T.R.

GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

ANDROID MALWARE DETECTION BASED ON IMAGE RECOGNITION

SEDEF ERDOĞDU

SUPERVISOR
PROF. DR. İBRAHIM SOĞUKPINAR

GEBZE
2024

T.R.
GEBZE TECHNICAL UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

ANDROID MALWARE DETECTION BASED ON IMAGE RECOGNITION

SEDEF ERDOĞDU

SUPERVISOR
PROF. DR. İBRAHIM SOĞUKPINAR

2024
GEBZE

GEBZE TECHNICAL UNIVERSITY	GRADUATION PROJECT JURY APPROVAL FORM

This study has been accepted as an Undergraduate Graduation Project in the Department of Computer Engineering on 25/01/2024 by the following jury.

JURY

Member
(Supervisor) : Prof. Dr. İbrahim Soğukpınar

Member : Prof. Dr. İbrahim Soğukpınar

Member : Dr. Gökhan Kaya

ABSTRACT

Nowadays, as the popularity of mobile applications increases, the potential for malware to spread on these platforms is also increasing. With the increasing use of mobile devices today, security analysis of applications running on the Android platform is of great importance. The open nature of Android, the fact that users can download applications from third-party sources, can make it more vulnerable to such attacks. Android is one of the most widely used mobile operating systems worldwide, so it makes an attractive target for attackers. Malware developers can use various tactics to steal users' personal information, hijack their devices, or cause other damage.

This research aims to fill an important gap in the field of mobile security and develop an image recognition-based approach to detecting Android malware. Considering that traditional antivirus software is insufficient and has difficulty keeping up with the evolving nature of malware, visualization of features extracted from Android applications and image recognition analysis can contribute to the effective detection of mobile malware. For this reason, extracting features from the application, converting them into images, and classifying them by training a deep learning algorithm with this data can play a critical role in determining possible security threats.

This project aims to analyze the features of Android applications and classify applications as malware (malicious software) or benign (benign software) by these features. From the Android applications, "AndroidManifest.xml", "META-INF/CERT.RSA", "classes.dex", "resources.arsc" features were extracted and converted into images. With these converted feature images, the Convolutional Neural Network (CNN) model is trained. The trained model has been integrated into the Android application.

Keywords: Android, CNN, Malware Detection

ÖZET

Günümüzde mobil uygulamaların popülaritesi arttıkça kötü amaçlı yazılımların bu platformlara yayılma potansiyeli de artmaktadır. Günümüzde mobil cihazların kullanımının artmasıyla birlikte Android platformunda çalışan uygulamaların güvenlik analizi büyük önem taşımaktadır. Android'in açık doğası, kullanıcıların üçüncü taraf kaynaklardan uygulama indirebilmeleri, onu bu tür saldırılara karşı daha savunmasız hale getirebilir. Android, dünya çapında en yaygın kullanılan mobil işletim sistemlerinden biridir, bu nedenle saldırganlar için çekici bir hedef haline getirir. Kötü amaçlı yazılım geliştiricileri, kullanıcıların kişisel bilgilerini çalmak, cihazlarını ele geçirmek veya başka zararlara neden olmak için çeşitli taktikler kullanabilir.

Bu araştırma, mobil güvenlik alanındaki önemli bir boşluğu doldurmayı ve Android kötü amaçlı yazılımları tespit etmek için görüntü tanıma tabanlı bir yaklaşım geliştirmeyi amaçlamaktadır. Geleneksel virüsten koruma yazılımının yetersiz olduğu ve kötü amaçlı yazılımın gelişen doğasına ayak uydurmakta zorlandığı düşünüldüğünde, Android uygulamalarından çıkarılan özelliklerin görselleştirilmesi ve görüntü tanıma analizi, mobil kötü amaçlı yazılımın etkin bir şekilde algılanmasına katkıda bulunabilir. Bu nedenle uygulamadan özelliklerin çıkarılması, görüntülere dönüştürülmesi ve bu verilerle derin öğrenme algoritması eğitilerek sınıflandırılması olası güvenlik tehditlerinin belirlenmesinde kritik rol oynayabilir.

Bu proje, Android uygulamalarının özelliklerini analiz etmeyi ve uygulamaları bu özelliklerle kötü amaçlı yazılım (kötü amaçlı yazılım) veya iyi huylu (iyi huylu yazılım) olarak sınıflandırmayı amaçlamaktadır. Android uygulamalarından "AndroidManifest.xml", "META-INF/CERT.RSA", "classes.dex", "resources.arsc" özellikleri çıkarılıp ve görüntülere dönüştürüldü. Bu dönüştürülmüş görüntülerle Evrişimsel Sinir Ağları modeli eğitildi. Eğitilmiş model Android uygulamasına entegre edildi.

Anahtar Kelimeler: Android, CNN, Kötü Amaçlı Yazılım Tespiti

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my main advisor, Professor Dr. İbrahim Soğukpınar, for his invaluable guidance, patience, and support throughout this project. His professional mentorship has been instrumental in my successful completion of this endeavor. I would also like to thank Dr. Gökhan Kaya for serving as the jury member for this project.

Additionally, I would like to extend our heartfelt appreciation to my family. Their patience, understanding, and unwavering support have been indispensable in achieving the completion of this project. Without them, this accomplishment would not have been possible.

Sedef Erdoğdu

LIST OF SYMBOLS AND ABBREVIATIONS

Symbol or Abbreviation	:	**Explanation**
CNN	:	Convolutional Neural Network
AI	:	Artificial Intelligence
APK	:	Android Package Kit
PIL	:	Python Imaging Library

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

In recent years, the proliferation of Android devices has made them a primary target for malicious, commonly known as malware. As the number of Android apps continues to grow exponentially in various applications nowadays, the need for powerful malware detection mechanisms is becoming increasingly critical. Educating users about potential risks, safe app installation practices, and the importance of keeping software up-to-date is crucial. Users should be cautious when downloading apps from third-party stores and should be aware of the permissions requested by the apps they install. Traditional signature-based methods and behavioral analysis alone may not be enough to combat the evolving nature of Android malware.

Image Recognition-Based Android Malware Detection is a critical issue in terms of mobile security. The visualization of features in Android applications and the use of image recognition techniques offer a security measure beyond traditional methods. Traditional antivirus software is often inadequate and may have difficulty keeping up with the evolving nature of malware. In this regard, visualization of features extracted from Android applications can provide an enhanced layer of security. This research aims to protect users from potential dangers by providing a new perspective in the detection of mobile malware.

