Computers and Electrical Engineering 123 (2025) 110039

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

CNN-ViT synergy: An efficient Android malware detection approach through deep learning

Md. Shadman Wasif [a,*], Md. Palash Miah [a], Md. Shohrab Hossain [a], Mohammed J.F. Alenazi [b], Mohammed Atiquzzaman [c]

[a] Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, 1000, Bangladesh
[b] College of Computer and Information Sciences (CCIS), King Saud University, Riyadh, 11451, Saudi Arabia
[c] School of Computer Science, University of Oklahoma, Norman, 73019, United States of America

ARTICLE INFO

Keywords:
Android malware
Network traffic
Network flow
Deep learning
CNN
ViT

ABSTRACT

The surge in malicious Android applications poses a significant risk to global smartphone security which demands robust detection strategies that are both effective and efficient. Traditional malware detection methods often rely on complex feature sets that can slow down analysis and obscure key insights. To simplify malware detection, this study presents a novel approach by converting network traffic data into images, which are then analyzed using deep learning models. We introduce hybrid models that seamlessly integrate Convolutional Neural Networks (CNN) and Vision Transformers (ViT) to capitalize on their respective strengths in identifying malicious traffic. Notably, our method explores various image resolutions, finding that a 180x180 resolution optimizes detection accuracy without compromising much processing speed. The proposed model achieves a groundbreaking 99.61% multiclass accuracy rate which demonstrates the effectiveness in distinguishing between benign and malicious applications with high precision. This research not only sets a new standard in Android malware detection efficiency but also paves the way for future advancements in the application of deep learning for cybersecurity.

1. Introduction

Malware, designed to disrupt or damage devices, poses a significant threat in today's digital landscape, particularly on mobile devices due to their storage of sensitive personal data. More than 2.4 million applications are available on Google Play, according to recent reports [1,2]. The sheer volume of apps creates opportunities for hackers to exploit vulnerabilities. In the second quarter of 2023, Kaspersky Security Network detected 370,327 malicious installation packages, including 59,167 mobile banking Trojans and 1318 mobile ransomware Trojans [3]. These malicious installation packages are often distributed through untrustworthy channels like third-party websites and dubious app stores. Despite rigorous security measures on official platforms like Google Play, hackers sometimes bypass these checks by initially uploading benign apps that later receive malicious updates. Both new users and those who have previously downloaded the software may become infected as a result of this. Even when such apps are quickly removed, many users may have already downloaded them, highlighting the persistent challenge of securing mobile applications against malware.

In response to the sophisticated tactics employed by cyber attackers, a malware detection approach based on network traffic analysis emerges as a pivotal countermeasure. By continuously monitoring the app-generated network traffic for anomalies that

* Correspondence to: Bangladesh University of Engineering and Technology, Dhaka, 1000, Bangladesh.
E-mail address: shadman.wasif@gmail.com (M.S. Wasif).

https://doi.org/10.1016/j.compeleceng.2024.110039
Received 27 April 2024; Received in revised form 28 October 2024; Accepted 24 December 2024
Available online 6 January 2025

deviate from baseline patterns, this approach offers a dynamic and robust defense mechanism. It effectively counters the challenge posed by malware that activates post-installation, ensuring that even apps that pass initial security checks but later turn malicious can be detected and mitigated. This adds an essential layer of security that adapts to the evolving tactics of cyber threats, safeguarding sensitive user information against unauthorized access and exploitation.

While there have been numerous efforts with machine learning and deep learning to detect Android malware, there is still room for exploring faster and more optimized detection methods with efficient implementations. This need arises because cybercriminals are increasingly targeting Android devices due to the widespread popularity of the Android operating system [4]. Therefore, it is an ongoing area of development that demands the discovery of effective methods for detecting harmful software that contains malicious network traffic.

Numerous solutions in deep learning have focused on network traffic analysis. Many of the research studies have relied on raw packet capture files. These studies often involve altering and trimming the data packets to ensure uniform data sizes, which they believe can lead to improved accuracy. However, this practice of data modification can raise concerns about the data's validity and introduce bias. Our proposed approach stands out from earlier studies by focusing on improving accuracy by analyzing network traffic features instead of modifying the data itself. An efficient algorithm is employed to convert network traffic data into an image dataset, avoiding unnecessary data modifications. Moreover, our proposed models showed robust performance across different image resolutions, suggesting the adaptability and effectiveness of the approach in various scenarios.

