a data-driven characterization of modern Android spyware from July 2016 to July 2017, utilizing both traditional and deep ML approaches. The authors introduce an innovative Ensemble Late Fusion (ELF) architecture, which combines the predicted probabilities of multiple classifiers to generate a final prediction. ELF outperforms several well-known traditional and DL classifiers, demonstrating its effectiveness in Android spyware detection. Additionally, the study automatically identifies key features that distinguish spyware from goodware and other malware, offering valuable insights into the distinct characteristics of these malicious entities.

1) COMPARISON OF DeepImageDroid WITH STATE-OF-THE-ART METHODS

Ensemble methods have emerged as a promising approach for enhancing the accuracy and robustness of Android malware detection systems. Our proposed DeepImageDroid model leverages this paradigm by combining the strengths of CNNs and ViTs through a weighted averaging ensemble. While

TABLE 1. A summary of related works.

Ref.	Focus	DL Method	Data Representation	Strengths	Limitations
Kinkkead et al. [9]	Identifying critical locations in opcode sequence	CNN	Opcode sequence	Aligns with explainability methods	Focuses only on opcode sequence
Vu et al. [10]	Treating apps as images using adjacency matrix	CNN	Adjacency matrix	High accuracy (98.26%) family recognition	Requires generating processing adjacency matrix
Nicheporuk et al. [11]	Mixed-data model for API calls permissions	CNN	API calls (Word2vec) Permissions (binary)	Integrates multiple data sources	Might be complex for interpretability
Ganesh et al. [12]	Identifying permission patterns	CNN	Permissions	High accuracy (93%)	Limited to permission data
Xiao et al. [13]	Feature learning from Dalvik bytecode	CNN	Dalvik bytecode	Efficient (0.22s) high accuracy (93%)	Might be vulnerable to obfuscation
Zhang et al. [14]	Feature extraction DL-based detection	CNN	Feature sets from static analysis	High accuracy (97.4%) fast (10x faster than Linear-SVM)	Feature engineering required
Hasegawa et al. [15]	Lightweight 1D CNN for minimal APK analysis	1D CNN	Last 512-1K bytes of APK file	High accuracy (95.40%-97.04%) lightweightLimited to APK file snippet	Limited to APK file snippet
Rahali et al. [16]	Source code analysis, classification	Transformer model (BERT), a variant of ViTs architecture	Source code (preprocessed features)	High performance for malware detection	Not tailored to Android malware
Jo et al. [17]	ViT-based malware detection, interpretability	ViT	App images	Interpretability, accuracy (80.27%)	Accuracy could be better
Ravi et al. [18]	Lightweight ViT for malware detection on edge devices	Lightweight ViT	Executable bytecode converted to images	Lightweight and efficient, accuracy 97%	Requires bytecode to image conversion
Seneviratne et al. [4]	Self-supervised ViT for malware detection	ViT	Android application binaries (image-based representation)	High accuracy (97%), self-supervised learning	Requires and relies on large unlabeled datasets
Wang et al. [19]	Large-scale detection efficiency	Hybrid DAE-CNN	High-dimensional features	Improved accuracy (98.60%) reduced training time	Might require more training data Used a few pre-trained models
Pei et al. [20]	Malware family classification interpretability	Combined techniques: GCNs, IndRNN CNN, LSTM Stacked ensemble for classification: CNN LSTM	Code, API calls, permissions intents	Robustness performance	Complex architecture
Lin et al. [21]	Ensemble learning for accuracy interpretability	Stacked CNN Models	Grayscale images without binary raw files from 25 families	High accuracy (99.87%) model interpretability/explainability	Might be computationally expensive, Not dealing with Android Malware
Pierazzi et al. [22]	Spyware detection	Ensemble Late Fusion (ELF): Combination of traditional ML and DL classifiers (MLP + CNN+BRBM.	Static and dynamic Features	Outperforms traditional DL methods in spyware and goodware detection	Limited to spyware detection
This paper	Generalizability on diverse Malware datasets and robust preprocessing techniques for Android malware detection	Ensemble Hybrid model through weighted averaging: ViTs CNN	Binary to grayscale images of Android Malware features (API calls, permission, Intent, etc.)	Robustness, high accuracy, generalizability Accuracy of 95.63% (ViT) + 96.11% (CNN) = 96.00% (DeepImageDroid)	Ensemble model accuracy could be improved

several studies have explored ensemble-based techniques for this task, DeepImageDroid distinguishes itself through its unique architecture and performance.

Comparing these recent models with DeepImageDroid reveals the robustness and effectiveness of our proposed hybrid ensemble approach. Unlike Wang et al.'s [19] DAE-CNN, which primarily focuses on reducing training time, DeepImageDroid emphasizes balanced accuracy and robustness by combining CNNs' local feature extraction with ViTs' global dependency-capturing capabilities. Compared to Pei et al.'s [20] MalNet, which incorporates complex sequential patterns using LSTMs and GCNs, DeepImageDroid achieves comparable performance using a more straightforward yet robust hybrid of CNN and ViT architectures. Lastly, Lin et al.'s [21] SDEL detector, which integrates interpretability through EDTD and IDrop, aligns with our goal of enhancing malware detection accuracy. However, DeepImageDroid's weighted averaging technique for combining model predictions ensures a more robust ensemble model that effectively reduces false positives and negatives

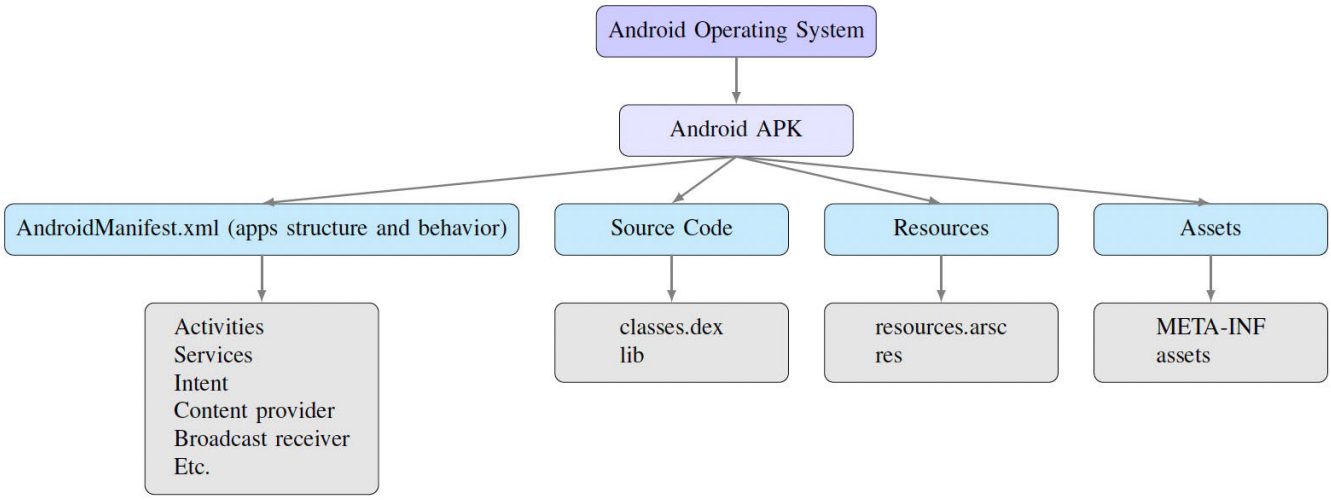

FIGURE 1. Android operating system high level files system.

and prioritizes the most reliable model predictions. These comparisons underscore the robustness of DeepImageDroid, highlighting its superior ability to integrate diverse model strengths for enhanced Android malware detection.

D. THE UNIQUENESS OF OUR STUDY

The proposed work distinguishes itself from the related works by focusing on the unique challenges of Android malware detection through a hybrid deep learning approach that combines the strengths of ViTs and CNNs. While the existing literature predominantly addresses intrusion detection in network environments, such as IoT and fog-cloud systems, as explored by Li et al. [23] and Binbusayyis [24], the proposed work explicitly targets the Android platform, which poses distinct challenges due to its open-source nature and widespread adoption. Unlike Jemili et al. [25], who utilize ensemble learning for big data classification, and Horchulhack et al. [26], who focus on network-based intrusion detection using image-based CNNs and transfer learning, our work uniquely integrates ViTs to capture global dependencies in data alongside CNNs for local feature extraction. This combination allows for a more comprehensive analysis of Android malware patterns.

Further, while Dai et al. [27] leverage CNNs, BiLSTM, and attention mechanisms to enhance temporal and spatial feature detection in network intrusions, our approach stands out by employing a novel weighted average ensemble method, improving the model's generalizability and robustness against diverse malware threats. Thus, the proposed work contributes to the field by providing a specialized and highly effective solution tailored for Android malware detection, expanding the applicability of hybrid deep learning models beyond traditional network-based intrusion scenarios.

A summary of the related works is given in Table 1. Building upon these foundational studies, our research introduces several novel elements in the domain of Android malware detection, setting it apart from existing works. Unlike methods that rely on static analysis and single models, we utilize grayscale images for greater generalizability and employ a hybrid architecture combining ViTs and CNNs. This approach potentially leads to improved feature learning and model robustness. While some methods convert applications to grayscale images, we operate on raw binary files, extracting broader information within the binaries. This allows for a more comprehensive analysis than approaches focusing solely on specific patterns within the images. Our combination of ViTs and CNNs enhances the models' complexity and feature learning capabilities, achieving an accuracy of 95.35%, surpassing models' performance using a single architecture.

Lastly, compared to hybrid ensemble methods that do not utilize images and are based on fewer datasets, our extensive use of diverse datasets and preprocessing binaries into grayscale images contributes to robustness and generalizability in detecting unknown Android malware. Our hybrid DeepImageDroid model demonstrates strong performance with an accuracy of 96%, highlighting this novel approach's potential and exceeding the results of most hybrid ensemble studies in Android malware detection and classification.

III. BACKGROUND

This section delves into the underlying structure of the Android OS and its Android Package Kits (APKs). It explores the inherent security challenges lurking within APKs, highlighting the specific features we leverage for our study. Additionally, it justifies the suitability of DL approaches for detecting and classifying Android malware, providing context for our chosen methods. Understanding these key concepts is crucial for comprehending the foundation and potential of our proposed approach.

A. ANDROID OS

Fig. 1 provides a high-level taxonomy of the Android OS components, illustrating the structure of an Android APK (application package) and its key elements such as the AndroidManifest.xml file, source code, resources, and assets. Each component is further detailed to show the specific files and functionalities it includes, emphasizing the organized framework within which Android applications operate. Applications are distributed through APKs, each containing code, resources, and a manifest file outlining permissions and components. While this openness empowers developers, it also creates attack vectors for malicious actors. Despite security measures, APKs remain susceptible to manipulation. Code obfuscation, hidden resources, and fake signatures are just a few tricks malware employs to evade detection. The sheer volume and diversity of new APKs further pose a constant challenge for traditional security approaches [10], [22], [28], [29].

1) ANDROID APPLICATION PACKAGES (APK)

Android applications are packaged as compressed files known as APKs. Each APK must contain essential elements for installation: program code, developer certificates, and a Manifest file. This Manifest serves as a blueprint, specifying the app's components and the permissions it needs to access system resources. Crucially, Android enforces a permission-based security model. Apps can only access specific resources if they explicitly request the corresponding permissions within the Manifest file. Users are responsible for granting or denying these permissions during installation or runtime, offering control over their data and devices.