To enhance the precision and efficiency of malware detection, an extensive dataset comprising samples from diverse Android malware families was gathered. The dataset extraction process involved isolating key features from Android applications, specifically focusing on the ”AndroidManifest.xml”, ”META-INF/CERT.RSA”, ”classes.dex”, ”resources.arsc”. These extracted features were subsequently transformed into image representations. The CNN model is trained with these converted feature images.

This paper presents the Android malware detection system. Image recognition-based Android uses the CNN model for malware detection. By adopting image recognition techniques, the Android malware detection system classifies malicious samples and benign samples.

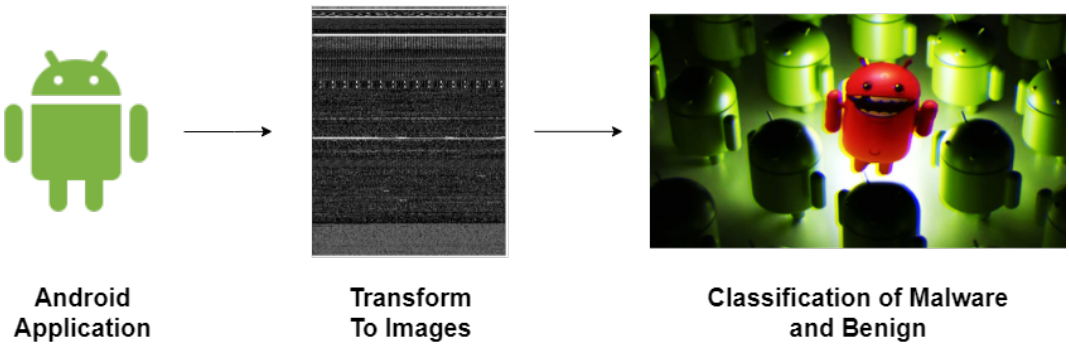

Figure 1.1: Schema of the project.

1.1. Project Definition

The project focuses on a beneficial image recognition algorithm to enable Android malware detection. An extensive dataset comprising samples from diverse Android malware families was gathered. The features extracted from Android applications will be visualized and malware will be detected with the help of image recognition. The system will utilize a combination of image collection, feature extraction, and classification methods to identify malicious and benign Android applications.

1.2. Goal of the Project

This project aims to develop an effective Android malware detection system using image recognition techniques. This approach aims to enhance traditional methods of malware detection by incorporating visual features extracted from Android applications. Through this project; users will be able to find out whether the applications downloaded to their devices are malware or benign.

1.3. Research and Literature Review

For the detection of unknown malicious code, most successful classification algorithms are based on PC file patterns, which are extracted features based on the inspected files or the files after disassembly. In this paper, the inspected APK files are represented using compressed image patterns from Android files, and a classification algorithm is used to find similarities between the malware images. [1].

In this paper, a malware classification model has been proposed for detecting malware samples in the Android environment. The proposed model is based on

converting some files from the source of the Android applications into grayscale images. Some image-based local features and global features, including four different types of local features and three different types of global features, have been extracted from the constructed grayscale image datasets and used for training the proposed model. The bag of visual words algorithm has been used to construct one feature vector from the descriptors of the local feature extracted from each image. The extracted local and global features have been used for training multiple machine learning classifiers including Random-forest, k-nearest neighbors, Decision Tree, Bagging, AdaBoost, and Gradient Boost. The proposed method obtained a very high classification accuracy reaching 98.75% with a typical computational time that does not exceed 0.018 s for each sample. [2].

In this paper, a unique image-based deep learning system for android malware detection has been proposed for detecting malware samples. The suggested system predicts if an application is malicious or genuine based on network traffic represented in picture format. The proposed method is tested against 13,533 applications from various banking, gambling, and utilities industries. The technique in the paper, is effective, with an accuracy of 98.44% and a recall of 98.30%. It also outperformed conventional machine learning methods. [3].

2. THEORETICAL FOUNDATIONS

2.1. Image Recognition

Image recognition is a field of artificial intelligence (AI) that focuses on enabling machines to interpret and understand visual information from images or videos. The primary goal is to develop algorithms and models that can replicate human-like visual perception. This involves extracting meaningful features from images, identifying patterns, and making sense of the content within the visual data. CNNs are commonly employed in image recognition tasks due to their ability to automatically learn hierarchical representations of features. During the image recognition process, the algorithm undergoes training on a labeled dataset, where it learns to recognize patterns and objects by adjusting its parameters. Once trained, the model can generalize its knowledge to new, unseen images, accurately classifying or detecting objects, faces, scenes, or other relevant information. Image recognition has diverse applications, including facial recognition, object detection, medical image analysis, autonomous vehicles, and more.

2.2. Deep Learning

Deep learning is a subfield of machine learning that focuses on the development and training of artificial neural networks, which are inspired by the structure and function of the human brain. The term ”deep” in deep learning refers to the use of deep neural networks, meaning networks with many layers (hence the term ”deep”). These networks are capable of learning and representing intricate hierarchical patterns and features from data.

At the core of deep learning are artificial neural networks, which are inspired by the structure and functioning of the human brain. A neural network consists of interconnected nodes, or artificial neurons, organized into layers. A neural network consists of interconnected nodes (neurons) organized into layers. The three main types of layers are the input layer, hidden layers, and output layer.

- Input Layer: This layer receives the initial data.
- Hidden Layers: These layers process the input data through weighted connections.
- Output Layer: This layer produces the final output or prediction

Each connection between neurons has a weight, and the network learns to adjust these weights during training. A deep neural network has more than one hidden layer, allowing it to learn hierarchical representations of data. The depth of the network allows it to automatically learn features at different levels of abstraction.

Each neuron in a neural network receives input from the previous layer, performs a weighted sum of these inputs, adds a bias term, and then applies an activation function to produce the final output. Common activation functions include sigmoid (2.1), hyperbolic tangent (tanh) (2.2), and rectified linear unit (ReLU)(2.3). The activation function introduces non-linearity into the model, allowing the neural network to learn complex relationships within the data.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

Sigmoid Activation Function

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.2}$$

Hyperbolic Tangent Activation Function

$$\text{ReLU}(x) = \max(0, x) \tag{2.3}$$

2.3. Convolutional Neural Network

CNN stands for Convolutional Neural Network. It is a type of artificial neural network designed for processing structured grid data, such as images or videos. CNNs are particularly effective in tasks related to computer vision, such as image and video recognition, image classification, object detection, and segmentation.

The key idea behind CNNs is the use of convolutional layers, which apply convolutional operations to input data. These convolutional operations involve passing a small filter (a matrix of weights) over the input data to detect patterns and features. The use of shared weights and biases in the convolutional layers allows CNNs to capture hierarchical and spatial features in the data, making them well-suited for tasks where the spatial arrangement of features is crucial.

