Android Malware Detection Through CNN Ensemble Learning on Grayscale Images

El Youssofi Chaymae, Chougdali Khalid
Engineering Sciences Laboratory, Ibn Tofail University
Kenitra, Morocco

***Abstract*—With Android's widespread adoption as the leading mobile operating system, it has become a prominent target for malware attacks. Many of these attacks employ advanced obfuscation techniques, rendering traditional detection methods, such as static and dynamic analysis, less effective. Image-based approaches provide an alternative for effective detection that addresses some limitations of conventional methods. This research introduces a novel image-based framework for Android malware detection. Using the CICMalDroid 2020 dataset, Dalvik Executable (DEX) files from Android Package (APK) files are extracted and converted into grayscale images, with dimensions scaled according to file size to preserve structural characteristics. Various Convolutional Neural Network (CNN) models are then employed to classify benign and malicious applications, with performance further enhanced through a weighted voting ensemble optimized by Bayesian Optimization to balance the contribution of each model. An ablation study was conducted to demonstrate the effectiveness of the six-model ensemble, showing consistent improvements in accuracy as models were added incrementally, culminating in the highest accuracy of 99.3%. This result surpasses previous research benchmarks in Android malware detection, validating the robustness and efficiency of the proposed methodology.**

***Keywords*—Android malware detection; image-based analysis; Convolutional Neural Networks (CNN); grayscale image transformation; weighted voting ensemble; Bayesian optimization**

I. Introduction

Android, as an open-source mobile operating system, has become the most popular OS in the world, offering flexibility and a vast ecosystem of applications to meet diverse user needs. In 2024, Android commands 71.74% of the mobile OS market and has a user base of more than 3.3 billion [1], [2]. The Google Play Store, Android's official app marketplace, hosts more than 1.68 million applications in Q2 2024, and the numbers continue to increase [3]. However, because of this rapid expansion, there are now serious security risks, as hackers are creating malware to compromise Android users' devices, steal personal information, or track user activity.

Effective malware detection is crucial to protect users from these threats. Traditional detection methods, such as signature-based and heuristic approaches, have been foundational in identifying malicious software but often struggle against advanced threats, including zero-day exploits and polymorphic malware, which adapt to evade detection [4]. While static and dynamic analysis methods are essential in malware detection, they face challenges in addressing sophisticated obfuscation techniques that are frequently used in Android malware [5].

Artificial Intelligence (AI) has emerged as a promising solution to these challenges. AI, through machine learning (ML) and deep learning (DL) models, enables the analysis of extensive datasets to identify complex patterns indicative of malware, even in obfuscated applications [6]. Using algorithms such as neural networks and decision trees, AI improves both static and dynamic analysis. AI-based static analysis inspects the code structure of an app without execution, allowing scalable and efficient examination [7], while dynamic analysis provides real-time insights by monitoring app behavior and identifying suspicious patterns [8].

Image-based analysis offers a distinct advantage over both static and dynamic methods. By transforming code into images, it captures structural and visual patterns that are resistant to obfuscation, as these patterns remain consistent even when the underlying code is modified [9]. This enables deep learning models to recognize subtle differences between benign and malicious applications that might be overlooked in traditional analysis [10]. Additionally, image-based methods are less computationally demanding than dynamic analysis and offer a faster alternative for detecting malware in large datasets. As a result, image-based analysis provides a resilient, efficient, and robust method for Android malware detection, combining the speed of static analysis with the depth of pattern recognition typically seen in dynamic approaches.

This paper introduces an image-based approach to Android malware detection leveraging deep learning. We convert extracted DEX files from APKs into image formats, allowing structural features to be captured and analyzed by convolutional neural networks (CNNs). Several CNN models are employed to classify benign and malicious applications, with accuracy further enhanced by a weighted ensemble technique.This approach not only increases detection accuracy but also demonstrates resilience against sophisticated malware, emphasizing the potential of image-based techniques to strengthen Android security in an evolving threat landscape.

The main contributions of this study are structured as follows:

- Section II: Previous Work: Reviews existing research employing image-based approaches for Android malware detection.
- Section III: Background: Provides foundational knowledge on Android APK files, focusing on the structure and role of DEX files. Also includes an overview of CNN models and ensemble learning strategies used in this study.
- Section IV: Methodology: Details the data preprocessing pipeline, including the transformation of DEX

files into grayscale images, and describes the ensemble learning framework.

- Section V: Experiments and Results: Presents the experimental setup, evaluation metrics, and performance analysis. Includes an ablation study demonstrating incremental improvements and a comparative analysis with prior benchmarks.
- Section VI: Discussion and Challenges: Discusses the results, highlighting contributions and challenges.
- Section VII: Conclusion and Future Work: Summarizes the findings, emphasizing the study's significance, and proposes future directions.

II. Previous Work

Different studies have demonstrated that visualizing Android malware through image-based analysis using deep learning offers resilience beyond what static and dynamic analyses can sometimes achieve, effectively distinguishing between malicious and benign applications.

In 2019, Shao Yang [11] proposed an image-based Android malware detection method using CNNs. This approach converts Dalvik bytecode files ('classes.dex') into RGB images, capturing code patterns by mapping byte sequences to pixel values. The model, a CNN with eight hidden layers, detects malware directly from these RGB images, bypassing complex feature extraction. Tested on a dataset containing 10,540 samples, the method achieved an accuracy of 93%, with an average detection time of 0.22 seconds.

In 2020, Ding et al. [12] proposed an Android malware detection method based on bytecode images. Their approach involves extracting the 'classes.dex' file from APKs and converting it into grayscale images by transforming the byte stream into a two-dimensional matrix. Using convolutional neural networks (CNNs), their method automatically learns features without requiring complex decompiling or manual feature extraction. Tested on the Drebin dataset, it achieved an accuracy of 95.1%.

In 2020, Rahali et al. [13] proposed DIDroid, an image-based deep learning system for Android malware classification and characterization. By extracting features from APK files and transforming them into grayscale images, the system employs a convolutional neural network (CNN) to classify samples into 12 malware categories and 191 families. Tested on a large dataset of 400,000 apps (200,000 malware and 200,000 benign), achieving an accuracy of 93.36%.