This research contributes significantly in several areas: (i) it introduces a novel malware detection technique that identifies malware based on network traffic anomalies, (ii) it pioneers a creative approach for converting network traffic data into images, and (iii) it develops advanced deep learning models which utilize these images to improve Android malware detection.

Our proposed work addresses several key limitations in current Android malware detection methods. (a) Traditional methods often modify network traffic data through trimming or addition which can lead to bias and compromise data integrity. In contrast, our approach transforms network traffic data into an image dataset without any modifications. (b) Most research focuses on only CNNs to detect Android malware. Our study introduces ViTs, a largely unexplored strategy in Android malware detection with network traffic data, to the best of our knowledge. Furthermore, We have developed a sophisticated hybrid framework combining CNNs and ViTs, achieving a notable accuracy rate of 99.61%.

The rest of the paper is organized as follows: Section 2 explains relevant terms and concepts, while Section 3 offers a comprehensive list of previous research in the field. Section 4 describes the details of our proposed approach and the strengths of our approach, followed by the implementation in Section 5. In Section 6 the experimental results are presented. Finally, we conclude the paper in Section 7, where we offer our final remarks and conclusions.

2. Background

In our research, we utilize Vision Transformers (ViT) because they are particularly effective at processing image data by using self-attention mechanisms. This allows them to analyze relationships across the entire image simultaneously, which is beneficial for interpreting complex network traffic images. Most existing methods rely on Convolutional Neural Networks (CNNs). Most existing researches rely on Convolutional Neural Networks (CNNs). While CNN is still effective, it inherently focuses on local features due to their convolutional nature. Also, it is essential to identify global dependencies to detect complex malware patterns that might appear subtly across an entire network traffic image. To better understand the application of this paper, this section explains several key terminologies and concepts.

2.1. Malware analysis

Android malware detection can be categorized into three main strategies: static, dynamic, and hybrid analysis. Static analysis looks at an app's code and metadata without running the app. In contrast, dynamic analysis observes the app's behavior by running it in a secure environment and monitoring its activities, such as logs and network traffic data. Hybrid analysis combines these two approaches for a more thorough examination.

2.2. Network traffic

When classifying network traffic using machine learning, the goal is to organize the ongoing flow of data into distinct groups. Most prior research [5,6] in this field has broken down network traffic into 'flows' and 'sessions'. A flow represents collections of packets that share identical characteristics. A session is defined as a bidirectional traffic flow, where the source and destination IP addresses and ports are interchangeable. Different data packet formats and software tools for capturing those data are available in open source, offering a wide range of options for analyzing network traffic.

2.3. Convolutional neural network

Convolutional Neural Network (CNN) is a deep learning architecture specially designed for image processing and pattern recognition tasks [7]. The fundamental architecture of a CNN includes convolutional layers, pooling layers, and fully connected layers. Convolutional layers use filters to analyze input data and extract local features which allow the network to understand spatial patterns. Pooling layers then downsample the spatial dimensions, reducing computational complexity while preserving important features. Finally, fully connected layers process the spatial hierarchies and features extracted by earlier convolutional and pooling layers for classification or regression. CNNs are widely used for tasks where computers need to understand and work with visual data.

2.4. Vision transformer

The Vision Transformer (ViT) is a groundbreaking deep learning architecture designed for image recognition tasks, as pioneered in the paper [8], ViT relies on several key components to transform images into structured data for analysis. ViT processes images by breaking them down into smaller segments, known as patches, and then learns to identify the content of these patches. It assigns a role to each patch for classification, understands where each patch is concerning others, and uses a process called Multi-Head Attention to focus on various parts of the image at the same time. This helps ViT get a full understanding of both local and global features. To ensure efficient training, ViT employs techniques such as layer normalization and skip connection. ViT also incorporates a Multi-Layer Perceptron (MLP) to extract complex high-level features from the input image. This innovative architecture revolutionizes image recognition and analysis.