Furthermore, the *AndroidManifest.xml* file holds critical information about an app's structure and behavior. This file can be used to define key components like activities (user interfaces), services (background processes), intents (communication messages), content providers (data exchange), and broadcast receivers (message listeners). Each component plays a specific role within the app, and their presence and configuration can indicate potential security concerns [22], [28].

Further, analyzing the contents of the *AndroidManifest.xml* file provides valuable insights for identifying malicious applications. Different features can be extracted based on the specific role and purpose of each component listed within the file. This means the methods for extracting and representing these features will vary depending on the analyzed component. However, it is crucial to emphasize that malware with root privileges on the device must not include all the information mentioned above in the Manifest. By carefully examining these features and their relationships, researchers can develop more effective tools for detecting and mitigating malware threats on Android devices [10], [22].

2) DETAILED ANALYSIS OF APK STRUCTURE

While the *AndroidManifest.xml* file provides valuable information for malware detection, it's just one piece of the puzzle. To gain a comprehensive understanding of an app, it's crucial to analyze other essential files within the APK. Key elements are outlined as follows.

a: SOURCE CODE

- classes.dex: This file contains the compiled application code, typically in the Dalvik Executable (DEX) format. Examining the code structure and functionalities can reveal suspicious behavior or hidden features.
- lib: Native libraries, often written in C or C++, are included here. These libraries can bypass security restrictions, so their presence and purpose warrant investigation.

b: RESOURCES

- res: This directory stores uncompiled resources like images, layouts, and strings. Analyzing these resources for unusual content or inconsistencies can help identify malicious behavior.
- resources.arsc: This compiled file holds processed resource information. Identifying inconsistencies between uncompiled and compiled resources might indicate attempts to obfuscate malicious content.

c: ASSETS

- assets: This directory contains raw assets like configuration files or data used by the app. Examining these assets can reveal hidden functionalities or sensitive information that might be misused for malicious purposes.

d: SECURITY AND SIGNATURES

- META-INF: This directory holds critical security information, including digests and signatures for verifying the app's integrity and authenticity. Malicious actors might tamper with these signatures, highlighting the need for careful validation [10], [14], [28].

Analyzing these diverse APK components with the Manifest file paints a more comprehensive picture of an app's behavior and potential security risks. By understanding the roles and functionalities of each file type, more robust and effective methods for detecting and preventing malware threats on Android devices can be developed.

B. DEEP LEARNING FOR ANDROID MALWARE DETECTION

Our approach to addressing the Android malware detection challenge leverages DL techniques to achieve optimal solutions to the ever-evolving malware threat landscape. While traditional ML algorithms such as Support Vector Machines (SVMs), Random Forests, and XGBoost have demonstrated their efficacy, they fall short in handling intricate patterns within high-dimensional data when faced with rapidly evolving Android malware [6], [30]. Besides, inspired by the mammalian visual cortex, CNNs are a powerful class of DL models used in image recognition tasks. Their architecture uses hierarchical convolutional and fully connected layers, with convolutional layers acting as

feature detectors that extract increasingly complex features. These features are then combined and classified by the fully connected layers. This ability to automatically learn relevant features makes CNNs ideal for image classification and object detection [31], [32].

Several key factors contribute to these challenges. Traditional ML approaches often rely on manually crafted features, a labor-intensive process requiring significant expertise. High-dimensional data, such as APKs, present a substantial obstacle, necessitating considerable human effort to effectively identify and transform relevant features. Additionally, the dynamic nature of the Android ecosystem means that training data quickly becomes outdated as applications evolve and development practices change. This necessitates constant retraining of models to maintain accuracy, which is a resource-intensive endeavor. Furthermore, attackers continuously devise new techniques to bypass security measures, rendering even well-trained traditional ML models susceptible to evolving threats and leaving users and businesses vulnerable [6], [30], [33].

Given these limitations, constructing robust and transparent defenses against Android malware using traditional ML techniques alone has become increasingly challenging. DL emerges as a compelling alternative, offering several key advantages. As a powerful subset of artificial neural networks, DL leverages massive datasets to effectively address classification and prediction tasks. Its prominence is underscored by its foundational role in transformative technologies such as computer vision, natural language processing, self-driving cars, fraud detection, and malware detection [33].

The unique architecture of DL models, characterized by multiple layers of interconnected processing units, allows for the automatic learning of increasingly abstract data representations. This capability enables machines to learn directly from raw data, extracting critical features necessary for accurate detection and classification without the need for manual feature engineering [30]. Moreover, the continuous learning mechanism inherent in DL models ensures their adaptability to evolving threats, maintaining an edge over attackers' tactics. The ability of DL to discern intricate patterns enhances its resilience against obfuscation and other techniques employed by malware authors [6], [33]. Our study employed CNN and ViT DL architectures to achieve these objectives.

C. VISUALIZING MALWARE AS IMAGE

Image-based malware detection has gained significant traction in recent years, leveraging the strengths of DL models adept at processing visual data. Visualization techniques, particularly those creating image representations of malware, have become valuable tools in malware research. Researchers in the Android and Windows-based malware space, such as [4], [34], [35], [36], [37], and [38], employed image processing to convert binaries into grayscale images or RBG, effectively reducing the need for expensive feature engineering and specialized knowledge. This method's success in accurately classifying malware based on grayscale and RGB features highlighted the potential of using malware as images for more advanced research and detection techniques.

Further, studies have shown that image-based techniques can significantly reduce preprocessing requirements. Researchers can directly feed this information into powerful models like ViTs and CNNs by representing malware as grayscale pixel data. This approach streamlines the analysis process and unlocks the ability to learn complex patterns within the data, leading to potentially more accurate and robust malware detection systems [6], [33].

ViTs have gained traction in computer vision due to their impressive performance and efficient computation compared to traditional CNNs. ViTs employ a unique architecture based on self-attention and point-wise fully connected layers in both encoder and decoder modules. Self-attention lets ViTs capture global dependencies within the input data, enabling them to learn complex relationships between image elements. ViTs split the input image into smaller patches for patch processing, transforming these into embeddings for further processing. Although the decoder is not used directly for malware detection, it helps during training by encouraging the model to learn informative features. ViTs offer several advantages for Android malware detection. The self-attention mechanism allows them to capture more global context within images, which is crucial for accurate classification [39].

Additionally, skip connections in early layers facilitate better information propagation, leading to more robust feature learning and improved performance [18], [39]. ViTs are not yet widely popular in Android malware detection despite their advantages. To the best of our knowledge, our study is among the few that explore ViTs for this purpose [4].

IV. METHODOLOGY AND FRAMEWORK DESIGN

This section discusses the datasets used in this work, including their sources, sizes, and compositions. We elaborate on the pre-processing steps and provide details about the DL models we leverage, including the novel DeepImageDroid hybrid model proposed for this work. Further, we describe the training process of our models, including parameters, hyperparameter tuning strategies, and evaluation metrics.

A. DATASETS

The experiments conducted to assess DeepImageDroid utilized three datasets sourced from distinct collections of Android app samples. Table 2 provides comprehensive details for each dataset. The initial one, Drebin-215 [40], comprises vectors of 215 features extracted from 15,036 app samples. Among these, 9,476 were identified as benign, while 5,560 were classified as malware. The second dataset, Android-HealthCheck [41], encompasses various flavors and diverse Android malware APK files, featuring 215 distinct features derived from 16,300 APK files. Within this set, 15,506

TABLE 2. Datasets used for DeepImageDroid experiments.

Datasets	Samples	Malware	Benign	Features
Drebin-215	15,036	5,560	9,467	215
AndroidHealthCheck	16,300	15,506	4,000	215
NATICUSdroid	29,332	14,700	14,632	86

were classified as malware, and 4,000 were labeled benign APK files. The third dataset, NATICUSdroid [42], primarily comprises 86 Android native and custom permission features extracted from 29,332 benign and malware Android apps released between 2010-2019. Within this dataset, 14,632 samples were identified as benign, while 14,700 were recognized as malicious. The diversity inherent in our dataset contributes to the robustness of our model, endowing it with the capability to generalize effectively.

B. DATASET PREPROCESSING

The initial datasets were provided in binary format, consisting solely of 0s and 1s. To facilitate analysis with DeepImageDroid, we considered the following comprehensive preprocessing pipeline:

- Data consolidation: The three datasets are merged into a single collection, categorizing each sample into one of two distinct classes: benign and malware. This unified structure simplifies model training and evaluation by providing a consistent labeling scheme.
- Automated feature extraction and image conversion: A custom module [43] is developed to extract relevant features from each data row. These features are then converted into grayscale image representations. This process transforms the numerical data into a visual format suitable for DeepImageDroid's image-based analysis.
- Normalization: To ensure consistent image intensity and prevent potential biases during training, the grayscale values are normalized to range 0-1 to help improve model convergence and performance.

Fig. 2 illustrates the preprocessing pipeline and emphasizes the key data transformation steps involved. After completing these preprocessing steps, 44,262 Android malware images are generated and categorized into benign and malware classes in our grey-scaled image dataset. Approximately 50% of the samples are benign, and slightly less than 50% are malware. This indicates a balanced dataset for both classes.

C. PROPOSED FRAMEWORK AND MODEL ARCHITECTURES

Utilizing the meticulously preprocessed datasets, we developed three distinct models to assess their performance on our novel grayscale representation. This marks the first attempt to integrate and use three diverse Android malware datasets in this fashion. Our objective was to evaluate the effectiveness of our grayscale data for both CNNs and ViTs before introducing the proposed hybrid model.

1) DeepImageDroid's CNN ARCHITECTURE

As shown in Fig. 3, the model starts with an Input Layer for grayscale images with dimensions of 64×64 pixels. The first Convolutional Layer uses 32 filters of size 3×3 to extract distinct feature maps. A ReLU activation function is applied after each convolution to introduce non-linearity and enhance the learning of complex relationships [32]. Following the convolution, a MaxPooling Layer downsamples the feature maps by taking the maximum value from each 2×2 block, reducing spatial dimensions while preserving key features to control computational costs and prevent overfitting.

A second convolutional layer with 64 filters of size 3×3 is applied, followed by a ReLU activation function for non-linearity, and another MaxPooling layer is used to reduce spatial dimensions further and enhance efficiency. To prevent overfitting and improve generalization, a Dropout Layer with a rate of 0.25 is introduced, randomly dropping neurons during training to ensure robust feature learning. The features are then flattened into a 1D vector for the fully connected layers, which handle high-level reasoning and classification tasks. The first Dense Layer with 128 neurons and ReLU activation learns complex patterns, followed by another Dropout Layer with a rate of 0.5 for additional regularization. A subsequent Dense Layer with 50 neurons and ReLU activation further refines the representations. The Output Layer, consisting of a single neuron with a sigmoid activation function, handles the binary classification task (malware vs. benign) by mapping the output to a probability between 0 and 1.