In 2021, Zhang et al. [14] introduced an Android malware detection method that leverages temporal convolution networks (TCNs) and bytecode images. This approach combines the 'AndroidManifest.xml' file with the data section of the 'classes.dex' file to create grayscale images, capturing both structural and sequential bytecode features. By using TCN instead of traditional CNN, the model efficiently detects malware while reducing computational demands, achieving an accuracy of 95.44%.

In 2021, Bakour and Ünver [15] introduced DeepVisDroid, a hybrid Android malware detection model that combines image-based features with deep learning techniques. They created four grayscale image datasets by converting different files from APKs and extracted both local (e.g. SIFT, SURF, ORB) and global (e.g. color histogram, Hu moments) features for training. Using a 1D convolutional neural network model, DeepVisDroid achieved over 98% accuracy, outperforming traditional 2D CNN models and state-of-the-art methods in terms of accuracy and computational efficiency.

In 2022, Mitsuhashi and Shinagawa [16] conducted an extensive study on image-based malware variant classification, evaluating 24 CNN models with various fine-tuning levels on datasets like Malimg and Drebin. Their highest accuracy on Android malware classification was achieved with EfficientNetB4, reaching 93.65% on the Drebin dataset.

In 2022, Ullah et al. [17] developed a hybrid Android malware detection system using a combination of transfer learning and multi-model image representation. Their approach combines both textual and texture features from network traffic, leveraging transfer learning to create embeddings from network data and generating malware images for visual analysis. Using CNNs, they extracted texture features, and an ensemble model combined these with textual features for final classification. Tested on the CIC-AAGM2017 and CICMalDroid 2020 datasets, the system achieved 99% accuracy.

In 2023, Jo et al. [18] proposed a Vision Transformer (ViT)-based Android malware detection method that combines high accuracy with interpretability. Their approach converts DEX files into RGB images and uses the ViT model's attention mechanism to detect malware while identifying malicious behavior by highlighting influential areas within the image, allowing extraction of class and method names and providing insights into the malware's underlying behavior. Tested on real-world datasets, the model achieved an accuracy of 80.27% with an interpretability score of 0.70.

In 2024, Aldini and Petrelli [19] proposed a method for Android malware detection and classification by visualizing app data as grayscale images. Their approach uses a static analysis of files within APKs, such as 'classes.dex' and 'AndroidManifest.xml', converting these to grayscale images. Multiple convolutional neural network (CNN) models were applied to detect and classify malware, with CNN-LSTM and CNN-SVM models showing high accuracy rates. The study tested the method on datasets including Drebin and AndroZoo, achieving accuracy rates around 99% for detection and 97% for classification.

In 2024, Kiraz and Doğru [20] presented an image-based approach for Android malware detection, focusing on visualizing static features. They used the AndroPyTool to extract permissions, intents, receivers, and services from Android apps and converted these features into embedding vectors using the BERT algorithm. The embeddings were then transformed into images and classified with a CNN model. Tested on the CICMalDroid 2020 dataset, their method achieved an accuracy of 91%.

In 2024, Tang et al. [21] introduced an Android malware detection approach that utilizes a unique mixed bytecode image combined with an attention mechanism. Their method processes Android executable files by converting bytecode into grayscale and Markov images, then fusing these into a mixed image for enhanced feature representation. This approach

integrates channel and spatial attention mechanisms within a ResNet model, improving classification accuracy. Testing on the Drebin and CICMalDroid 2020 datasets, their model achieved an accuracy of 98.67%.

In 2024, Wang et al. [22] proposed an Android malware detection method based on RGB images with multi-feature fusion. Their approach extracts features from DEX files, AndroidManifest.xml files, and API calls, converting each into grayscale images enhanced through techniques like Canny edge detection and histogram equalization. These images are then merged into RGB images, with each channel representing a different feature type. Tested on the CICMalDroid 2020 dataset with models like AlexNet, GoogleNet, and ResNet, the method achieved an accuracy of 97.25%.

In 2024, Yapici [23] introduced an image-based approach for Android malware detection, converting Dalvik bytecode files into grayscale and RGB images for deep learning analysis. Addressing issues of dataset duplication and class imbalance, the study incorporates data cleaning and augmentation to improve result accuracy. This method achieved an accuracy of 98.7%.

III. Background

A. Broader Security Innovations

As Android malware continues to evolve in complexity, advancements in security technologies offer critical complementary strategies to APK analysis. For instance, methods like reliable concurrent error detection have been developed to improve computational reliability, particularly in systems relying on elliptic curve cryptography, which is integral to secure communications [24]. Enhancements in elliptic curve techniques, such as binary Edwards curves optimized for resource-constrained environments, highlight significant progress in creating robust cryptographic frameworks for embedded systems [25].

Efforts to reduce the computational cost of cryptographic operations have also led to the design of low-cost S-box solutions, which are essential for encryption processes such as those employed in the Advanced Encryption Standard (AES) [26]. Furthermore, the development of constant-time cryptographic libraries for protocols like supersingular isogeny Diffie-Hellman (CSIDH) underscores the emphasis on mitigating timing attacks, thereby ensuring secure operations in the context of quantum-resistant cryptography [27].

In addition, the security of deeply embedded and cyber-physical systems, often constrained by limited resources, remains a focal point. Innovative approaches address challenges such as maintaining data confidentiality and integrity in environments requiring lightweight yet effective solutions [28].

These collective advancements contribute to fortifying the security landscape, enhancing the resilience and reliability of APK file analysis in identifying and mitigating Android malware.

B. APK File Structure

Android Package (APK) files serve as the standard format for distributing and installing applications on Android devices [29]. Essentially, an APK is a compressed archive containing various components that collectively define the app's functionality, behavior, and resources. The structure of an APK file is depicted in Fig. 1, and its key components include:

- Dalvik Bytecode (`classes.dex`): This file contains the compiled code that runs on the Android Runtime (ART) or Dalvik Virtual Machine (DVM). It defines the application's logic and behavior, including its methods, classes, and API calls. Due to its critical role in execution, the `classes.dex` file is often a focal point in malware detection research.
- Manifest (`AndroidManifest.xml`): This XML file provides essential metadata about the application, such as its package name, permissions, components (activities, services, etc.), and hardware/software requirements. Malware often manipulates the manifest file to gain unauthorized access or exploit vulnerabilities.
- Resources (`res/`): This folder contains non-compiled resources, such as layouts, images, and strings, that are used to define the application's user interface and content.
- Compiled Resources (`resources.arsc`): This file stores compiled resource data in binary form, optimized for efficient runtime access by the application.
- Assets (`assets/`): A directory for additional files that the application needs at runtime, such as configuration files, data files, or embedded libraries.
- Native Libraries (`lib/`): This folder contains native code files (e.g. `.so` files) that are platform-specific, often used to optimize performance or access device-specific functionality.
- Signatures (`META-INF/`): This folder contains cryptographic signatures and certificates used to verify the integrity of the APK file. Modifications to the APK often invalidate its signature, signaling potential tampering.