CNNs typically consist of multiple layers, including convolutional layers, pooling layers (to reduce spatial dimensions), and fully connected layers (for classification or regression). The architecture of a CNN is designed to automatically and adaptively learn spatial hierarchies of features from the input data.

- Input Layer: The first layer takes input data into the network. Typically, this could be an image or a sequence of pixel values.

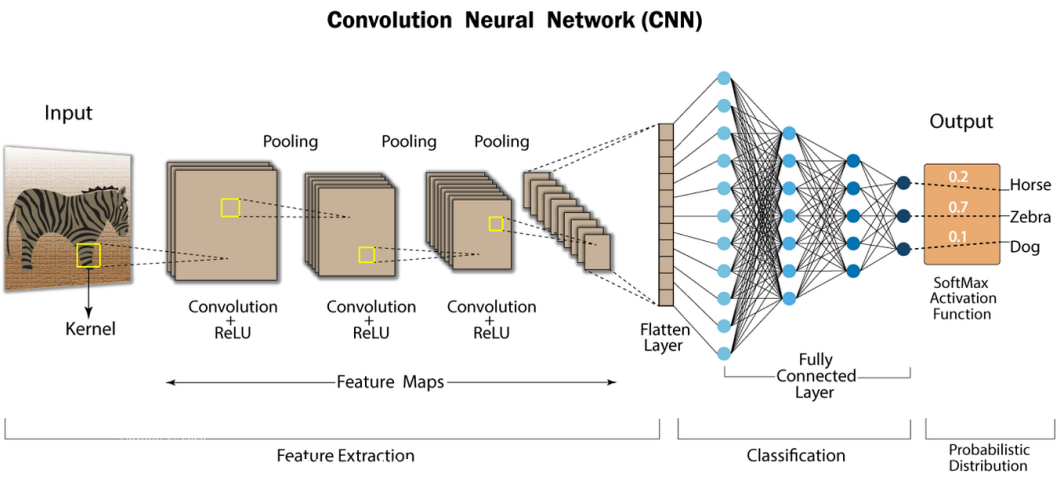

Figure 2.1: Convolutional Neural Network.

- Convolutional Layers: These layers apply a convolution operation on the input data, generating feature maps. Each convolutional layer contains a specific number of filters (kernels). Each filter is used to highlight specific features. Activation functions (usually ReLU (2.3)) add non-linearity to the output.
- Hidden Layers: Hidden layers typically come after convolutional layers and are used to further process and learn from the feature maps. They may include pooling layers (e.g., max pooling) or normalization layers (e.g., batch normalization).
- Pooling Layer: Pooling layers are used to downsample the spatial dimensions of the input volume. This is done to reduce the computational complexity of the network and make the learned features more invariant to scale and orientation changes.
  - Max Pooling: Takes the maximum value from a group of neighboring pixels in the input.
  - Average Pooling: Takes the average value from a group of neighboring pixels.
- Normalization Layer: Normalization layers, such as batch normalization, aim to improve the training stability and speed up convergence by normalizing the input to a layer.
  - Batch Normalization: Batch Normalization is a technique used in neural networks that normalizes the input of a layer by adjusting and scaling the activations, which helps improve training stability, accelerate convergence, and mitigate issues related to internal covariate shifts.

- Flattening Layer: At the end of the CNN, feature maps are flattened into a vector. This can be used as input to fully connected layers.
- Fully Connected Layers: The flattened vector is fed into fully connected layers. These layers use the learned features to compute probabilities for belonging to a specific class. The last fully connected layer often represents the output classes.
- Output Layer: The output layer, typically used in classification problems, employs the softmax activation function (2.4) to generate class probabilities. For regression problems, a linear activation function is commonly used.

$$\sigma_i(z) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \tag{2.4}$$

Softmax Activation Function

CNNs have been highly successful in various computer vision tasks and have significantly contributed to advancements in areas such as image recognition, facial recognition, and medical image analysis.

3. PROJECT DESIGN PLAN

Figure 3.1: Project Design Plan.

Project design plan includes these steps:

- Collecting Android application dataset
- Extract features from APK
- Convert features to images
- Model development
- Developing application and integrate model this application

3.1. Data Collection and Preperation

In this project, the android files data are collected from University of New Brunswick's CIC-AndMal2017 and CICMalDroid 2020 data. A dataset consisting of different Android malware families and benign files was created with the data collected from here. By extracting "AndroidManifest.xml", "META-INF/CERT.RSA", "classes.dex" and "resources.arsc". features from this dataset, images were obtained and a data set was created with these images for the model. Some preprocessing steps are applied to these inages. Images are labeled according to the appropriate classes as benign or malware.

3.2. Extracting Features

Android application file format is APK which is based on ZIP file format.
An APK file normally includes following files: [1]

- c1asses.dex: Dalvik opcode, which can be run by Oalvik virtual machine.
- AndroidManifest.xml: presents essential information about the APK file to the Android system, information the system must have before it can run any of the

classes.dex code. It describes the components of the application - the activities, services, broadcast receivers, and content providers, it declares the permissions and lists the libraries.

- META-INF folder which contains three files
  - MANIFEST.MF: this lists each file in the archive, and the base64-encoded SHAI hash of the contents of the file
  - CERT.SF: this is similar to the MANIFEST.MF file, except that instead of the SHA I hash of the contents for a file, it lists a SHA 1 hash of the lines for the file from the MANIFEST.MF file
  - CERT.RSA: this contains the signature of the CERT.SF file, as well as the certificate used for signing
- res: all the resource files need by the APK
- assets: the original resource files without complied
- resources.arsc: the compile binary resource files
- lib: the library folder

The APK files should be unzipped for the features to be extracted. After the unzip,"AndroidManifest.xml", "META-INF/CERT.RSA", "classes.dex" and "resources.arsc" features will be taken.

3.3. Convert Features to Images

"AndroidManifest.xml", "META-INF/CERT.RSA", "classes.dex" and "resources.arsc" are extracted from APK files. 4 files are read as a vector of 8-bit unsigned integers and then organized into a 2D array and are visualize as a grayscale image.