We utilized the binary cross-entropy loss function to optimize the model's performance, a standard choice for binary classification tasks. The Adam optimizer was selected for its efficiency in gradient descent and parameter updates during training. Accuracy was the primary metric to monitor the model's learning progress and assess its effectiveness in distinguishing between benign and malicious Android app samples. By adopting this CNN architecture, we aimed to leverage the robust feature extraction capabilities of CNNs to analyze the grayscale representations of our malware datasets.

2) DeepImageDroid's ViT ARCHITECTURE

Fig. 4 depicts the DeepImageDroid's ViT architecture, where the model begins with an input layer accepting grayscale images of a fixed size (64×64 pixels in this case). The key stages are outlined next.

a: PATCH CREATION AND POSITIONAL ENCODING

This initial stage of the ViT architecture involves patching. Here, the input image is divided into smaller, non-overlapping squares known as patches. The chosen patch size (e.g., 11×11 in this study) determines the granularity of the feature extraction process. Each patch is flattened and transformed into a vector using a projection layer. Since the

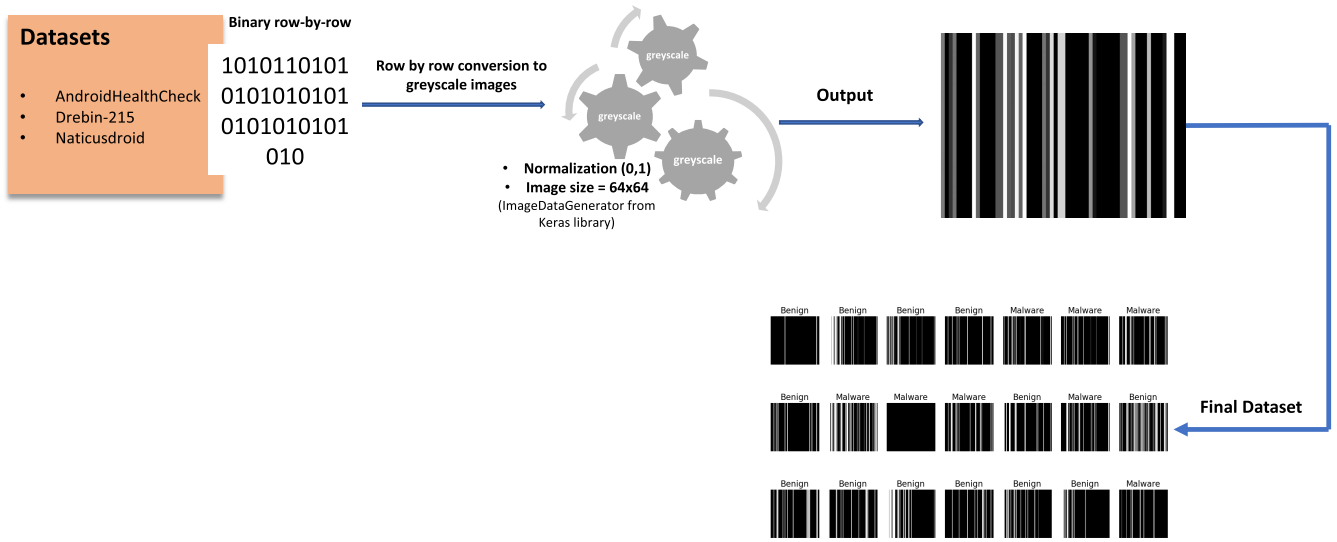

FIGURE 2. Datasets preprocessing steps.

order of patches within an image holds valuable information, the ViT incorporates positional encoding. This technique injects information about the relative position of each patch into its corresponding vector representation. This enables the model to understand the spatial relationships between different parts of the image and effectively capture global dependencies [39].

b: TRANSFORMER ENCODER

The heart of the ViT architecture lies in the transformer encoder, consisting of several stacked transformer blocks. Each block comprises two key components:

- Multi-head attention: This mechanism enables the model to learn relationships between different patches within the image, capturing long-range dependencies that CNNs might struggle with. It essentially allows the model to "attend" to specific regions of the image based on their relevance to the task at hand (malware detection in this case) [39].
- Multi-layer perceptron (MLP): This component further refines the encoded features by applying non-linear transformations through fully connected layers [39].

The transformer encoder in this ViT model utilizes 8 such transformer blocks, allowing for the progressive extraction of increasingly complex features from the input data.

c: CLASSIFICATION HEAD

Following the final transformer block, a classification head is employed to convert the learned feature representation into class probabilities. This head typically consists of a series of fully connected layers, culminating in a final output layer with a single neuron using a sigmoid activation function for binary classification tasks like malware detection. The sigmoid function maps the output value between 0 and 1, representing the probability of the input image belonging to the malware class. Furthermore, the ViT model is trained using the AdamW optimizer with a learning rate of 0.0005 and weight decay of 0.0001. The training process involves feeding the model batches of 64 grayscale images and their corresponding labels for 20 epochs. Early stopping with a patience of 15 epochs is employed to prevent overfitting by halting training if the validation loss fails to improve for a specified number of epochs.

D. DeepImageDroid's HYBRID MODEL: COMBINING STRENGTHS FOR ENHANCED MALWARE DETECTION

DeepImageDroid goes beyond leveraging the individual strengths of CNNs and ViTs. It takes a further step by implementing a hybrid model based on ensemble learning. This section delves into the specific architecture and functionalities of this ensemble approach. DeepImageDroid's core innovation lies in its hybrid model, which strategically combines the strengths of both CNNs and ViTs to achieve superior malware detection performance. This approach leverages the unique capabilities of each model to create a more robust and comprehensive feature extraction process.

1) HYBRID MODEL ARCHITECTURE

ViTs excels at capturing long-range dependencies, identifying subtle relationships in data that CNNs might miss due to their focus on local features. Conversely, CNNs are proficient at learning local features, which is crucial for detecting specific visual patterns in malware. Combining these strengths, a hybrid model can achieve higher accuracy by leveraging the diverse feature extraction capabilities of both ViTs and CNNs, leading to a comprehensive understanding of the data [30], [39]. Ensemble learning enhances this approach by combining predictions from multiple models, improving

Algorithm 1 DeepImageDroid: Hybrid CNN and ViT Model for Android Malware Detection

Input: Dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$ where x_i is the grayscale image and y_i is the label (0: benign, 1: malware)
Output: Trained hybrid model H
procedure TRAIN HYBRID MODEL
    // Preprocess the data
    $D_{train}, D_{test} \leftarrow$ split(D, train_test_ratio $= 0.8$)
    $X_{train}, Y_{train} \leftarrow$ extract(D_{train})
    $X_{test}, Y_{test} \leftarrow$ extract(D_{test})
    // Initialize CNN and ViT models
    $CNN \leftarrow$ initialize_CNN()
    $ViT \leftarrow$ initialize_ViT()
    // Train CNN model
    for each epoch in epochs **do**
        $CNN \leftarrow$ train($CNN, X_{train}, Y_{train}$)
    // Train ViT model
    for each epoch in epochs **do**
        $ViT \leftarrow$ train($ViT, X_{train}, Y_{train}$)
    // Make predictions on the test set
    $P_{CNN} \leftarrow$ predict(CNN, X_{test})
    $P_{ViT} \leftarrow$ predict(ViT, X_{test})
    // Combine predictions using weighted average
    $\alpha_{CNN} \leftarrow 0.9$
    $\alpha_{ViT} \leftarrow 0.5$
    for each i in 1 to $|X_{test}|$ **do**
        $P_{hybrid}[i] = \frac{\alpha_{CNN} \cdot P_{CNN}[i] + \alpha_{ViT} \cdot P_{ViT}[i]}{\alpha_{CNN} + \alpha_{ViT}}$
    return Trained hybrid model $H = (CNN, ViT, P_{hybrid})$
procedure ESTIMATE
    Input: $X = \{x_1, x_2, \ldots, x_V\}$, Trained hybrid model $H = (CNN, ViT, P_{hybrid})$
    // Generate embeddings and predictions
    $P_{CNN} \leftarrow$ predict(CNN, X)
    $P_{ViT} \leftarrow$ predict(ViT, X)
    // Combine predictions using weighted average
    $\alpha_{CNN} \leftarrow 0.9$
    $\alpha_{ViT} \leftarrow 0.5$
    for each v in 1 to $|X|$ **do**
        $P_{final}[v] = \frac{\alpha_{CNN} \cdot P_{CNN}[v] + \alpha_{ViT} \cdot P_{ViT}[v]}{\alpha_{CNN} + \alpha_{ViT}}$
    // Predict the final class
    $\hat{Y} \leftarrow$ argmax(P_{final})
    return Estimated $\hat{Y}$

performance, reducing variance, and resulting in more robust and generalizable outcomes [44], [45].

Building on these advantages, we propose a hybrid model consisting of two primary components: a CNN-based malware detection model and a ViT-based classification model. Figures 3 and 4 show that both models are pretrained on our robust dataset and fine-tuned to optimize performance for malware detection tasks. The pretrained models are loaded into memory and combined using a weighted ensemble approach. Predictions from each model are weighted based on their performance, with the weights calibrated to optimize classification accuracy. The weighted predictions are then combined to generate a final ensemble prediction.

Further contributing to our approach, weighted averaging is a powerful ensemble method that leverages the strengths of multiple learners by combining their predictions into a more robust output. This method assigns weights to each individual learner, reflecting its perceived reliability or accuracy within the ensemble. Consequently, predictions from learners deemed more reliable or accurate contribute more significantly to the final combined prediction, while those from less reliable learners have a diminished influence [44]. The weighted average prediction is expressed as:

$$\hat{y} \sum_{i=1}^{K} w_i \hat{y}_i \mathbf{w}^T \cdot \hat{\mathbf{y}} \tag{1}$$

where:

- $\hat{y}$ is the weighted average prediction
- K is the number of ensemble members
- w_i is the weight of the i-th ensemble member
- $\hat{y}_i$ is the prediction of the i-th ensemble member
- $\mathbf{w}$ is the weight vector
- $\hat{\mathbf{y}}$ is the vector of ensemble member predictions

2) OPERATIONAL WORKFLOW

The hybrid ensemble model follows a step-by-step approach to classify malware images, leveraging the strengths of the two individual models (CNN and ViTs) and combining their predictions for improved accuracy. As detailed in Algorithm 1, our proposed hybrid model leverages the strengths of both CNN and ViT to enhance the accuracy of Android malware detection. This algorithm outlines the steps for training the hybrid model and combining predictions using weighted averaging to achieve robust classification performance. Additionally, Fig. 5 illustrates the operational workflow of the proposed model, which consists of the following:

- Model loading: The workflow begins with loading two pre-trained models, CNN and ViT, both designed for Android malware detection. These models are retrieved from saved files and put into memory for subsequent use.
- Individual predictions: The loaded models are independently applied to the test dataset of unseen malware and benign images. Each model generates a prediction representing the probability of an image being malware.
- Weighted averaging: Predefined weights are assigned to each model's predictions based on their validation performance. The individual weighted predictions are then combined using a weighted averaging technique (np.tensordot), amplifying the influence of the better-performing model.
- Consensus prediction: The weighted predictions from both models are combined to form a single consensus prediction for each test sample, reflecting the ensemble's hybrid nature and leveraging both models' strengths.
- Thresholding and binary classification: The consensus predictions are subjected to a threshold (0.5), converting them into binary classification results. Predictions exceeding the threshold are classified as malware, and those below as benign.