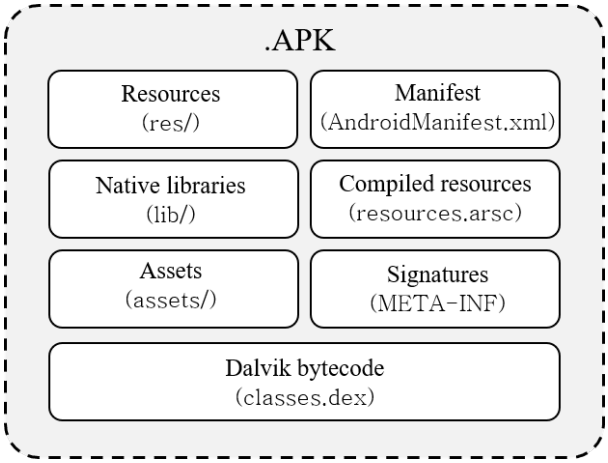

Fig. 1. APK file structure.

C. DEX File Structure

The Dalvik Executable (DEX) file is central to defining the behavior and logic of Android applications [29], comprising several essential sections that govern its execution flow. The structure of this file is illustrated in Fig. 2 and includes:

- Header: Contains metadata such as the magic number, checksum, file size, and offsets to other sections.
- String_IDs: Holds identifiers for strings, including class names, method names, and constant values.
- Type_IDs: Defines data types referenced in the application, including classes and primitives.
- Proto_IDs: Specifies method prototypes, detailing return types and parameter lists.
- Field_IDs: Provides definitions of fields within classes, specifying their types and names.
- Method_IDs: Enumerates methods, linking them to their respective classes and prototypes.
- Class_Defs: Describes each class, including its fields, methods, and metadata.
- Data Section: Contains supplementary information such as constants, initialization data, and debugging details.

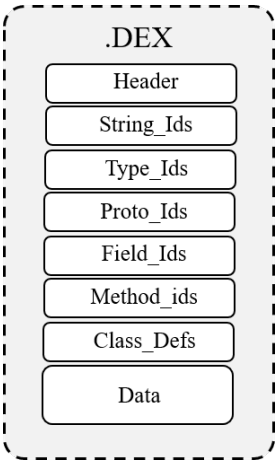

Fig. 2. DEX file structure.

D. CNN Models

In this study, six pre-trained convolutional neural network (CNN) architectures were employed for Android malware detection. Each model was chosen for its unique strengths and suitability for image-based classification tasks. Additionally, these models were fine-tuned for binary classification to distinguish between benign and malware samples:

- ResNet50 [30]: A residual network with 50 layers and 25.6 million parameters, excelling in avoiding vanishing gradient issues through its residual learning approach.
- AlexNet [31]: A classic CNN with 8 layers and 60 million parameters, known for its simplicity and speed, particularly effective on smaller datasets.
- DenseNet121 [32]: A densely connected network with 121 layers and 8 million parameters, designed to maximize feature reuse and minimize redundancy.
- MobileNetV2 [33]: A lightweight CNN with 53 layers and 3.4 million parameters, optimized for deployment on mobile and embedded systems.
- EfficientNetB0 [34]: A scaled CNN with 16 layers and 5.3 million parameters, achieving an excellent balance between high accuracy and computational efficiency.
- ShuffleNetV2 [35]: A channel-shuffling network with 50 layers and 2.3 million parameters, designed for extreme speed and low computational overhead.

E. Weighted Voting Ensemble

The Weighted Voting Ensemble is a technique that combines the predictions of multiple models by assigning each a weight based on its performance [36]. For a given input, the ensemble calculates the probability of classification as a weighted sum of individual model outputs:

$$P_{\text{ensemble}}(x) = \sum_{i=1}^{n} w_i \cdot P_i(x), \tag{1}$$

where $P_i(x)$ is the probability assigned by the i-th model for input x, w_i is its corresponding weight, and n is the total number of models in the ensemble. A threshold is then applied to determine the final classification. This approach leverages the complementary strengths of individual models, enhancing accuracy and reducing errors.

F. Bayesian Optimization

Bayesian Optimization is a probabilistic framework for optimizing expensive-to-evaluate functions [37]. It employs a surrogate model, typically a Gaussian Process (GP), to approximate the objective function $f(x)$, where $f(x)$ in this study represents the ensemble accuracy. The GP is characterized as:

$$f(x) \sim \mathcal{GP}(\mu(x), k(x, x')), \tag{2}$$

where $\mu(x)$ is the mean function, and $k(x, x')$ is the kernel function measuring similarity between points x and x'.

Using the Expected Improvement (EI) criterion, Bayesian Optimization iteratively refines weights by balancing exploration and exploitation:

$$\text{EI}(x) = \mathbb{E}[\max(f(x) - f(x^*), 0)], \tag{3}$$

where $f(x^*)$ is the best observed value of the objective function. This method efficiently identifies the optimal weight configuration to maximize ensemble accuracy.

IV. Methodology

A. Proposed Architecture

This study presents a novel image-based framework for Android malware detection, leveraging convolutional neural networks (CNNs) and a weighted voting ensemble to enhance detection accuracy.

The methodology is organized into three primary stages: data preprocessing, model training, and ensemble prediction.

In the preprocessing stage, the DEX file is extracted from Android APKs and transformed into grayscale images that encapsulate structural patterns indicative of malware. During model training, advanced CNN architectures are employed to analyze these images, enabling deep feature extraction and precise classification. Lastly, an optimized weighted voting ensemble integrates predictions from multiple models to improve overall performance and reliability.

The overall workflow of the proposed framework is illustrated in Fig. 3.