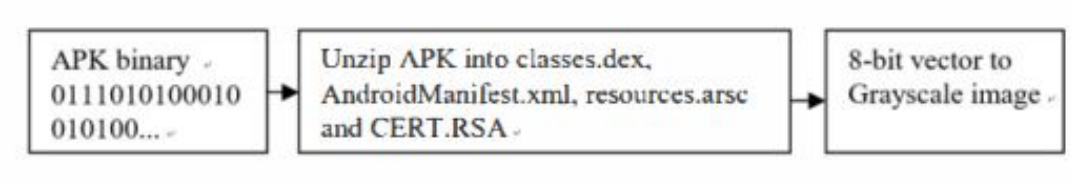

Figure 3.2: Visualizing APK as an image.[1]

3.4. Model Development

The CNN model is trained for the classification of images created with features extracted from Android files. The programming language that are considered to be used is Python. Keras and TensorFlow libraries will be used for the model. Keras is a high-level artificial neural network API (Application Programming Interface) library used to develop deep learning applications and TensorFlow is an open-source machine learning and deep learning library.

3.5. Developing Application and Integration Model

An Android application will be developed and the CNN model trained will be integrated into the model through the service. The application will send a request to the service, and the predictions of the model can be displayed via the application.

Android application provides a user interface for users to view APK file classification via the backend system. Server retrieves model prediction for APKs.

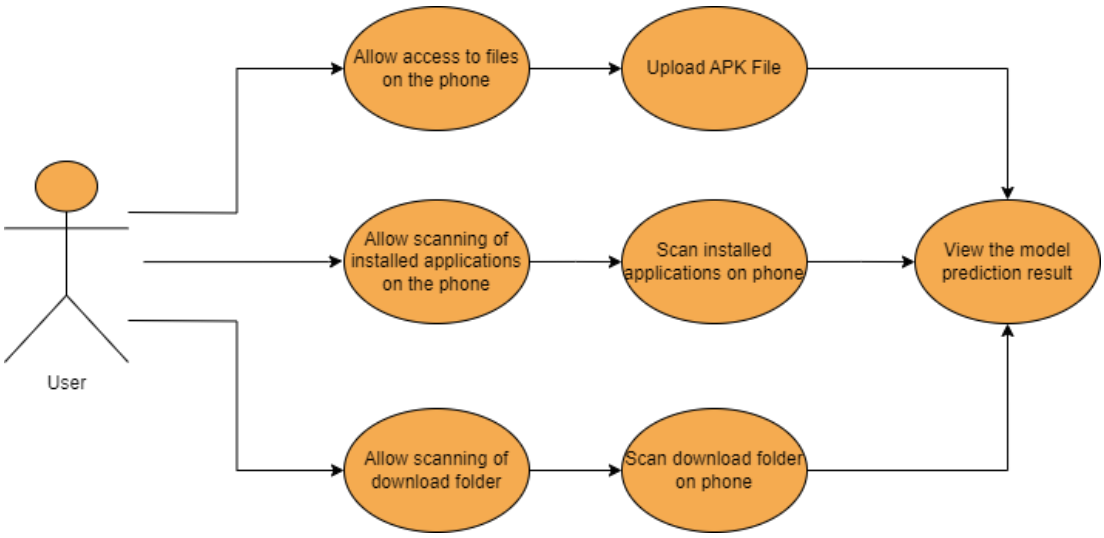

Figure 3.3: Use Case Diagram.

If the users allow access to the applications on their phones, they can send the APK files on their phone to the service for classification as malware or benign and view the prediction result of the model.

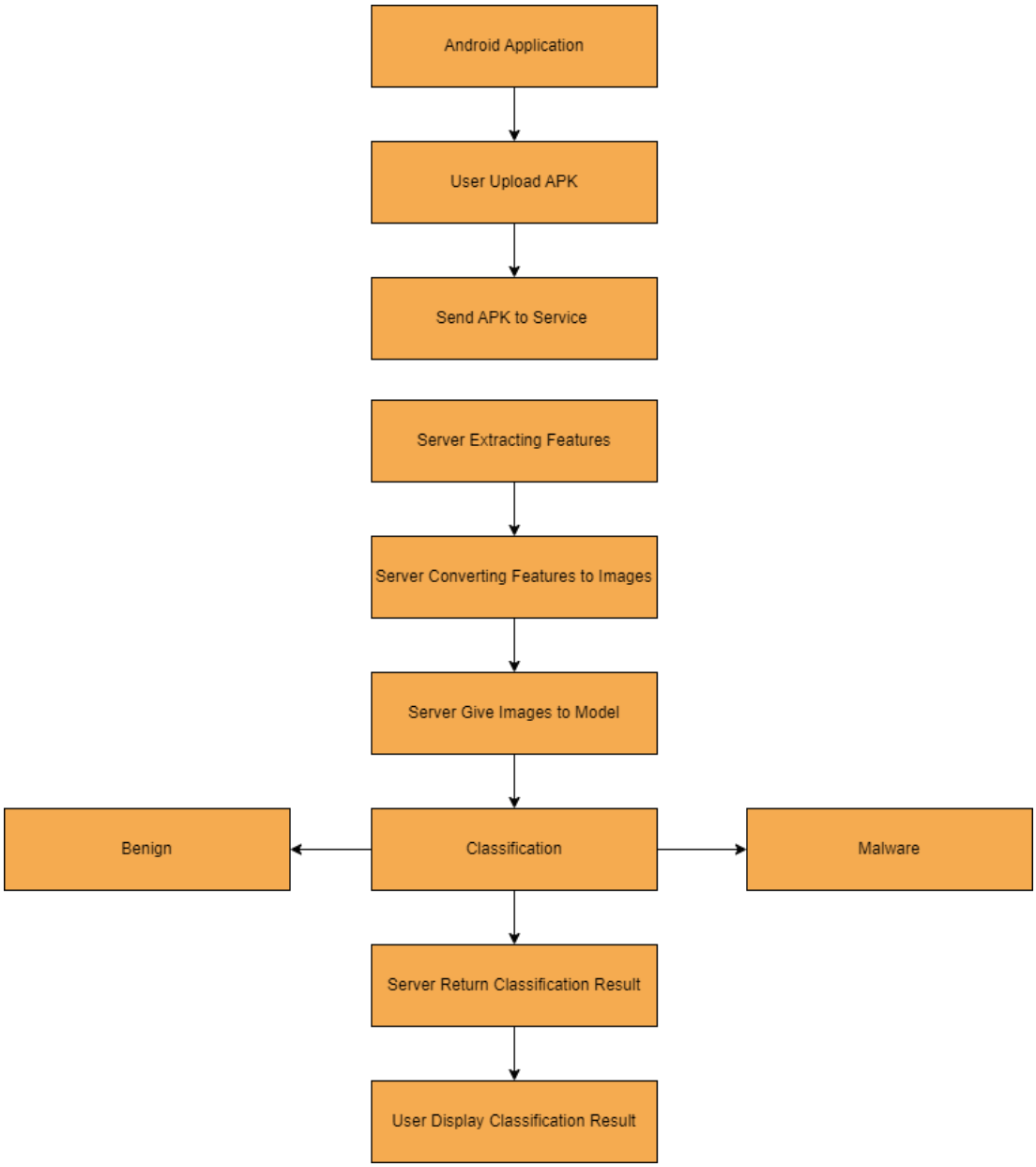

Figure 3.4: Data Flow Diagram.