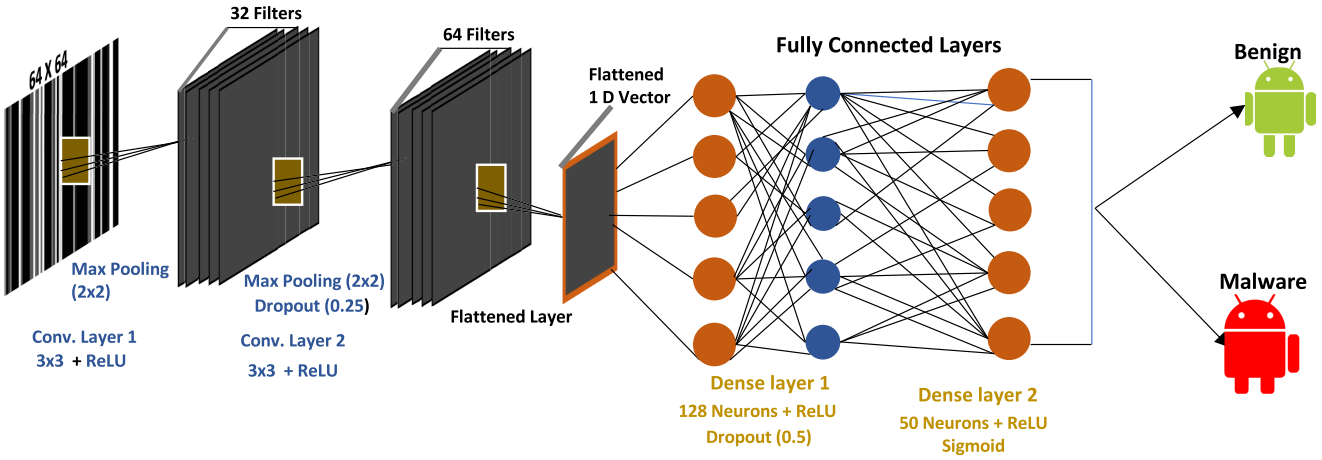

FIGURE 3. DeepImageDroid's CNN architecture.

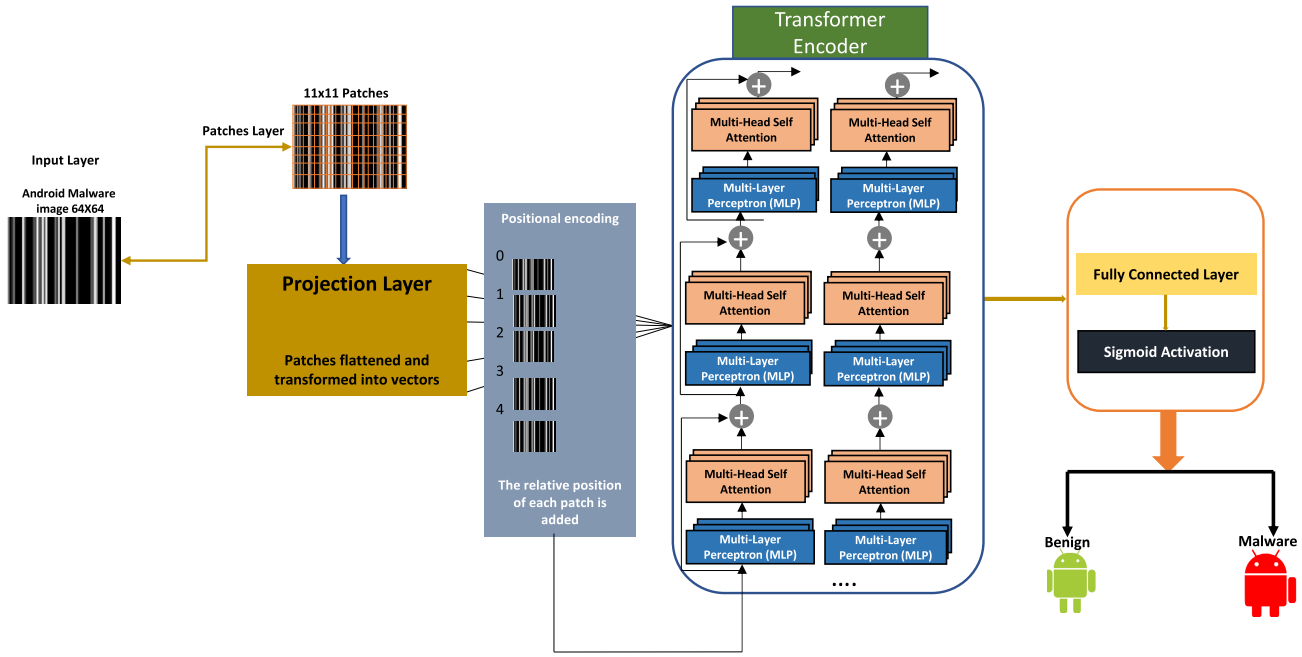

FIGURE 4. DeepImageDroid's ViT architecture.

- Performance evaluation: The final binary classifications are compared with the actual labels of the test data. Standard performance metrics are calculated to assess the effectiveness of the ensemble model in identifying malware and benign samples.

By integrating the complementary strengths of CNNs and ViTs, DeepImageDroid harnesses both models' comprehensive feature extraction capabilities. This hybrid approach captures global relationships and fine-grained details within malware images, resulting in a more accurate and robust data analysis. Consequently, this enhanced understanding significantly improves the model's performance in identifying malicious applications, surpassing the effectiveness of relying solely on either CNNs or ViTs.

V. EVALUATION

This section uses a robust grayscale dataset to evaluate the effectiveness of DeepImageDroid, our proposed hybrid model for Android malware detection. Dataset preprocessing details are provided in Subsection IV-B and Fig. 2. We discuss the evaluation metrics used to assess model performance and present the experimental results of the CNN, ViT, and DeepImageDroid models. Finally, we compare DeepImageDroid with the standalone models to demonstrate the enhanced efficiency of the hybrid approach.

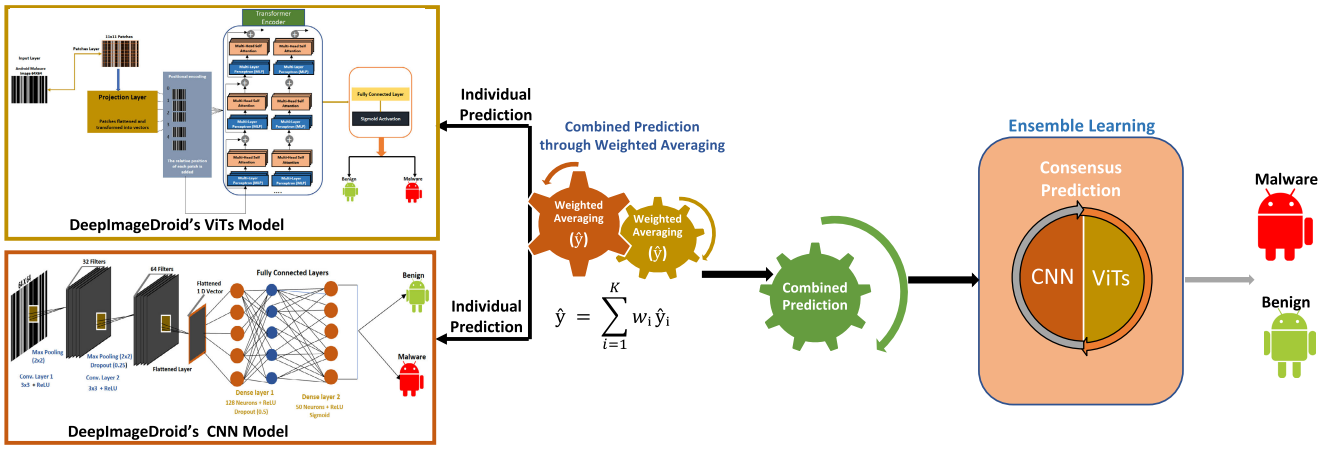

FIGURE 5. The proposed DeepImageDroid hybrid model.

A. EVALUATION METRICS

The following various metrics are used to assess the performance of our malware classification framework.

- Accuracy (ACC): The proportion of correctly classified images relative to the total number of samples, expresses as: $\frac{TP+TN}{TP+TN+FP+FN}$. TP, TN, FP, and FN represent true positives, true negatives, false positives, and false negatives, respectively.
- Precision (PRE): The ratio of accurately classified positive images to the total number of predicted positive images, expressed as: $\frac{TP}{TP+FP}$.
- Recall (REC): The proportion of correctly identified positive images relative to the total number of actual positive cases, expressed as: $\frac{TP}{TP+FN}$.
- F1-Score: A metric combining precision and recall, calculated as the harmonic mean of precision and recall, calculated as: $2 \cdot \frac{PRE \cdot REC}{PRE+REC}$.
- Confusion matrix: A visual representation of TP, FP, FN, and TN for both malware and benign classes. It helps identify the model's strengths and weaknesses and compute the false positive rate (FPR: $\frac{FP}{FP+TN}$), false negative rate (FNR: $\frac{FN}{FN+TP}$), and true positive rate (TPR: $\frac{TP}{TP+FN}$).

These metrics are all critical for evaluating the performance of any deep learning model. They were chosen because they comprehensively evaluate all the models in our study. However, since our study primarily concerns malware detection and the classification of a novel model, DeepImageDroid, in the context of Android malware detection and classification, we prioritize recall, precision, F1-Score FNR, and FPR, for the following reasons:

1) Recall: High recall is essential because the primary goal of malware detection is to identify as many malware instances as possible. Missing a piece of malware (false negative) can have severe consequences, including security breaches and data loss. Ensuring that the model detects nearly all instances of malware helps minimize the risk of undetected threats that could compromise the system.
2) Precision: Precision is important to reduce the number of false positives, which are benign instances incorrectly classified as malware. High precision ensures that it is likely to be correct when the model identifies malware. A high number of false positives can lead to alert fatigue, where users start ignoring warnings because too many benign files are flagged as malicious. This can undermine the effectiveness of the malware detection system.
3) F1-Score: The F1-score balances precision and recall, providing a single measure that considers both the ability to identify malware and the accuracy of positive predictions correctly. In malware detection, it is crucial to have a balanced view of the model's performance since both missing malware (low recall) and false alarms (low precision) are undesirable.
4) False Negative Rate (FNR): FNR is the complement of recall and directly indicates the proportion of malware that the model fails to detect. A low FNR is crucial to ensure the system is robust against threats and minimizes the risk of undetected malware.
5) False Positive Rate (FPR): FPR measures the rate at which benign instances are incorrectly classified as malware. While not as critical as recall or precision, maintaining a low FPR is still important to ensure the system remains user-friendly and efficient, avoiding unnecessary interruptions due to false alarms.

Our study demonstrates that the novel model, DeepImageDroid, performed well across all these validation metrics. By considering these metrics together, we can obtain a comprehensive assessment of DeepImageDroid's effectiveness in detecting Android malware, thereby providing a robust cybersecurity solution.

TABLE 3. Test set accuracy for ViT and CNN models on unseen data.