B. Data Preprocessing

In this study, the data preprocessing stage transforms Android APK files into grayscale images, which serve as input for the proposed deep learning framework. This transformation captures the structural patterns embedded in the DEX files, enabling robust malware detection. The preprocessing pipeline consists of the following steps:

1) DEX File Extraction: The first step involves extracting the `classes.dex` file from Android APKs. This file was chosen as the primary feature for analysis due to its structural richness and resilience to obfuscation techniques. Unlike other APK components, such as resource files or manifest configurations, the bytecode in the DEX file retains identifiable patterns that are critical for distinguishing between benign and malicious applications.

The extraction process treats the APK as a compressed archive, which is unzipped using Python's `zipfile` module. The `classes.dex` file is typically located in the root directory of the APK. To handle improperly named files, the script automatically renames files lacking the correct `.apk` extension, ensuring compatibility with the extraction process. This automated pipeline consistently prepares the DEX file for transformation into grayscale images.

2) Binary-to-Image Conversion: Once the DEX file is extracted, its binary content is read and converted into an 8-bit grayscale pixel matrix. Each byte in the binary sequence is mapped to a pixel intensity (0–255). This mapping ensures that the structural patterns in the bytecode are preserved in the resulting image.

The file size determines the width of the image, using a method adapted from Nataraj et al. [9]. The height is calculated dynamically to fit all pixels into a 2D matrix, using the formula:

$$\text{Height} = \frac{\text{Total Number of Bytes}}{\text{Image Width}}.$$

Table I outlines the mapping of file size ranges to image widths. This scaling ensures consistency while maintaining the integrity of structural characteristics.

TABLE I. Image Width Calculation Based on File Size

File Size Range	Image Width
<10 kB	32
10 kB – 30 kB	64
30 kB – 60 kB	128
60 kB – 100 kB	256
100 kB – 200 kB	384
200 kB – 500 kB	512
500 kB – 1000 kB	768
>1000 kB	1024

By retaining the sequence of bytecode as pixel intensities, this method preserves the unique structural characteristics of the application, such as opcode patterns and control flow representations. These features are critical to distinguish malware from benign applications.

C. Model Training

This study initially experimented with various pre-trained CNN architectures to identify the models most effective for classifying benign and malicious Android applications. Following extensive evaluations, six CNN models—ResNet50, AlexNet, DenseNet121, MobileNetV2, EfficientNetB0, and ShuffleNetV2—were selected for their complementary strengths in feature extraction and classification. These models demonstrated high performance in terms of accuracy, precision, recall, and F1-scores during preliminary testing. Each model was initialized with pre-trained weights from ImageNet and trained using a consistent pipeline to ensure fairness and comparability.

1) Image Preparation: The grayscale images generated from the `classes.dex` files were resized to a fixed input dimension of 224×224 pixels. They were normalized using the ImageNet dataset's mean ($[0.485, 0.456, 0.406]$) and standard deviation ($[0.229, 0.224, 0.225]$). Data augmentation techniques, including resizing and normalization, were applied to enhance model generalization.

2) Classifier Adaptation: Each model's classifier was customized for binary classification by replacing the original fully connected layers with the following configuration:

- A dropout layer ($p = 0.4$) to mitigate overfitting.
- A dense layer with 256 units and ReLU activation.
- A second dropout layer ($p = 0.4$).
- A final dense layer with one output node and a sigmoid activation function.

3) Training Process: The models were trained using the AdamW optimizer with a learning rate of 0.0001 and a weight decay of 1×10^{-5}. The binary cross-entropy loss function was employed for optimization. Training was conducted over 20 epochs with a batch size of 32. An early stopping mechanism,

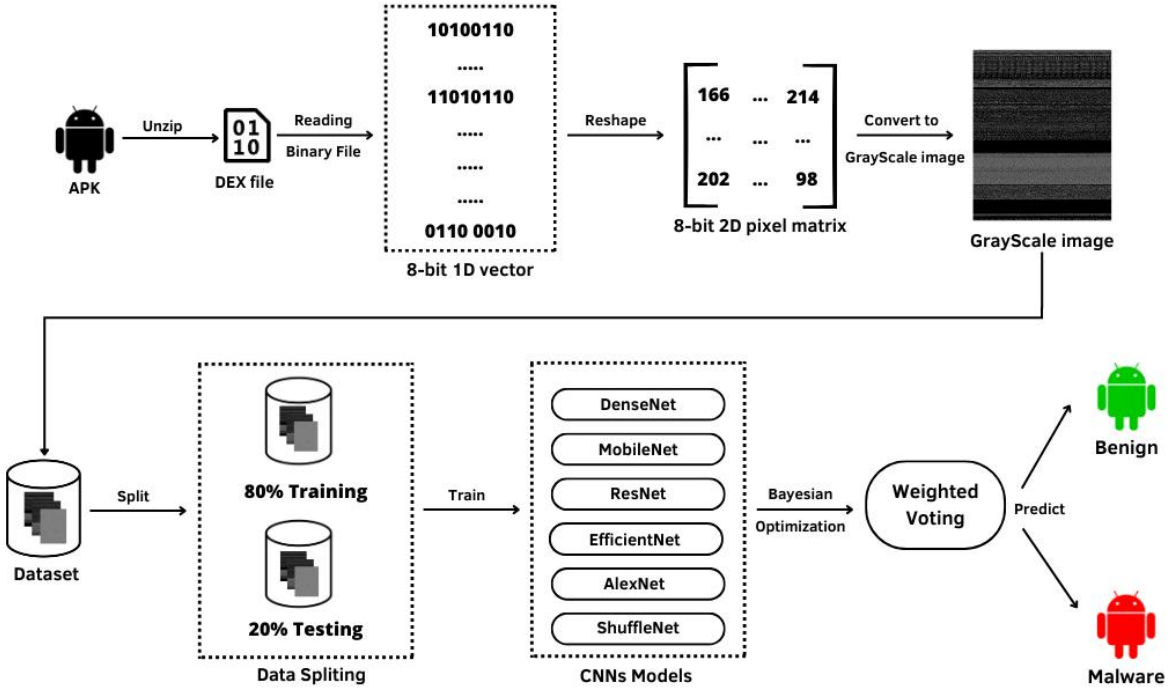

Fig. 3. The proposed architecture for android malware detection.

with a patience of 5 epochs, was implemented to prevent overfitting while ensuring optimal model performance.