The service receives the APK files uploaded by the user. The service unzips this APK file and extracts ”AndroidManifest.xml”, ”META-INF/CERT.RSA”, ”classes.dex” and ”resources.arsc” features. It covert these features into grayscale images and sends them to the model. It sends the prediction result of the model.

The programming languages that are considered to be used are Python for service and Kotlin for Android application.

4. IMPLEMENTATION AND EVALUATION

4.1. Dataset Details

In this project, the android files data are collected from the University of New Brunswick's CIC-AndMal2017 and CICMalDroid 2020 data. A dataset consisting of different Android malware families and benign files was created with the data collected from here.

There were corrupted files and duplicate files in the dataset. These have been identified and removed from the dataset. After this removal process, there are a total of 11090 files left in the dataset. There are 6163 pieces of malware and 4927 pieces of benign data.

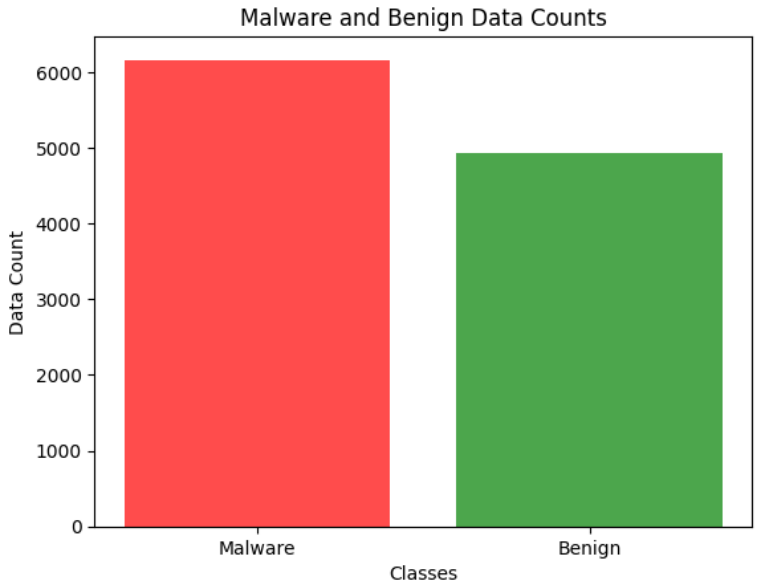

Figure 4.1: Malware and Benign Data Counts.

As can be seen in the graph 4.1, there is a class imbalance in the given.

Class imbalance refers to a situation where the number of examples belonging to one class is significantly greater or lesser than those belonging to other classes. Class imbalance is a crucial consideration, particularly when working with machine learning models. It indicates an uneven distribution of classes in the dataset used for training a model.

When there is a class imbalance in a dataset, training a model can present challenges due to this imbalance. The model often tends to perform better on examples

from the majority class while potentially neglecting examples from the minority class. This imbalance can create difficulties in accurately learning rare instances from the minority class.

Various strategies are employed to address class imbalance, including oversampling, undersampling, synthetic example generation, custom weights, or specialized loss functions. These strategies aim to reduce the imbalance during model training, promoting more balanced and generalized learning across classes.

Oversampling involves increasing the number of samples in the minority class, enabling the model to better learn from the underrepresented class. Undersampling achieves balance by decreasing the number of samples in the majority class, though this approach may result in data loss as some majority class samples are excluded.

Synthetic example generation techniques, such as SMOTE (Synthetic Minority Over-sampling Technique), focus on creating synthetic samples to mimic the minority class, thereby expanding its representation in the dataset.

Custom weights can be assigned to different classes in certain machine learning algorithms, allowing for a more effective handling of class imbalance during training.

Specialized loss functions can be tailored during model training to provide a more focused approach in addressing class imbalance, enhancing the model's ability to handle uneven class distributions.

In order to reduce the class imbalance, the undersampling method was applied. Data belonging to the malware class has been reduced. After this procedure, there are a total of 9978 files left in the dataset. There are 5051 pieces of malware and 4927 pieces of benign data.

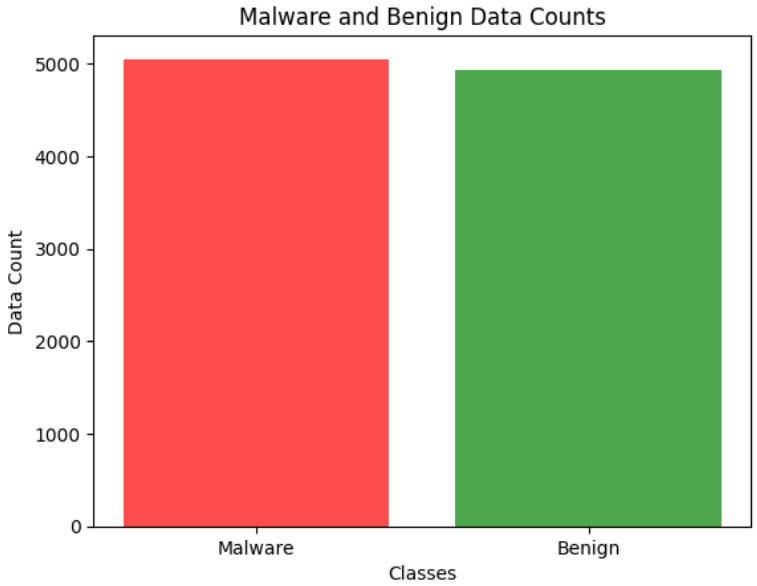

Figure 4.2: Malware and Benign Data Counts After Undersampling.

Android application file format is APK which is based on ZIP file format. The APK files should be unzipped for the features to be extracted. After the unzip, extract ”AndroidManifest.xml”, ”META-INF/CERT.RSA”, ”classes.dex” and ”resources.arsc” features from these applications. After the extracted 4 files are read as a vector of 8-bit unsigned integers, they are subsequently organized into a 2D array, and this array is then visualized as a grayscale image using Python Imaging Library (PIL) from Python. PIL is a library used for image processing in the Python language.