Model	Accuracy
ViT	0.9537
CNN	0.9611

B. TRAINING AND TESTING STRATEGIES

To ensure the effectiveness of our hybrid model, each constituent model within the ensemble needs to be robust and capable of accurate classification. In this context, we evaluate the performance of our CNN and ViT models in distinguishing between benign and malicious malware images. Our well-preprocessed grayscale dataset (split into training and testing sets) is utilized for the evaluation.

1) TRAINING PROCESS

The training data, comprising grayscale image features and corresponding labels (malware or benign), is fed to both the CNN and ViT models. These models undergo training for 20 epochs, iteratively learning to map the input features to the correct class labels. During training, the models process the data in batches, adjusting their internal parameters to minimize the difference between their predictions and the true labels.

2) TESTING THE MODELS: EVALUATION ON UNSEEN DATA

Once trained, the CNN and ViT models are evaluated on a separate, unseen test dataset to assess their generalizability and performance on unfamiliar data. Our model achieved 96.11% accuracy on the test set, as shown in Table 3. The models make predictions based on the test data, which are compared to the true labels to calculate various evaluation metrics, such as accuracy, precision, recall, and F1-score.

C. CNN RESULTS AND ANALYSIS

As highlighted in Table 4 and Fig. 6, our CNN model achieved excellent performance on the evaluation. Specifically, the model achieved an overall validation accuracy of 96.11%, indicating a high proportion of correctly classified samples in the test set. The training accuracy increased across epochs, reaching a final value of 96.72%, suggesting that the model correctly classifies nearly 97% of the training samples. The validation accuracy remained above 95% throughout the training process, demonstrating that the model generalizes well and avoids overfitting.

In terms of precision, the model achieved 95.65% for the benign class and 96.60% for the malware class, indicating high accuracy in identifying both benign and malicious samples with few false positives. The recall values were 96.80% for benign and 95.38% for malware, showing that the model misses very few actual malware or benign samples. The F1-Score, a harmonic mean of precision and recall, was 0.9622 for the benign class and 0.9599 for the malware class, providing a balanced view of the model's performance.

Last, FPR was 0.031, meaning only 3.1% of benign images were incorrectly classified as malware, minimizing unnecessary alarms and reducing the burden on human analysts. FNR was 0.046, indicating that the model misses 4.6% of actual malware images. Despite this, the model's TPR of 0.9538 demonstrates its effectiveness, correctly identifying 95.38% of actual malware images.

TABLE 4. Classification report for CNN model.

		Precision	Recall	F1-score	Support
Benign		0.9565	0.9680	0.9622	6799
Malware		0.9660	0.9538	0.9599	6480
Accuracy	0.9611				
macro avg		0.9613	0.9609	0.9611	13279
weighted avg		0.9612	0.9611	0.9611	13279
FPR	0.031%				
FNR	0.046%				
TPR	0.95.38%				

These results demonstrate the robustness and efficacy of our CNN model in differentiating between benign and malicious malware images. This individual solid model performance contributes significantly to the overall effectiveness of the hybrid DeepImageDroid model.

D. ViTs RESULTS AND ANALYSIS

As evidenced by the compelling results presented in Table 5 and Fig. 7, our ViT model also demonstrates remarkable performance in the task of Android malware classification. This achievement underscores the promising potential of ViTs for this critical domain.

As evidenced by the compelling results presented in Table 5 and Fig. 7, our ViT model also demonstrates remarkable performance in the task of Android malware classification. Specifically, for the "Benign" class, the precision is 0.9658, indicating that 96.58% of instances predicted as benign are actually benign. The recall for this class is 0.9482, meaning the model correctly identifies 94.82% of all actual benign instances. For the "Malware" class, the precision is 0.9466, indicating that 94.66% of instances predicted as malware are indeed malware. The recall for this class is 0.9648, meaning the model correctly identifies 96.48% of all actual malware instances. It is worth noting that the ViT model exhibits more fluctuations in validation performance compared to the CNN, as observed in the validation loss trends. This can be attributed to ViT's reliance on self-attention mechanisms for capturing global dependencies, which increases its sensitivity to variations in the dataset, leading to greater instability during training. In contrast, CNN's localized feature extraction contributes to a more stable validation trend.

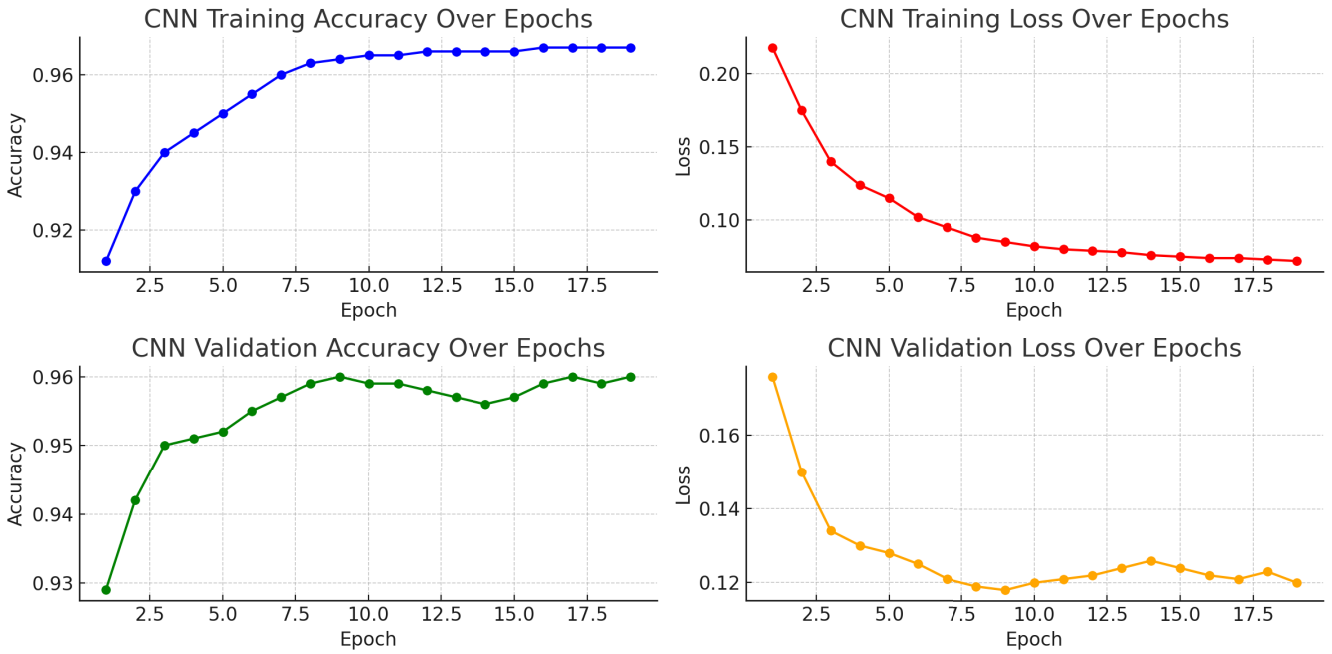

FIGURE 6. Analysis of our CNN model training session: The training accuracy starts at 0.9112 and steadily increases across epochs, reaching a final value of 96.72%. This suggests the model is correctly classifying nearly 97% of the training samples. The validation accuracy gradually moves above 95% during the training process, indicating the model is generalizing well and avoiding overfitting the training data. The training loss steadily decreases across epochs, indicating the model is effectively learning to minimize its error during training. This is a positive sign, suggesting the model improves its ability to distinguish between benign and malicious samples.

Additionally, the F1-scores, which balance precision and recall, are around 0.95 for both classes, indicating good overall performance. FPR is 0.051, meaning 5.1% of benign instances are incorrectly classified as malware, indicating a low potential for false alarms. TPR is 0.9648, showing the model successfully identifies 96.48% of malware images. FNR is 3.5%, indicating a slight chance of overlooking some malware.

Last, The overall validation accuracy of our ViT model reaches 0.9563, correctly predicting approximately 95.7% of instances, which suggests good generalization and avoidance of overfitting. The training accuracy increases steadily throughout the epochs, reaching a final value of 96.27%, indicating effective learning and differentiation between the two image classes during training.

Overall, the ViTs model demonstrates strong performance in distinguishing between benign and malware instances, as indicated by high precision, recall, and F1-score values, along with a relatively low false positive rate and FNR. This contributes significantly to the overall effectiveness of the hybrid DeepImageDroid model.

TABLE 5. Classification and confusion matrix report for ViT model.

		Precision	Recall	F1-score	Support
Benign		0.9658	0.9482	0.9569	6799
Malware		0.9466	0.9648	0.9556	6480
Accuracy	0.9563				
macro avg		0.9562	0.9565	0.9563	13279
weighted avg		0.9565	0.9563	0.9563	13279
FPR	0.051%				
FNR	0.035%				
TPR	0.96.48%				

E. DeepImageDroid MODEL RESULTS AND ANALYSIS

By combining the unique capabilities of our robust ViTs and CNN models as described in Fig. 5, our hybrid model, DeepImageDroid, employs a collaborative approach, combining the predictions of two diverse models and leveraging their individual strengths to achieve potentially improved malware classification performance as seen in Table 6. The introduced weighted average for our ensemble DeepImageDroid model is simplified as follows:

$$\hat{y} = \sum_{i=1}^{K} w_i \hat{y}_i, \tag{2}$$

where $\hat{y}$, $\hat{y}_i$, and $\mathbf{w}$ represent the final ensemble prediction, predictions of the CNN and ViTs models, and the vector of weights, respectively.

This can be equivalently expressed in the form of a dot product as $\hat{y} = \mathbf{w}^T \cdot \hat{\mathbf{y}}$, where $\hat{y}$, $\hat{\mathbf{y}}$, and $\mathbf{w}$ now denote the final prediction of the ensemble model, a vector containing the