D. Ensemble Prediction

To enhance the overall performance and robustness of the classification, an ensemble prediction strategy was employed. The ensemble combines the outputs of six pre-trained convolutional neural network (CNN) models: ResNet50, AlexNet, DenseNet121, MobileNetV2, EfficientNetB0, and ShuffleNetV2. By leveraging the strengths of each model, the ensemble aims to improve classification accuracy and reduce errors in detecting Android malware.

The ensemble prediction uses a weighted voting mechanism, where the outputs from each model are aggregated based on their performance during validation. For each input image, the probability of classification is calculated as a weighted sum of the individual model predictions. Weights were constrained to [0, 1], normalized to sum to 1, and refined over 300 iterations using the Expected Improvement (EI) criterion.

Bayesian Optimization was employed to determine the optimal weights for the ensemble, as detailed in Algorithm 1. This probabilistic technique was chosen for its ability to efficiently explore high-dimensional parameter spaces while balancing exploration with exploitation. Unlike manual selection or traditional methods such as grid or random search, Bayesian Optimization leverages information from prior iterations to refine the weight configurations, resulting in faster convergence and more effective optimization.

Algorithm 1 Bayesian Optimization for Ensemble Weights

Require: Validation dataset $\mathcal{D}$, pre-trained models: {ResNet50, AlexNet, DenseNet121, MobileNetV2, EfficientNetB0, ShuffleNetV2},
number of iterations $N = 300$, batch size $= 32$.
Ensure: Optimal weights $\{w_1, w_2, \ldots, w_6\}$ for the ensemble.
1: Initialize bounds $[0, 1]$ for each w_i and randomly generate initial weights.
2: **for** $i = 1$ to N **do**
3: Generate candidate weights $\{w_1, w_2, \ldots, w_6\}$.
4: Normalize weights such that $\sum_{i=1}^{6} w_i = 1$.
5: Compute ensemble prediction for each image x:

$$P_{\text{ensemble}}(x) = \sum_{i=1}^{6} w_i \cdot P_i(x)$$

6: Evaluate ensemble accuracy on $\mathcal{D}$.
7: Refine weights using the Expected Improvement (EI) criterion:
8: a. Explore unvisited regions of the weight space.
9: b. Exploit regions with promising results.
10: Update surrogate model with new results.
11: **end for**
12: **return** Optimal weights $\{w_1, w_2, \ldots, w_6\}$ achieving the highest ensemble accuracy.

V. Evaluation and Result

A. Dataset

In this study, the CICMalDroid 2020 dataset [38] was utilized, a benchmark dataset designed to support research and development in Android malware detection. The dataset was created by the Canadian Institute for Cybersecurity (CIC) and contains a total of 17,341 APK files. It categorizes APKs into five classes: **Adware**, **Banking**, **SMS**, **Riskware**, and **Benign**. Each class represents a specific type of Android application behavior, with malware types grouped based on their malicious intent, and benign applications serving as the control group.

1) Dataset Preparation: During the preprocessing stage, 415 APK files were excluded due to various issues, including:

- Missing or inaccessible `classes.dex` files.
- Permission restrictions preventing file access.
- Corrupted or non-standard file formats.
- Invalid or unusual filenames.
- Integrity check failures, such as bad CRC-32.

These issues prevented consistent processing of a subset of the dataset. As a result, only valid and complete files were included to maintain data quality and ensure reliable model training.

2) Data Splitting: Initially, the dataset was split into training (80%) and validation (20%) subsets for each class, ensuring balanced representation in both subsets. Following this, all malware classes were concatenated into a single class named **Malware** for binary classification, while the **Benign** class remained unchanged. This process resulted in the following data distribution:

- Benign: 3,216 samples for training and 805 for validation.
- Malware: 10,001 samples for training and 2,502 for validation.

B. Experimental Setup and Performance Metrics

The experiments were conducted on a Windows operating system using Python as the programming language. The models were implemented with the PyTorch 2.4.1 deep learning framework, and the training process was accelerated using CUDA 11.8 on an NVIDIA RTX 2060 GPU.

To evaluate the performance of the proposed models, a variety of metrics were employed to ensure a comprehensive understanding of their classification capabilities. Each metric serves a distinct purpose, offering insights into specific aspects of the models' performance and their ability to distinguish between benign and malware samples. The definitions of the metrics used are as follows:

- Accuracy (AC): Represents the overall proportion of correctly classified samples:

$$AC = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

where TP is true positives, TN is true negatives, FP is false positives, and FN is false negatives.

- Precision (P): Measures the proportion of true positive predictions out of all positive predictions:

$$P = \frac{TP}{TP + FP} \tag{5}$$

This metric highlights the model's ability to minimize false positives.

- Recall (R): Evaluates the proportion of true positive predictions among all actual positive samples:

$$R = \frac{TP}{TP + FN} \tag{6}$$

Recall is critical in malware detection to ensure that malicious applications are not overlooked.

- F1-Score (F): The harmonic mean of precision and recall, providing a balance between the two:

$$F = \frac{2 \cdot P \cdot R}{P + R} \tag{7}$$

The defined metrics were utilized to evaluate the classification performance of six individual CNN models and the proposed ensemble learning approach. Table II provides a comparative summary of the results, presenting the accuracy, precision, recall, and F1-score for each model. This analysis highlights the unique strengths and weaknesses of each model while demonstrating the superior performance of the ensemble approach.

TABLE II. Performance Metrics for All Models

Model	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
ResNet50	98.54	98.76	99.32	99.04
AlexNet	98.51	99.11	98.92	99.01
DenseNet121	98.42	99.59	98.32	98.95
MobileNetV2	98.45	99.35	98.60	98.97
EfficientNetB0	98.52	99.04	99.00	99.02
ShuffleNetV2	98.39	98.64	99.24	98.94
Ensemble Learning	**99.30**	**99.59**	**99.48**	**99.54**

The superior performance of the ensemble was achieved through the optimal weights determined by Bayesian Optimization. These weights were carefully assigned to balance the contributions of each model, reflecting their relative importance in the ensemble's predictions. Table III presents the final weights for each model, highlighting the significant contributions of MobileNetV2 and EfficientNetB0, which had the highest weights, underscoring their strong performance in feature extraction and classification.