4 Images were obtained from each APK and a data set was created with these images for the model.

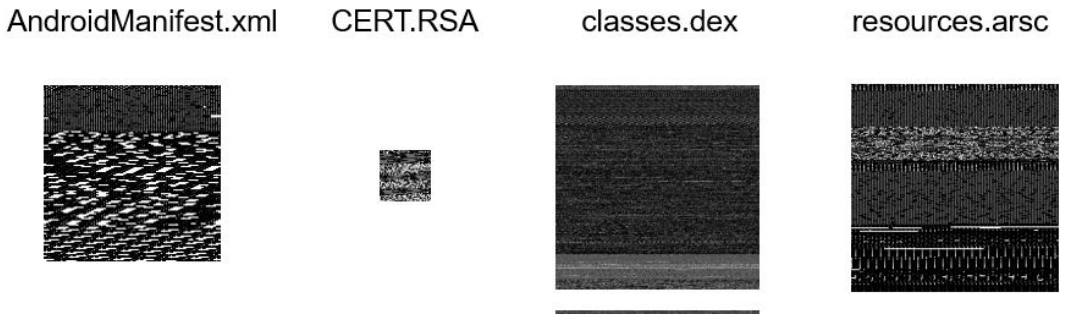

Figure 4.3: 4 Gray Scale Images for Malware Data.

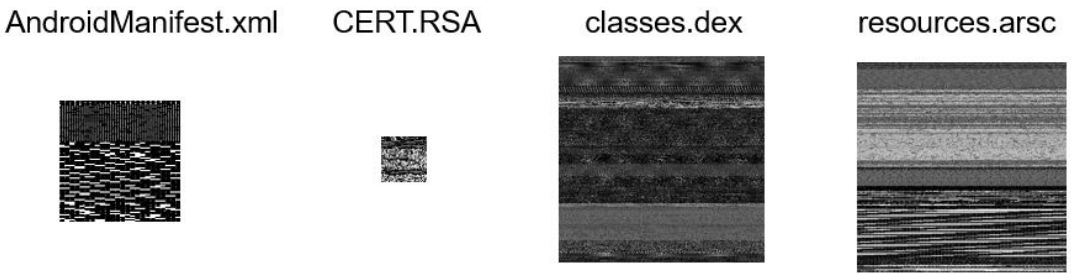

Figure 4.4: 4 Gray Scale Images for Benign Data.

4.2. Model Training and Evaluation

The CNN model is trained with images created by converting the extracted features from APKs into visual representations, allowing the neural network to learn and recognize intricate patterns, correlations, and anomalies associated with Android applications.

4.2.1. Preprocessing

Before initiating the model training, a preprocessing procedure is applied to each image within the created dataset. Each photograph is resized to 128x128, and the pixel values of the images are normalized to the range [0, 1].

The dimensions of photos and normalization play pivotal roles in the efficacy of CNNs. Properly managing the size of images is essential, influencing the computational cost and general performance of the model. Resizing all images to a standardized format ensures compatibility with the model's expectations and facilitates learning features at different scales. On the other hand, normalization, achieved by adjusting pixel values to a scale from 0 to 1, provides several benefits. It assists the model in learning more effectively, enhances overall numerical stability, and allows the model to perform well on various types of datasets.This preprocessing step not only accelerates the learning process but also makes the model more robust to outliers, contributing to its overall reliability and efficiency.

These operations have prepared the images in a suitable format that can be fed into the CNN model.

4.2.2. Training

Firstly, the data for each class is partitioned into 80% for training, 10% for validation, and 10% for testing, and benign data are labeled as 0, and malware data are labeled as 1. Random permutations are applied to shuffle the training, testing, and validation datasets. This is done to ensure that the model doesn't learn any spurious patterns related to the order of the data.

The Sequential class of the Keras library has been utilized in constructing the model. The Sequential class allows a model to be created by adding layers sequentially. A tensor with dimensions (128, 128, 4) is accepted as input data by the model. This tensor represents a matrix of a 128x128-pixel image, with 4 features at each pixel.

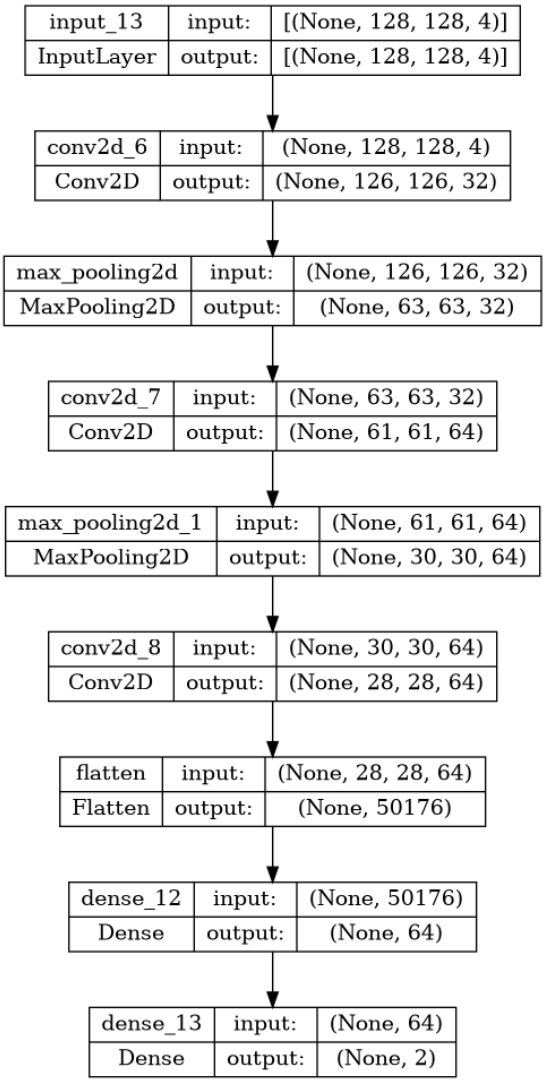

Figure 4.5: CNN Model.

The architecture of the model is as follows:

- A Conv2D layer has been added, employing 32 filters with a 3x3 window and applying the ReLU activation function (2.3).
- Subsequently, a MaxPooling2D layer has been added, and a maximum pooling operation with a window of size (2, 2) has been applied.
- Following these operations, another Conv2D layer has been added, this time using 64 filters. Again, a 3x3 window has been applied, and the ReLU activation function (2.3) has been employed.
- Another MaxPooling2D layer has been added with a window size of (2, 2), applying a maximum pooling operation.