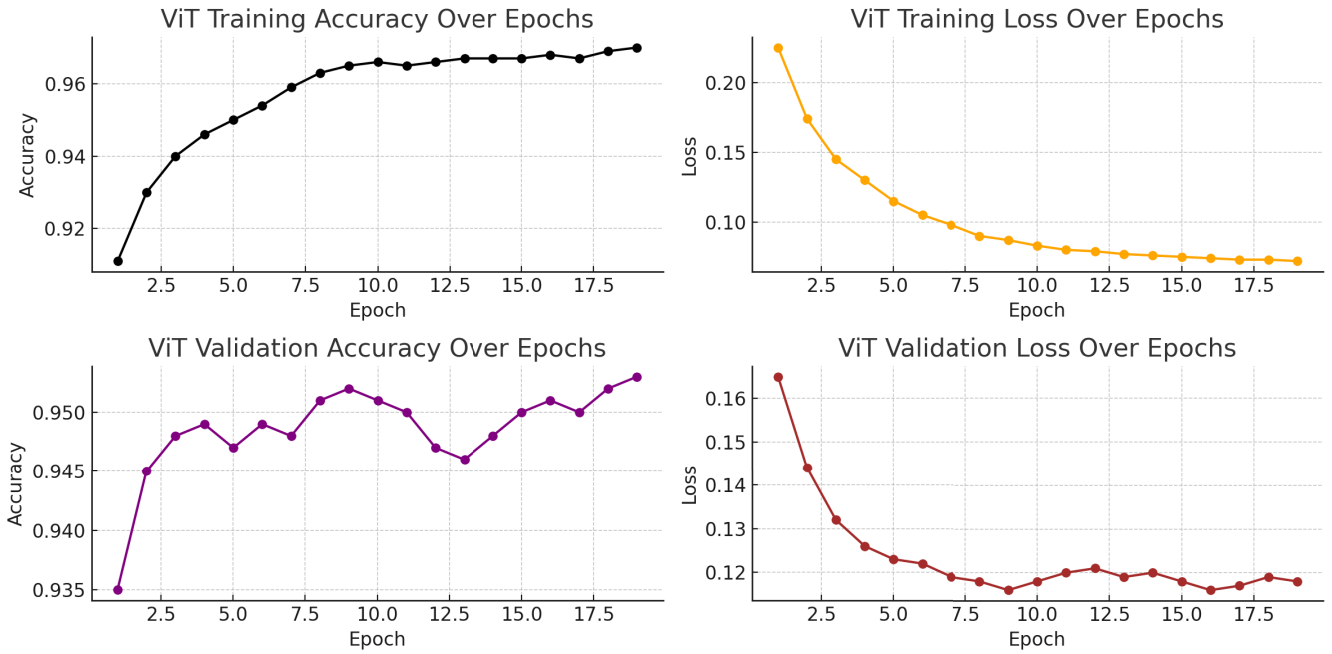

FIGURE 7. Analysis of our ViTs model training session: The training accuracy steadily increases throughout the epochs, reaching a final value of nearly 96%. This indicates that the model effectively learns to distinguish between the two image classes during training. The validation accuracy also increases throughout the training process, reaching a final value of over 95%. This suggests the model generalizes well and doesn't simply memorize the training data. The training loss follows a decreasing trend, reaching a final value below 0.10. This signifies that the model progressively minimizes its errors during training. The validation loss also shows a decreasing trend, though with some fluctuations, which is expected during the validation process.

predictions of the individual models, and a vector containing the weights assigned to the CNN (0.9) and ViT (0.5) model, respectively. We assigned a higher weight to the model that performed well on the validation data. $\mathbf{w}^T$ denotes the transpose of the weight vector. Last, the dot . product signifies the weighted sum.

The obtained results from the DeepImageDroid ensemble hybrid model for malware classification demonstrate promising performance, suggesting a successful collaboration between the CNN and ViT models. Specifically, the model shows a high overall accuracy of 0.96, correctly classifying nearly 96% of malware and benign images in the test set, showcasing the effectiveness of the ensemble approach. It achieves balanced class performance, with precision and recall values above 0.96 for benign and malware classes, indicating no significant bias. The model maintains a low FPR of 0.042%, minimizing misclassification of benign images as malware, and a low FNR of 0.037%, indicating few missed malware images. Additionally, the high TPR of 0.9628% shows that DeepImageDroid accurately identifies over 96% of actual malware images.

TABLE 6. Classification report and confusion matrix for DeepImageDroid.

		Precision	Recall	F1-score	Support
Benign		0.96	0.96	0.96	6799
Malware		0.96	0.96	0.96	6480
Accuracy	0.96				
macro avg		0.96	0.96	0.96	13279
weighted avg		0.96	0.96	0.96	13279
FPR	0.042%				
FNR	0.037%				
TPR	0.96%				

F. DISCUSSIONS

1) COMPARATIVE PERFORMANCE ANALYSIS OF RESULTS

The results of our study demonstrate the effectiveness of CNN, ViT, and the hybrid DeepImageDroid model in detecting Android malware. Each model exhibits vital performance metrics, including precision, recall, F1-score, FPR, TPR, and FNR. However, the hybrid model achieves a performance comparable to each model's. Table 7 presents the results of all models.

As shown in Table 7, the CNN model achieved the highest overall accuracy (0.9611) and F1-score (0.9611) among the individual models. This aligns with the strengths of CNNs in image classification. The ViT and DeepImageDroid models displayed competitive performance, highlighting the potential of transformer-based architectures and ensemble approaches. Additionally, the ViT model achieves an accuracy of approximately 95%, with an FPR of 5.1%, TPR of 96.48%, and FNR of 3.5%. The CNN model achieves

an accuracy of around 96.11%, with a FPR of 3.1%, TPR of 95.38%, and FNR of 4.6%. Both models demonstrate balanced precision and recall.

The DeepImageDroid hybrid model leverages the strengths of both CNNs and ViTs. While performing slightly lower than the individual CNN model in accuracy and F1-score (0.96 vs. 0.9611), it exhibits a well-balanced performance across all metrics, with high accuracy (0.96), precision (0.96), and recall (0.96). The hybrid model maintains a low FPR (0.042) and FNR (0.037), indicating a good balance between minimizing false alarms and correctly identifying true positives. These results suggest that the DeepImageDroid hybrid model offers competitive performance and effectively combines the strengths of CNNs and ViTs for malware classification.

The preceding comparative performance analysis of the CNN, ViT, and the hybrid model (DeepImageDroid) for Android malware detection provides valuable insights into their effectiveness and potential areas for improvement. However, the hybrid model, DeepImageDroid, demonstrates a balanced performance by leveraging the strengths of both CNN and ViT models. While it does not outperform the CNN model in terms of accuracy and precision, it achieves a better balance between TPR and FPR, reducing the chances of both false positives and false negatives.

- **FPR**: DeepImageDroid has a moderate FPR, lower than ViT but higher than CNN. This balance helps in reducing unnecessary alerts while maintaining a good detection rate.
- **TPR**: The hybrid model's TPR is closer to the ViT model, ensuring a high detection rate for malware instances.
- **FNR**: With an FNR of 0.037, DeepImageDroid misses fewer malware instances than the CNN model, indicating improved reliability in malware detection.

The hybrid model DeepImageDroid provides a well-rounded performance in Android malware detection, balancing the strengths of CNN and ViT models. While the CNN model achieves the highest accuracy and lowest FPR, the ViT model excels in TPR and FNR. The hybrid approach effectively integrates these strengths, resulting in a reliable and robust malware detection system.

2) COMPARATIVE FEATURE EXTRACTION CAPABILITIES OF CNNs AND ViTs

In this study, CNNs and ViTs demonstrated distinct strengths in feature extraction capabilities, which contributed to the overall robustness of the proposed DeepImageDroid framework. CNNs are particularly effective at capturing local features within images, thanks to their use of convolutional layers and pooling operations. These layers allow CNNs to detect fine-grained patterns and details, such as edges and textures, by focusing on small receptive fields and progressively building more complex feature hierarchies. This capability is crucial for identifying specific characteristics of malware, such as unusual patterns or structures in binary images that may indicate malicious activity.

On the other hand, ViTs excels at capturing global context and long-range dependencies within the data. Unlike CNNs, which rely on localized convolutional filters, ViTs utilize a self-attention mechanism that can simultaneously consider relationships between all parts of an image. This enables ViTs to understand the broader context and more complex relationships within the data, which is particularly useful for detecting subtle and sophisticated malware behaviors that might span across different parts of an image or involve complex interactions. The ability of ViTs to integrate information from the entire image allows them to capture global patterns that might be overlooked by CNNs, which are more focused on local features.

While CNNs can extract detailed local features, ViTs provide a complementary strength by capturing global dependencies and contextual information. Integrating these two models in the DeepImageDroid framework leverages their strengths, resulting in a more comprehensive and practical approach to Android malware detection. This combination enhances the model's ability to detect a wide range of malware characteristics, from specific patterns to broader contextual cues, thereby improving detection accuracy and robustness.

3) COMPARATIVE ANALYSIS OF DeepImageDroid WITH STATE-OF-THE-ART MODELS

To evaluate the effectiveness and robustness of the proposed DeepImageDroid model, we compare its performance against state-of-the-art models for Android malware detection. These models, selected from the most recent literature, employ various deep learning techniques, including CNNs, Vision Transformers, and self-supervised learning methods, each offering unique strengths and approaches to malware detection. By analyzing key metrics such as accuracy, precision, recall, F1-score, and False Positive Rate (FPR), we aim to highlight the advantages and limitations of DeepImageDroid relative to these models. The following comparative analysis focuses on understanding where DeepImageDroid excels and how its hybrid architecture balances the trade-offs between performance criteria.

- **DeepImageDroid vs. AdMat**: AdMat achieves a higher accuracy (98.26%) compared to DeepImageDroid (96.00%) and also outperforms DeepImageDroid in precision (97.8%), recall (96.9%), and F1-score (97.3%). In contrast, DeepImageDroid offers consistent performance across all key metrics (96.00% for accuracy, precision, recall, and F1-score) and demonstrates a lower False Positive Rate (FPR) of 0.042, which is beneficial in reducing false alarms—a critical factor in practical malware detection settings. This highlights DeepImageDroid's focus on minimizing false positives, making it suitable for environments that require high reliability, such as mobile and edge computing.

TABLE 7. A comparative analysis of the models.

Model	Accuracy	Precision	Recall	F1-Score	FPR	TPR	FNR
CNN	0.9611	0.9612	0.9611	0.9611	0.031	0.9538	0.046
ViT	0.9563	0.9565	0.9563	0.9563	0.051	0.9648	0.035
DeepImageDroid	0.96	0.96	0.96	0.96	0.042	0.9628	0.037

TABLE 8. Performance comparison of DeepImageDroid with state-of-the-art models.

Model	Accuracy	Precision	Recall	F1-Score	FPR
AdMat [10]	98.26%	97.8	96.9	97.3	-
ViT4Mal [18]	97%	-	-	-	-
Self-Supervised Vision Transformers [4]	97%	87%	89%	87.8%	-
MalSSL [46]	96.2%	-	-	-	0.6
DeepImageDroid	96%	96.5%	96.48%	95%	0.042

- **DeepImageDroid vs. ViT4Mal**: ViT4Mal achieves a higher accuracy (97%) compared to DeepImageDroid's accuracy (96.00%). However, ViT4Mal's approach involves converting executable byte-code into images, adding computational overhead that may limit scalability. DeepImageDroid's hybrid model effectively combines CNN and ViT, balancing local and global feature extraction. Although DeepImageDroid shows slightly lower accuracy, its hybrid approach allows for a broader generalization across diverse malware patterns, making it a more versatile choice for comprehensive malware detection.
- **DeepImageDroid vs. Self-Supervised Vision Transformers (SSViT)**: SSViT achieves a marginally higher accuracy (97%) but falls short in precision (87%), recall (89%), and F1-score (87.8%), compared to DeepImageDroid's precision, recall, and F1-score of 96%. Additionally, DeepImageDroid has a lower FPR (0.042) than SSViT, suggesting fewer false alarms. This efficiency, coupled with DeepImageDroid's simpler architecture, makes it a more practical choice for environments with limited computational resources.
- **DeepImageDroid vs. MalSSL**: MalSSL achieves a slightly higher accuracy (96.2%) but lacks comprehensive evaluation metrics such as precision, recall, and F1-score, limiting direct comparison. DeepImageDroid, however, demonstrates consistent performance across all metrics (96%) and has a lower FPR (0.042), underscoring its strength in minimizing false alarms. Integrating CNN and ViT models, this balanced approach makes DeepImageDroid highly adaptable to different malware patterns, providing a robust Android detection solution.