For further insights into the models' classification performance, the Receiver Operating Characteristic (ROC) curves, presented in Fig. 4, offer a detailed evaluation of the individual models and the ensemble learning approach. All models exhibit excellent discriminatory capabilities, with Area Under the Curve (AUC) values exceeding 0.997. The ensemble learning approach outperformed all individual models, achieving the

TABLE III. OPTIMAL WEIGHTS FOR ENSEMBLE MODELS

Model	Optimal Weight
AlexNet	0.037
DenseNet121	0.172
EfficientNetB0	0.225
MobileNetV2	0.288
ResNet50	0.124
ShuffleNetV2	0.154

highest AUC of 0.9993. This superior performance highlights the ensemble's ability to effectively combine the strengths of individual models, resulting in enhanced classification accuracy.

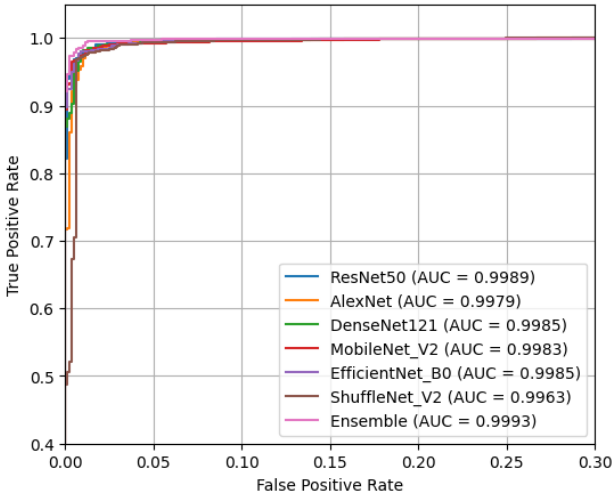

Fig. 4. ROC curves for CNN models and ensemble learning.

To provide a more detailed evaluation of the ensemble's predictions, the confusion matrix in Fig. 5 illustrates its classification outcomes for benign and malware samples. The model successfully classified 795 benign and 2,489 malware samples, with only 10 false positives and 13 false negatives. These results demonstrate the ensemble's high precision and recall, ensuring robust malware detection with minimal errors.

The proposed ensemble learning approach demonstrated significant improvements in Android malware detection accuracy compared to prior methods. Table IV provides a comparative analysis of existing approaches, highlighting datasets, models, and achieved accuracies. The proposed method outperformed techniques such as CNN-LSTM, CNN-SVM [20], and single-model approaches such as ResNet [19] and EfficientNet [16], it also surpassed studies employing multi-feature fusion strategies [22].

C. Ablation Study: Incremental Model Analysis

An ablation study was conducted to determine the optimal configuration of the ensemble by incrementally adding models and evaluating their impact on performance metrics. Starting with a baseline ensemble of ResNet50, AlexNet, and EfficientNetB0, additional models—MobileNetV2, DenseNet121, and ShuffleNetV2—were iteratively incorporated based on their individual metrics, such as precision, recall, and F1-score.

Each addition was rigorously evaluated for its contribution to accuracy and robustness, with weights re-optimized using Bayesian Optimization to ensure balanced contributions. The

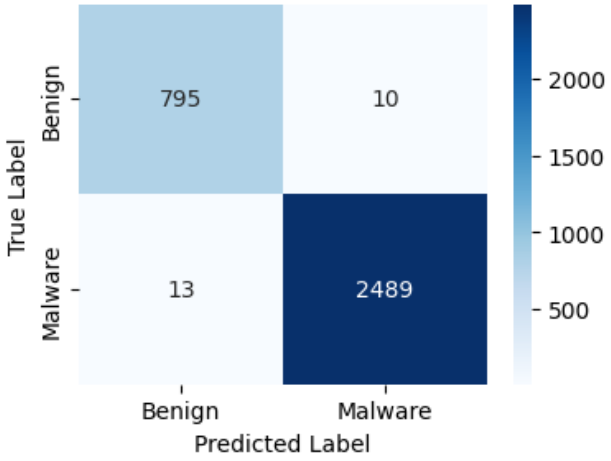

Fig. 5. Confusion matrix for the ensemble learning approach.

TABLE IV. COMPARATIVE ANALYSIS OF ANDROID MALWARE DETECTION METHODS

Ref	Dataset	Model	Acc (%)
[11]	AMD (6,134 malware), Google Play (4,406 benign)	CNN	93
[12]	Drebin	CNN	95.1
[13]	400,000 apps (200,000 malware, 200,000 benign)	CNN	93.36
[14]	CICAndMal2017, CICInvesAndMal2019, CICMalDroid 2020	TCN	95.44
[15]	Custom Dataset	DeepVisDroid (1D-CNN)	98
[16]	Malimg, Drebin	EfficientNetB4	93.65
[17]	CIC-AAGM2017, CICMalDroid 2020	Hybrid (Textual+Image)	99
[18]	Real-world datasets	ViT	80.27
[19]	Drebin, CICMalDroid 2020	ResNet	98.67
[20]	Drebin, AndroZoo	CNN-LSTM, CNN-SVM	99 detection, 97 classification
[21]	CICMalDroid 2020	CNN	91
[22]	CICMalDroid 2020	AlexNet, GoogleNet, ResNet	97.25
[23]	Custom Dataset (Dalvik Bytecode)	Grayscale+RGB Deep Learning	98.7
Our Method	**CICMalDroid 2020**	**Ensemble CNN Models**	**99.3**

study demonstrated a consistent progression in performance: accuracy improved from 99.09% with three models to 99.30% with six models, achieving the best performance across all metrics in the final configuration.

The results highlight the incremental benefits of integrating diverse architectures into the ensemble, validating the soundness of the methodology. Fig. 6 illustrates the steady improvements, emphasizing the rationale and effectiveness of this approach.