- Finally, one more Conv2D layer has been added, utilizing 64 filters and passing through a 3x3 window, with the application of the ReLU activation function (2.3).
- After these layers, a flattening operation has been performed using the Flatten function.
- A Dense layer has been added on the flattened data, using 64 neurons and applying the ReLU activation function.
- Lastly, a Dense layer has been added as the output layer, utilizing 2 neurons and applying the softmax activation function (2.4).

This model represents a CNN that performs convolution and pooling operations on input data, followed by utilizing the learned features in fully connected layers for the classification process.

When compiling the model, the Adam optimizer has been employed as the optimization algorithm. This optimizer is a gradient-based optimization algorithm that manages the updating of weights by determining the learning rate. The learning rate regulates how much the weights will change in each update step. The learning rate, set to 1e-5 in this case, governs the step size used to update the model's weights.

During training, the model's performance is evaluated and optimized using the sparse categorical crossentropy loss function. This specific loss function is suitable for classification problems, especially when there are non-linear relationships between classes, and the classes are distinctly separated. The term "sparse" implies that labels are represented as integers, commonly used in multi-class classification problems.

4.2.3. Model Evaluation

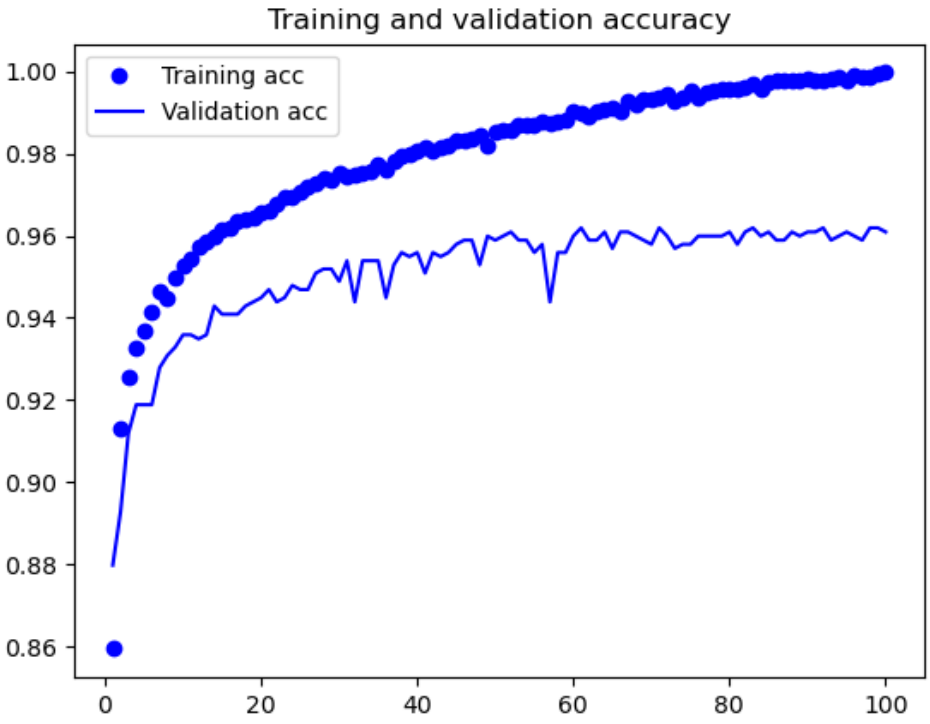

Figure 4.6: Training Validation Accuracy.

A training validation accuracy graph is shown in Figure 4.11.

Training accuracy indicates how well the model performs on the training dataset. It represents the percentage of correct predictions the model makes on the data it was trained on. Validation accuracy, on the other hand, signifies the performance of the model on a validation dataset that it has not seen during training. It is used to assess the generalization ability of the model. When compared to Training Accuracy, Validation Accuracy reveals how effectively the model generalizes to new data.

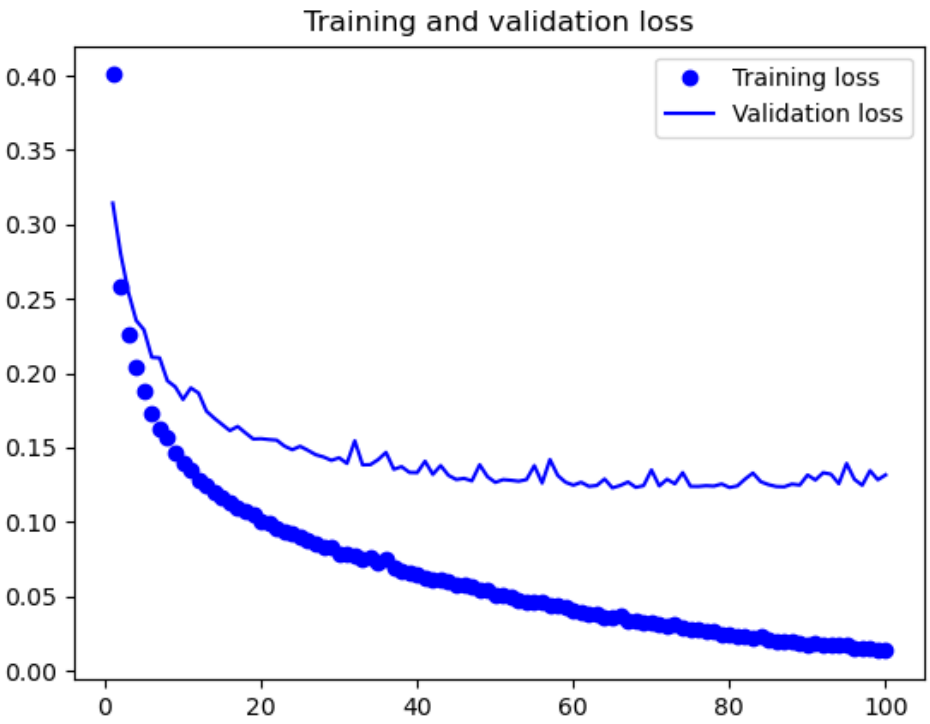

Figure 4.7: Training Validation Loss.

A training validation loss graph is shown in Figure 4.7. Training loss is a metric measuring the performance of the model on the training dataset. Training loss represents the difference between the actual values and the predicted values during the training process. A low training loss indicates that the model is performing well on the training dataset.

Validation loss is a metric measuring the performance of the model on a validation dataset that it has not seen during training. Validation loss is used to evaluate the generalization ability of the model. When compared to the training loss, validation loss shows how well the model generalizes to new data.

The precision, accuracy, recall and F1 score values obtained after the training are shown in Table 4.1.

Table 4.1: Evaluation Metrics.