DeepImageDroid stands out for its hybrid model architecture that leverages the strengths of both CNNs and ViTs, providing a comprehensive feature extraction mechanism capable of handling complex malware detection tasks. The model's balanced performance across all evaluation metrics makes it highly reliable for varied and unpredictable malware landscapes. Moreover, its lower computational overhead compared to models that rely on more resource-intensive methods like self-supervised learning makes it suitable for real-world applications, including mobile and edge devices.

4) INTEGRATION WITH REAL-TIME MONITORING SYSTEMS FOR DYNAMIC MALWARE DETECTION

Integrating the framework with real-time monitoring systems is crucial to enhance the practical applicability and effectiveness of DeepImageDroid. Such systems can provide continuous surveillance and dynamic detection capabilities, enabling timely responses to emerging threats. Real-time monitoring systems typically operate by continuously analyzing network traffic, system logs, and other data sources to identify suspicious activities. Integrating DeepImageDroid into these systems can provide the following advantages.

Immediate Threat Detection: By continuously monitoring devices and network activities, real-time systems can instantly utilize DeepImageDroid's capabilities to detect anomalies and potential malware threats. This is especially vital in scenarios requiring quick action to mitigate security risks.

Dynamic Adaptation to New Threats: The evolving nature of malware, characterized by frequent updates and new variants, necessitates an adaptive approach to detection. Real-time systems equipped with DeepImageDroid can be updated with new detection models as they become available, ensuring the system remains robust against new threats.

Resource Optimization: Real-time monitoring systems can prioritize resources by focusing more intensive scanning efforts on devices or network segments flagged by DeepImageDroid as potentially compromised. This selective approach helps optimize computational resources and reduces the overall burden on the system.

Enhanced Reporting and Alerting: The integration allows for the generation of detailed reports and alerts based on DeepImageDroid's findings. These reports can include insights into the types of malware detected, potential entry points, and recommended mitigation strategies. Alerts can be

configured to notify administrators or automated systems for immediate action.

Cross-Platform Protection: Given that Android devices often interact with other platforms and network infrastructures, real-time monitoring systems can provide cross-platform protection by correlating data from multiple sources. DeepImageDroid can be a crucial component in this broader security ecosystem, enhancing overall system resilience.

By integrating DeepImageDroid with real-time monitoring systems, organizations can significantly bolster their security posture, providing robust and adaptive protection against the ever-evolving landscape of Android malware.

5) MODEL INTERPRETABILITY

In security-sensitive environments, understanding the decision-making processes of ML models is crucial for building trust and ensuring responsible deployment. While the current implementation of DeepImageDroid prioritizes achieving high accuracy and robust malware detection, the interpretability of the hybrid model remains an area that requires further development. Enhancing interpretability is essential for fostering greater trust and ease of adoption, particularly in contexts where understanding model decisions is critical.

To address this, future work will focus on integrating model explainability techniques such as Layer-wise Relevance Propagation (LRP) [47], SHapley Additive exPlanations (SHAP) [48], and Local Interpretable Model-agnostic Explanations (LIME) [49]. These techniques can provide insights into which features or parts of the input data significantly influence the model's decisions. By doing so, security analysts and stakeholders can better understand the reasoning behind classifications, such as why a particular sample was labeled as benign or malicious.

6) ABLATION STUDY FOR DEEPIMAGEDROID

The ablation study for DeepImageDroid was conducted to provide a detailed analysis of the contributions of each component- CNN, ViT, and their hybrid combination- to the model's overall performance. This analysis helps to understand which components are most critical for achieving high accuracy in malware detection and how different settings affect their effectiveness.

a: CNN COMPONENT ANALYSIS

The CNN component of DeepImageDroid, as depicted in Fig.3, is designed to learn local patterns in grayscale images. The CNN architecture includes:

- **Input Layer:** Grayscale images with dimensions 64×64 pixels.
- **Convolutional Layers:** Two convolutional layers with ReLU activation functions, where the first layer uses 32 filters and the second layer uses 64 filters, both with a kernel size of 3×3.
- **Pooling Layers:** Max pooling layers after each convolutional layer with a pool size of 2×2 to reduce spatial dimensions.
- **Dropout Layers:** Dropout layers with rates of 0.25 and 0.5 for regularization, preventing overfitting by randomly setting a fraction of input units to zero at each update during training.
- **Dense Layers:** Two dense (fully connected) layers with 128 and 50 neurons, respectively, followed by ReLU activation.
- **Output Layer:** A single neuron with a sigmoid activation function for binary classification.

Hyperparameters: The CNN was trained with a learning rate of 0.001, batch size of 32, and trained for 20 epochs. The optimizer used was Adam, and the loss function was binary cross-entropy. The dropout rates and the number of filters were tuned based on validation accuracy to avoid overfitting and improve generalization.

Impact on performance: The CNN model achieved an accuracy of 96.11% with high precision (95.65%) and recall (96.80%) for the benign class. The CNN was particularly effective in capturing local dependencies, contributing to a lower false positive rate (FPR) of 0.031.

b: ViT COMPONENT ANALYSIS

The ViT component, as illustrated in Fig. 4, focuses on capturing long-range dependencies by processing the image as a sequence of patches. Key components include:

- **Patch Creation and Embedding:** The input image is divided into non-overlapping patches of size 16×16. Each patch is linearly embedded into a vector.
- **Transformer Encoder:** Comprises 8 transformer layers, each with multi-head self-attention (with 8 attention heads) and a feed-forward network.
- **Positional Encoding:** Adds positional information to each patch embedding to retain spatial structure.
- **Classification Head:** An MLP head with fully connected layers leads to a sigmoid-activated output for binary classification.

Hyperparameters: The ViT model was trained with a learning rate of 0.0005, batch size of 64, and 20 epochs. The weight decay was set to 0.0001, and the AdamW optimizer was used. These hyperparameters were chosen through a grid search to balance underfitting and overfitting while maximizing the validation accuracy.

Impact on performance: The ViT model achieved an accuracy of 95.63%, with a higher True Positive Rate (TPR) of 0.9648 compared to the CNN, which shows its strength in capturing global dependencies. However, the ViT had a higher False Positive Rate (FPR) of 0.051, indicating more false alarms than the CNN.

c: HYBRID MODEL COMPONENT ANALYSIS

The hybrid model integrates the strengths of both CNNs and ViTs through a weighted ensemble approach, combining their

predictions to form a consensus as shown in Fig. 5. The ensemble strategy involves:

- **Weighted Averaging:** Predictions from the CNN and ViT models are weighted based on their validation performance. The weights assigned were 0.6 for CNN and 0.4 for ViT.
- **Final Prediction:** The final output is determined by combining the weighted outputs of both models, aiming to leverage the local feature extraction power of CNNs and the global dependency modeling of ViTs.

Hyperparameters: The weights for combining the predictions were determined based on cross-validation results, and the weighted averaging was implemented using the numpy library.

d: PERFORMANCE COMPARISON OF THE DIFFERENT MODELS

Table 7 demonstrates the performance of the individual CNN and ViT models and their combination in DeepImageDroid. The CNN achieved an accuracy of 96.11% with a low FPR of 0.031 but had a TPR of 0.9538, showing room for improvement in sensitivity. The ViT had a slightly lower accuracy of 95.63% but a higher TPR of 0.9648 and a lower FNR of 0.035, indicating better sensitivity despite a higher FPR of 0.051. Combining CNN and ViT predictions without weighted averaging resulted in an accuracy of 95.24%, benefiting from their complementary strengths. However, the weighted hybrid model outperformed all configurations with an accuracy of 96.00%, precision, recall, and F1-score of 96.00%, a TPR of 0.9628, FNR of 0.037, and FPR of 0.042, effectively leveraging the strengths of both models.

Impact on Performance: The hybrid model yielded the best performance, with an accuracy of 96.00%, precision, recall, and F1-score at 96.00%. It achieved a balanced FPR of 0.042 and FNR of 0.037, indicating enhanced sensitivity and specificity. As demonstrated in Table 7, the hybrid model effectively combines the strengths of CNN and ViT, resulting in a more robust malware detection system.

The ablation study demonstrates the critical role each component plays in DeepImageDroid. The CNN effectively captures local features with a low FPR, while the ViT excels in capturing global dependencies with a high TPR. The hybrid model's weighted combination of CNN and ViT predictions leverages these strengths, leading to a more robust and accurate model for Android malware detection. The choice of hyperparameters for each model was crucial in optimizing their performance and ensuring the hybrid model's superior results.

7) COMPUTATIONAL EFFICIENCY OF DeepImageDroid

DeepImageDroid demonstrates computational advantages due to its hybrid architecture, which effectively balances using CNNs and ViTs. The CNN component captures local features with high efficiency, utilizes fewer parameters, and requires lower computational power than transformer-based architectures. The ViT component is then used to capture global dependencies, which enhances the model's ability to generalize to diverse malware patterns. By leveraging each component for its strength, the hybrid model avoids redundant and resource-intensive operations if a single, unified architecture (such as a standalone ViT) is used for local and global feature extraction.

The use of a weighted average ensemble method further contributes to computational efficiency. Instead of training a larger monolithic model, DeepImageDroid combines the predictions of the CNN and ViT through a weighted ensemble approach. This reduces the need for exhaustive training and optimizes resource usage during the training and inference phases. The weighted fusion strategy focuses on maximizing the predictive power of both models without incurring the additional computational cost of a fully unified deep model.

Additionally, DeepImageDroid's modular hybrid approach results in a lower overall parameter count compared to standalone transformer models, which often contain a large number of parameters to achieve similar performance. This makes DeepImageDroid more suitable for real-time applications and resource-constrained environments such as mobile or edge devices, where maintaining high performance with limited computational resources is critical.

VI. CONCLUSION

This paper proposes a novel Android malware detection and classification framework using a hybrid ensemble weighted averaging approach. Our approach has demonstrated the effectiveness of individual CNN and ViT models and the hybrid DeepImageDroid model in Android malware detection. These models exhibit high precision, recall, F1-scores, robustness, and generalizability, highlighting their potential for real-world mobile security applications. Further, the ViT and CNN models each showcase robust performance in different aspects of malware detection. The hybrid model, combining both strengths, achieves high accuracy and a balanced FPR and TPR, demonstrating the benefits of an ensemble approach. Future improvements could include feature fusion techniques, hyperparameter tuning, ensemble diversity, and data augmentation.

Our study contributes to the field of Android malware detection and DL. The hybrid DeepImageDroid model represents a promising avenue for future research and practical implementation in securing mobile devices against evolving malware threats. We plan to explore more sophisticated ensemble methods like stacking or boosting and extend the model's application to other cybersecurity areas, such as IoT security, network security, and intrusion detection systems, ultimately enhancing the security of critical systems and protecting users from evolving cyber threats.

REFERENCES

[1] Y. Wang, J. Zheng, C. Sun, and S. Mukkamala, "Quantitative security risk assessment of Android permissions and applications," in *Proc. 27th Annu. IFIP WG 11.3 Conf.*, Newark, NJ, USA. Springer, Jul. 2013, pp. 226–241.

[2] A. Mathur, L. M. Podila, K. Kulkarni, Q. Niyaz, and A. Y. Javaid, "NATICUSdroid: A malware detection framework for Android using native and custom permissions," *J. Inf. Secur. Appl.*, vol. 58, May 2021, Art. no. 102696.