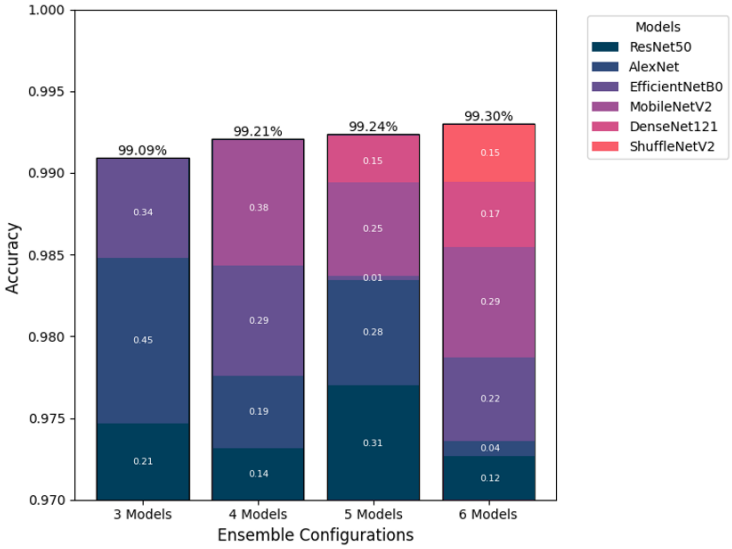

Fig. 6. Comparison of accuracy across ensemble learning configurations.

VI. Discussion and Challenges

The proposed ensemble approach achieved remarkable results, surpassing prior methodologies in Android malware detection with an accuracy of 99.3%, addressing challenges posed by evolving malware obfuscation techniques, particularly through the use of grayscale images derived from DEX files, which preserve critical bytecode patterns. The methodology not only outperformed single-model approaches, such as CNNs in [12] (95.1%) and TCN [14] (95.44%), but also exceeded advanced methods like CNN-LSTM [20] and multi-feature strategies [22], which achieved accuracies of 97% and 97.25%, respectively. This demonstrates the ensemble's capacity to generalize across diverse malware families while maintaining classification robustness.

However, several challenges were encountered during the study. One notable issue was overfitting, mitigated effectively by employing early stopping to ensure generalization without compromising performance. Another challenge was the lack of recent Android malware datasets. The rapid evolution of malware introduces the risk that outdated datasets may fail to capture current threats, limiting the generalizability of detection systems.

Furthermore, deploying a six-model ensemble in real-time environments poses practical difficulties due to the increased computational resources and latency required to run multiple models simultaneously. Addressing this limitation will be crucial for translating the proposed approach into real-world applications.

Despite these challenges, the study demonstrates the potential of ensemble learning in advancing the state of Android malware detection. By combining multiple CNN architectures and optimizing their contributions, this methodology provides a robust framework that paves the way for future research on integrating multi-feature analysis and scaling to larger datasets.

VII. Conclusion

This study introduced a novel ensemble learning framework for Android malware detection, utilizing grayscale images derived from DEX files and six pre-trained CNN models. Achieving an impressive accuracy of 99.3%, the proposed method surpassed existing approaches in the field. By leveraging a weighted voting mechanism, optimized through Bayesian Optimization, the ensemble demonstrated superior performance across key metrics, achieving precision, recall, and F1-scores of 99.59%, 99.48%, and 99.54%, respectively. This robust approach effectively minimized classification errors while ensuring reliable malware detection.

The findings underscore the benefits of combining multiple CNN architectures to harness their complementary strengths in feature extraction and classification. Additionally, the framework demonstrated resilience against obfuscation techniques frequently used by malware developers, enhancing its practicality for real-world applications.

Future research will focus on expanding this methodology to encompass larger, more diverse datasets and integrating multi-feature approaches with advanced analysis techniques. These efforts aim to further improve detection accuracy and adaptability, addressing the ever-evolving landscape of Android malware threats.

Acknowledgments

The authors would like to express their sincere gratitude to the Engineering Sciences Laboratory, National School of Applied Sciences, Ibn Tofail University, Kenitra, Morocco for providing the resources and support needed to carry out this research. The authors also extend their thanks to the CNRST (Centre National pour la Recherche Scientifique et Technique) for its financial support under the "PhD-Associate Scholarship - PASS" Program.

References

[1] "Android Statistics (2024)," Business of Apps. https://www.businessofapps.com/data/android-statistics/

[2] "Mobile & Tablet Android Version Market Share Worldwide," StatCounter Global Stats. https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide

[3] "Google Play Store: number of apps 2024," Statista. https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/

[4] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," Comput. Secur., vol. 81, pp. 123–147, Mar. 2019, doi: 10.1016/j.cose.2018.11.001.

[5] W. F. Elsersy, A. Feizollah, and N. B. Anuar, "The rise of obfuscated Android malware and impacts on detection methods," PeerJ Comput. Sci., vol. 8, p. e907, Mar. 2022, doi: 10.7717/peerj-cs.907.

[6] Z. Wang, Q. Liu, and Y. Chi, "Review of Android Malware Detection Based on Deep Learning," IEEE Access, vol. 8, pp. 181102–181126, 2020, doi: 10.1109/ACCESS.2020.3028370.

[7] F. Hamid, "Enhancing Malware Detection with Static Analysis using Machine Learning," Int. J. Res. Appl. Sci. Eng. Technol., vol. 7, no. 6, pp. 38–42, Jun. 2019, doi: 10.22214/ijraset.2019.6010.

[8] T. Bhatia and R. Kaushal, "Malware detection in android based on dynamic analysis," in 2017 International Conference on Cyber Security And Protection Of Digital Services (Cyber Security), Jun. 2017, pp. 1–6. doi: 10.1109/CyberSecPODS.2017.8074847.

[9] L. Nataraj, S. Karthikeyan, G. Jacob, and B. Manjunath, "Malware Images: Visualization and Automatic Classification," Jul. 2011, doi: 10.1145/2016904.2016908.

[10] V. Sihag, S. Prakash, G. Choudhary, N. Dragoni, and I. You, "DIMDA: Deep Learning and Image-Based Malware Detection for Android," in Futuristic Trends in Networks and Computing Technologies, P. K. Singh, S. T. Wierzchoń, J. K. Chhabra, and S. Tanwar, Eds., Singapore: Springer Nature, 2022, pp. 895–906. doi: 10.1007/978-981-19-5037-7_64.

[11] Shao Yang, "An Image-Inspired and CNN-Based Android Malware Detection Approach," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, San Diego, CA, USA: IEEE, Nov. 2019, pp. 1259–1261. doi: 10.1109/ASE.2019.00155.

[12] Y. Ding, X. Zhang, J. Hu, and W. Xu, "Android malware detection method based on bytecode image," *J Ambient Intell Human Comput*, vol. 14, no. 5, pp. 6401–6410, May 2023, doi: 10.1007/s12652-020-02196-4.