Metrics	Value
Precision	0.96429
Accuracy	0.96196
Recall	0.96047
F1 Score	0.96238

Precision indicates the probability that the instances predicted as positive are indeed positive. A high precision value demonstrates the model's ability to reduce false positives.

Accuracy represents the ratio of correctly classified instances to the total number of instances. A high accuracy value indicates the overall good performance of the model.

Recall shows how many of the truly positive instances the model can detect. A high recall value demonstrates the model's ability to not miss true positives.

The F1 Score is the harmonic mean of precision and recall. It focuses on both precision and recall. A high F1 score indicates the model's ability to both reduce false positives and detect true positives.

4.3. Android Application

The Android application is developed in the Kotlin programming language using Android Studio. It requests access to the user's phone, and once granted, the user can install any desired APK file through the application. Additionally, the user can view all downloaded APK files on their phone and list the applications under the "Downloads" folder. All listed or user-uploaded APK files are sent to a service. This service, implemented using FAST API, facilitates integration with the model. FAST API is a modern, fast, and high-performance web framework for Python. It supports asynchronous programming and is specifically designed for developing APIs (Application Programming Interfaces).

With this service, the APK files uploaded from the application are received, unzipped, and extracted the "AndroidManifest.xml", "META-INF/CERT.RSA", "classes.dex" and "resources.arsc" features. The transformed features, represented as gray scale images. Before giving the images to the model, preporcessing is done with resizing and

normalization. After that, images sent as input to the model. The prediction results from the service are returned to the application. These prediction results are displayed to the user in the form of a score and class within the application.

4.3.1. Missing Files in APK

When testing with applications on a phone, it was observed that not all extracted features from some APK files were present. Files such as ”resources.arsc” or ”META-INF/CERT.RSA” could be missing. In this case, without a readable file, it could not be converted into an image. To address this issue, when such an APK file was received for service, a custom grayscale image Figure 4.8 was used as a representation of the missing feature.

In testing the data, if both files are missing, the test accuracy value is 0.4935. If the ”META-INF/CERT.RSA” file is missing, the test accuracy value is 0.9309; if the ”resources.arsc” file is missing, the test accuracy value is 0.7507. Finally, random test data was divided into four sections: one section with no missing files, one with no ”resources.arsc” file, one with no ”META-INF/CERT.RSA” file, and one with neither file. In this case, the test accuracy value is 0.7857.

Figure 4.8: Custom Gray Scale Image.

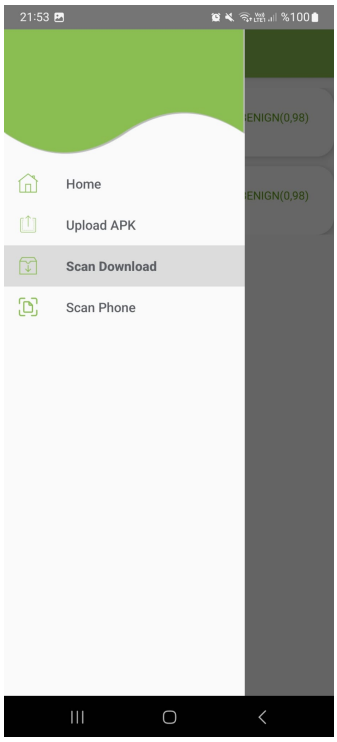

Figure 4.9: Android Application UI Screenshot (Sidebar) .

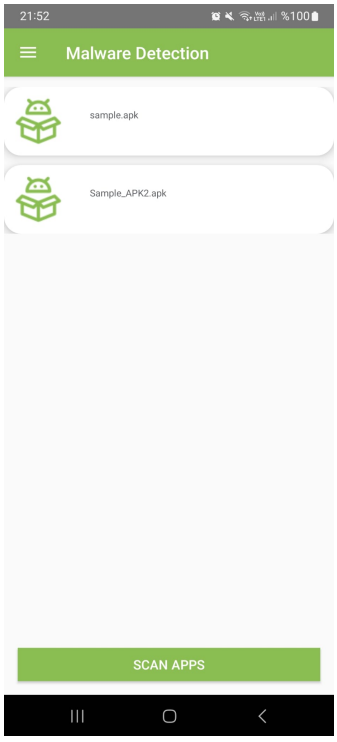

Figure 4.10: Android Application UI Screenshot (APKs).

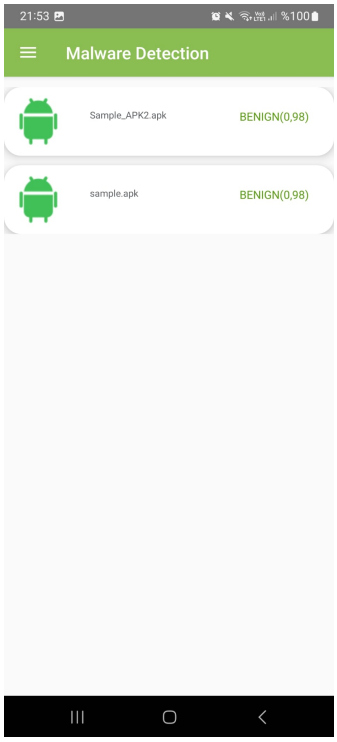

Figure 4.11: Android Application UI Screenshot (Results).

5. CONCLUSIONS

In conclusion, Image Recognition-Based Android Malware Detection emerges as a critical and imperative aspect in the realm of mobile security. As the prevalence of mobile devices continues to rise, ensuring the security of Android applications becomes increasingly challenging. The utilization of image recognition techniques in malware detection represents a proactive approach to identifying and mitigating potential threats. By leveraging this technology, the Android ecosystem can enhance its defenses against evolving malware, ultimately safeguarding user data and preserving the integrity of mobile devices. The ongoing development and implementation of innovative security measures underscore the importance of staying vigilant and adaptive in the face of emerging cybersecurity challenges in the mobile landscape.

As a result, this project successfully achieves the goal of image recognition-based Android malware detection. The number of applications in the dataset can be increased to improve the project. The model can be retrained by adding data from a different malware family. In addition, the model can be trained again to handle the missing file status in the APK, and the success of the model in this situation can be improved.

BIBLIOGRAPHY

[1] Manzhi Yang, Qiaoyan Wen, " Detecting Android Malware by Applying Classification Techniques on Images Patterns," 2017.

[2] Halil Murat Ünver, Khaled Bakour, "Android Malware Detection Based on Image-Based Features," 2019.

[3] Vikas Sihag, Surya Prakash, Gaurav Choudhary, Nicola Dragoni, and Ilsun You, "DIMDA: Deep Learning and Image-Based Malware Detection for Android," 2022.

CV

APPENDICES