[3] M. A. Omer, S. R. M. Zeebaree, M. A. M. Sadeeq, B. W. Salim, S. X. Mohsin, Z. N. Rashid, and L. M. Haji, "Efficiency of malware detection in Android system: A survey," *Asian J. Res. Comput. Sci.*, vol. 7, no. 4, pp. 59–69, Apr. 2021.

[4] S. Seneviratne, R. Shariffdeen, S. Rasnayaka, and N. Kasthuriarachchi, "Self-supervised vision transformers for malware detection," *IEEE Access*, vol. 10, pp. 103121–103135, 2022.

[5] X. Deng and J. Mirkovic, "Polymorphic malware behavior through network trace analysis," in *Proc. 14th Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2022, pp. 138–146.

[6] Y. Liu, C. Tantithamthavorn, L. Li, and Y. Liu, "Deep learning for Android malware defenses: A systematic literature review," *ACM Comput. Surveys*, vol. 55, no. 8, pp. 1–36, Aug. 2023.

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[8] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5987–5995.

[9] M. Kinkead, S. Millar, N. McLaughlin, and P. O'Kane, "Towards explainable CNNs for Android malware detection," *Proc. Comput. Sci.*, vol. 184, pp. 959–965, Jan. 2021.

[10] L. N. Vu and S. Jung, "AdMat: A CNN-on-matrix approach to Android malware detection and classification," *IEEE Access*, vol. 9, pp. 39680–39694, 2021.

[11] A. Nicheporuk, O. Savenko, A. Nicheporuk, and Y. Nicheporuk, "An Android malware detection method based on CNN mixed-data model," in *Proc. ICTERI Workshops*, 2020, pp. 198–213.

[12] M. Ganesh, P. Pednekar, P. Prabhuswamy, D. S. Nair, Y. Park, and H. Jeon, "CNN-based Android malware detection," in *Proc. Int. Conf. Softw. Secur. Assurance (ICSSA)*, Jul. 2017, pp. 60–65.

[13] X. Xiao and S. Yang, "An image-inspired and CNN-based Android malware detection approach," in *Proc. 34th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2019, pp. 1259–1261.

[14] Y. Zhang, Y. Yang, and X. Wang, "A novel Android malware detection approach based on convolutional neural network," in *Proc. 2nd Int. Conf. Cryptography, Secur. Privacy*, Mar. 2018, pp. 144–149.

[15] C. Hasegawa and H. Iyatomi, "One-dimensional convolutional neural networks for Android malware detection," in *Proc. IEEE 14th Int. Colloq. Signal Process. Its Appl. (CSPA)*, Mar. 2018, pp. 99–102.

[16] A. Rahali and M. A. Akhloufi, "MalBERT: Malware detection using bidirectional encoder representations from transformers," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2021, pp. 3226–3231.

[17] J. Jo, J. Cho, and J. Moon, "A malware detection and extraction method for the related information using the ViT attention mechanism on Android operating system," *Appl. Sci.*, vol. 13, no. 11, p. 6839, Jun. 2023.

[18] A. Ravi, V. Chaturvedi, and M. Shafique, "ViT4Mal: Lightweight vision transformer for malware detection on edge devices," *ACM Trans. Embedded Comput. Syst.*, vol. 22, no. 5, pp. 1–26, Oct. 2023.

[19] W. Wang, M. Zhao, and J. Wang, "Effective Android malware detection with a hybrid model based on deep autoencoder and convolutional neural network," *J. Ambient Intell. Humanized Comput.*, vol. 10, no. 8, pp. 3035–3043, Aug. 2019.

[20] X. Pei, L. Yu, and S. Tian, "AMalNet: A deep learning framework based on graph convolutional networks for malware detection," *Comput. Secur.*, vol. 93, Jun. 2020, Art. no. 101792.

[21] Y. Lin and X. Chang, "Towards interpretable ensemble learning for image-based malware detection," 2021, *arXiv:2101.04889*.

[22] F. Pierazzi, G. Mezzour, Q. Han, M. Colajanni, and V. S. Subrahmanian, "A data-driven characterization of modern Android spyware," *ACM Trans. Manage. Inf. Syst.*, vol. 11, no. 1, pp. 1–38, Mar. 2020.

[23] S. Li, Y. Cao, S. Liu, Y. Lai, Y. Zhu, and N. Ahmad, "HDA-IDS: A hybrid DoS attacks intrusion detection system for IoT by using semi-supervised CL-GAN," *Exp. Syst. Appl.*, vol. 238, Mar. 2024, Art. no. 122198.

[24] A. Binbusayyis, "Hybrid VGG19 and 2D-CNN for intrusion detection in the FOG-cloud environment," *Exp. Syst. Appl.*, vol. 238, Mar. 2024, Art. no. 121758.

[25] F. Jemili, R. Meddeb, and O. Korbaa, "Intrusion detection based on ensemble learning for big data classification," *Cluster Comput.*, vol. 27, no. 3, pp. 3771–3798, Jun. 2024.

[26] P. Horchulhack, E. K. Viegas, A. O. Santin, and J. A. Simioni, "Network-based intrusion detection through image-based CNN and transfer learning," in *Proc. Int. Wireless Commun. Mobile Comput. (IWCMC)*, May 2024, pp. 386–391.

[27] W. Dai, X. Li, W. Ji, and S. He, "Network intrusion detection method based on CNN, BiLSTM, and attention mechanism," *IEEE Access*, vol. 12, pp. 53099–53111, 2024.

[28] Q. Wu, X. Zhu, and B. Liu, "A survey of Android malware static detection technology based on machine learning," *Mobile Inf. Syst.*, vol. 2021, pp. 1–18, Mar. 2021.

[29] Z. Namrud, S. Kpodjedo, C. Talhi, A. Bali, and A. B. Belle, "Deep learning based Android anomaly detection using a combination of vulnerabilities dataset," *Appl. Sci.*, vol. 11, no. 16, p. 7538, Aug. 2021.

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

[31] D. Gibert, C. Mateu, J. Planes, and R. Vicens, "Using convolutional neural networks for classification of malware represented as images," *J. Comput. Virol. Hacking Techn.*, vol. 15, no. 1, pp. 15–28, Mar. 2019.

[32] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," 2015, *arXiv:1511.08458*.

[33] P. Maniriho, A. N. Mahmood, and M. J. M. Chowdhury, "Deep learning models for detecting malware attacks," 2022, *arXiv:2209.03622*.

[34] J.-S. Luo and D. C.-T. Lo, "Binary malware image classification using machine learning with local binary pattern," in *Proc. IEEE Int. Conf. Big Data*, Dec. 2017, pp. 4664–4667.

[35] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *Proc. 8th Int. Symp. Visualizat. Cyber Secur.*, Jul. 2011, pp. 1–7.

[36] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, and Q. Zheng, "IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture," *Comput. Netw.*, vol. 171, Apr. 2020, Art. no. 107138.

[37] D. Vasan, M. Alazab, S. Wassan, B. Safaei, and Q. Zheng, "Image-based malware classification using ensemble of CNN architectures (IMCEC)," *Comput. Secur.*, vol. 92, May 2020, Art. no. 101748.

[38] J. Su, D. V. Vasconcellos, S. Prasad, D. Sgandurra, Y. Feng, and K. Sakurai, "Lightweight classification of IoT malware based on image recognition," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, vol. 2, Jul. 2018, pp. 664–669.

[39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–9.

[40] S. Yerima. (Feb. 2018). *Android Malware Dataset for Machine Learning 2*. [Online]. Available: https://figshare.com/articles/dataset/Android_malware_dataset_for_machine_learning_2/5854653

[41] P. Agrawal and B. Trivedi, "Androhealthcheck: A malware detection system for Android using machine learning," in *Computer Networks, Big Data and IoT*. Berlin, Germany: Springer, 2021, pp. 35–41.

[42] A. Mathur, L. M. Podila, K. Kulkarni, Q. Niyaz, and A. Y. Javaid, "NATICUSdroid: A malware detection framework for Android using native and custom permissions," *J. Inf. Secur. Appl.*, vol. 58, 2021, Art. no. 102696.

[43] C. Obidiagha. (May 2024). *Python Codes Used to Develop Deepimagedroid*. [Online]. Available: https://github.com/chimeziepy/DeepImageDroid

[44] Z.-H. Zhou, *Ensemble Methods: Foundations and Algorithms*. Boca Raton, FL, USA: CRC Press, 2012.

[45] D. Opitz and R. Maclin, "Popular ensemble methods: An empirical study," *J. Artif. Intell. Res.*, vol. 11, pp. 169–198, Aug. 1999.

[46] S. J. I. Ismail, B. Rahardjo, T. Juhana, and Y. Musashi, "MalSSL—Self-supervised learning for accurate and label-efficient malware classification," *IEEE Access*, vol. 12, pp. 58823–58835, 2024.

[47] G. Montavon, A. Binder, S. Lapuschkin, W. Samek, and K.-R. Müller, "Layer-wise relevance propagation: An overview," in *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, 2019, pp. 193–209.

[48] L. Antwarg, R. M. Miller, B. Shapira, and L. Rokach, "Explaining anomalies detected by autoencoders using Shapley additive explanations," *Exp. Syst. Appl.*, vol. 186, Dec. 2021, Art. no. 115736.

[49] P. R. Chandre, V. Vanarote, R. Patil, P. N. Mahalle, G. R. Shinde, M. Nimbalkar, and J. Barot, "Explainable ai for intrusion prevention: A review of techniques and applications," in *Proc. Int. Conf. Inf. Commun. Technol. Intell. Syst.* Springer, 2023, pp. 339–350.

COLLINS CHIMEZIE OBIDIAGHA is currently pursuing the master's degree in cybersecurity with Fordham University, New York City, NY, USA. He is also an experienced Information Technology Consultant with a strong background in computer network engineering, systems repair and maintenance, computer programming, web design and development, and graphic design.

MOHAMED RAHOUTI (Member, IEEE) received the M.S. degree from the Mathematics Department, University of South Florida, Tampa, FL, USA, in 2016, and the Ph.D. degree from the Electrical Engineering Department, University of South Florida, in 2020. He is currently an Assistant Professor with the Department of Computer and Information Science, Fordham University, New York City, NY, USA. His current research interests include computer networking and security, blockchain and cryptocurrency technologies, and artificial intelligence/machine learning.

THAIER HAYAJNEH (Senior Member, IEEE) received the M.S. and Ph.D. degrees in information sciences with specialization in cybersecurity and networking from the University of Pittsburgh, PA, USA. He is currently the Founder and the Director of the Fordham Center for Cybersecurity, an University Professor, and the Director of the Cybersecurity Programs, Fordham University, New York City. He has over two decades of experience in cybersecurity research and education. His research interests include cybersecurity and artificial intelligence, including system security, applied cryptography, blockchain and cryptocurrency, the IoT security, privacy, and forensics. Over the past few years, he received over $12 million in federal funding from NSA, DoD, and NSF to support cybersecurity research, workforce development, advanced curriculum, capacity building, and scholarship for service programs. He has published over 100 papers in reputable journals and conferences, where his publications were cited over 5,000 times. He also served as the Editor-in-Chief, an editor, an associate editor, a guest editor, and the program chair for several reputable journals and leading conferences.