[13] A. Rahali, A. H. Lashkari, G. Kaur, L. Taheri, F. Gagnon, and F. Massicotte, "DIDroid: Android Malware Classification and Characterization Using Deep Image Learning," in _2020 the 10th International Conference on Communication and Network Security_, Tokyo Japan: ACM, Nov. 2020, pp. 70–82. doi: 10.1145/3442520.3442522.

[14] W. Zhang, N. Luktarhan, C. Ding, and B. Lu, "Android Malware Detection Using TCN with Bytecode Image," *Symmetry*, vol. 13, no. 7, Art. no. 7, Jul. 2021, doi: 10.3390/sym13071107.

[15] K. Bakour and H. M. Ünver, "DeepVisDroid: android malware detection by hybridizing image-based features with deep learning techniques," _Neural Comput & Applic_, vol. 33, no. 18, pp. 11499–11516, Sep. 2021, doi: 10.1007/s00521-021-05816-y.

[16] R. Mitsuhashi and T. Shinagawa, "Exploring Optimal Deep Learning Models for Image-based Malware Variant Classification," in *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*, Jun. 2022, pp. 779–788. doi: 10.1109/COMPSAC54236.2022.00128.

[17] F. Ullah, S. Ullah, M. R. Naeem, L. Mostarda, S. Rho, and X. Cheng, "Cyber-Threat Detection System Using a Hybrid Approach of Transfer Learning and Multi-Model Image Representation," _Sensors_, vol. 22, no. 15, p. 5883, Aug. 2022, doi: 10.3390/s22155883.

[18] 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," *Applied Sciences*, vol. 13, no. 11, p. 6839, Jun. 2023, doi: 10.3390/app13116839.

[19] A. Aldini and T. Petrelli, "Image-based detection and classification of Android malware through CNN models," in *Proceedings of the 19th International Conference on Availability, Reliability and Security*, Vienna Austria: ACM, Jul. 2024, pp. 1–11. doi: 10.1145/3664476.3670441.

[20] Ö. Kiraz and İ. A. Doğru, "Visualising Static Features and Classifying Android Malware Using a Convolutional Neural Network Approach," *Applied Sciences*, vol. 14, no. 11, p. 4772, May 2024, doi: 10.3390/app14114772.

[21] J. Tang *et al.*, "Android malware detection based on a novel mixed bytecode image combined with attention mechanism," *Journal of Information Security and Applications*, vol. 82, p. 103721, May 2024, doi: 10.1016/j.jisa.2024.103721.

[22] Z. Wang, Q. Yu, and S. Yuan, "Android Malware Detection Based on RGB Images and Multi-feature Fusion," Aug. 29, 2024, arXiv: arXiv:2408.16555. doi: 10.48550/arXiv.2408.16555.

[23] M. M. Yapici, "A Novel Image Based Approach for Mobile Android Malware Detection and Classification," 2024. doi: 10.2139/ssrn.4942956.

[24] M. Mozaffari-Kermani, R. Azarderakhsh, C.-Y. Lee, and S. Bayat-Sarmadi, "Reliable Concurrent Error Detection Architectures for Extended Euclidean-Based Division Over GF(2m̂)," IEEE Trans. Very Large Scale Integr. VLSI Syst., vol. 22, no. 5, pp. 995–1003, May 2014, doi: 10.1109/TVLSI.2013.2260570.

[25] B. Koziel, R. Azarderakhsh, and M. Mozaffari-Kermani, "Low-Resource and Fast Binary Edwards Curves Cryptography," in Progress in Cryptology – INDOCRYPT 2015, A. Biryukov and V. Goyal, Eds., Cham: Springer International Publishing, 2015, pp. 347–369. doi: 10.1007/978-3-319-26617-6_19.

[26] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "A low-cost S-box for the Advanced Encryption Standard using normal basis," in 2009 IEEE International Conference on Electro/Information Technology, Jun. 2009, pp. 52–55. doi: 10.1109/EIT.2009.5189583.

[27] A. Jalali, R. Azarderakhsh, M. M. Kermani, and D. Jao, "Towards Optimized and Constant-Time CSIDH on Embedded Devices," in Constructive Side-Channel Analysis and Secure Design, I. Polian and M. Stöttinger, Eds., Cham: Springer International Publishing, 2019, pp. 215–231. doi: 10.1007/978-3-030-16350-1_12.

[28] K.-K. R. Choo, M. M. Kermani, R. Azarderakhsh, and M. Govindarasu, "Emerging Embedded and Cyber Physical System Security Challenges and Innovations," IEEE Trans. Dependable Secure Comput., vol. 14, no. 3, pp. 235–236, May 2017, doi: 10.1109/TDSC.2017.2664183.

[29] R. Meier, "Professional Android Application Development". Indianapolis, IN, USA: Wiley Publishing, Inc., 2009.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Jun. 2016, pp. 770–778. doi: 10.1109/CVPR.2016.90.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," Commun ACM, vol. 60, no. 6, pp. 84–90, mai 2017, doi: 10.1145/3065386.

[32] G. Huang, Z. Liu, G. Pleiss, L. van der Maaten, and K. Q. Weinberger, "Convolutional Networks with Dense Connectivity," IEEE Trans. Pattern Anal. Mach. Intell., vol. 44, no. 12, pp. 8704–8716, Dec. 2022, doi: 10.1109/TPAMI.2019.2918284.

[33] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Jun. 2018, pp. 4510–4520. doi: 10.1109/CVPR.2018.00474.

[34] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," Sep. 11, 2020, arXiv: arXiv:1905.11946. doi: 10.48550/arXiv.1905.11946.

[35] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design," Jul. 30, 2018, arXiv: arXiv:1807.11164. doi: 10.48550/arXiv.1807.11164.

[36] A. Dogan and D. Birant, "A Weighted Majority Voting Ensemble Approach for Classification," in 2019 4th International Conference on Computer Science and Engineering (UBMK), Sep. 2019, pp. 1–6. doi: 10.1109/UBMK.2019.8907028.

[37] P. I. Frazier, "A Tutorial on Bayesian Optimization," Jul. 08, 2018, arXiv: arXiv:1807.02811. doi: 10.48550/arXiv.1807.02811.

[38] "MalDroid 2020 — Datasets — Research — Canadian Institute for Cybersecurity — UNB". https://www.unb.ca/cic/datasets/maldroid-2020.html