3. Literature review

In the field of Android malware detection, numerous studies have delved into the analysis of network traffic. However, it is noteworthy that the utilization of deep learning techniques, particularly image-based classification using Convolutional Neural Networks (CNN) or Vision Transformers (ViT), remains a relatively less-explored area.

Sihag et al. [5], proposed a novel framework named PICAndro, which utilized packet inspection of captured network traffic to enhance the accuracy and depth of malware detection. The research also compared the effectiveness of flow-based versus session-based network interactions for malware detection, showing that flow-based detection outperformed session-based approaches.

Gohari et al. [6] employed the CICAndMal2017 [9] dataset and extracted 86 network-flow features from each flow using CICFlowMeter [10]. They developed a deep learning model combining Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks to detect and classify Android malware. The researchers conducted extensive experiments across three scenarios (binary detection, category classification, and family characterization), providing detailed performance metrics such as precision and recall. However, the study mainly relied on static analysis.

Ding et al. [11], conducted their research using an APK dataset collected from the Drebin dataset and the Anzhi App Store. They introduced an innovative technique by visually representing the APK files of Android apps. They extracted Java bytecodes, divided the whole byte stream into subsequences, and generated images from the subsequences. They did not employ decompiling tools to analyze malware features, making it possible to detect encrypted malware. However, they achieved comparatively low accuracy.

Bakour et al. [12], introduced a new approach by extracting both local and global image-based features from APK files, which had not been previously utilized in the Android malware detection domain. This research also used static analysis only. They proposed a novel lightweight 1D-convolutional layers-based CNN model, DeepVisDroid, which outperformed classical CNN models in terms of classification accuracy and computational cost. However, this research also used static analysis only.

Abuthawabeh et al. [13] The study introduced a supervised-based model that improved the accuracy and depth of malware detection by utilizing conversation-level features. They employed an ensemble learning technique to select the most useful features for malware detection and categorization. However, more static features could be incorporated to further enhance malware detection.

Zulkifli et al. [14] suggested a dynamic malware detection technique based on decision tree algorithms emphasizing behavioral elements of the network. The authors used the Drebin [15] and Contagiodumpset [16] datasets for feature selection. However, The dataset used here is relatively small which can raise the concern of the model's effectiveness on unseen data. Moreover, the research acknowledges that the malware dataset does not run on actual mobile devices or emulators.

Kim et al. [17] proposed a novel deep learning-based system for Android malware detection called MAPAS, which focuses on analyzing API call graphs to identify malicious behaviors. The system used convolutional neural networks (CNN) for feature extraction but relies on a lightweight classifier for the actual detection. However, additional validation in real-world scenarios could be necessary to assess the practical performance of MAPAS.

Feng et al. [18] presented a two-layer deep learning approach for detecting Android malware. The first layer employed a fully connected neural network that utilized static features from Android apps. The output of the first layer was then input to the second layer. The second layer introduced a new method called CACNN (Cascading CNN and AutoEncoder) to classify benign or malicious applications through network traffic features. The proposed method demonstrated significant effectiveness in the binary and category classification of malware. However, the family classification accuracy of the method is quite low.

Shen et al. [19] introduced a model called SelAttConvLstm which integrates self-attention mechanisms to CNN and LSTM layers to improve the classification of Android malware. The result demonstrated that the model outperformed other deep learning models, such as CNN and CNN-LSTM. However, the results also indicated that smaller datasets led to poor classification accuracy. So, the performance of the proposed model is significantly influenced by the size of the dataset.

Alkahtani et al. [20] utilized several machine learning and deep learning algorithms for Android malware classification. The performance of these models was evaluated using two benchmark datasets. The results indicated that the support vector machine (SVM) algorithm achieved the highest accuracy of 100% on the CICAndMal2017 dataset. The deep learning models CNN and CNN-LSTM also performed well but the accuracy was lower than SVM. However, this perfect accuracy raises concerns about potential overfitting.

Zhang et al. [21] introduced a two-stage framework for early Android malware detection which focused particularly on encrypted traffic. The first stage used CNN to extract binary classification features while the second stage applied principal component analysis (PCA) to extract key features for multiclass malware classification. These features were combined with traffic payloads to create a

comprehensive set of classification features. These features were then processed by a cascaded deep forest classifier for multiclass malware classification. While this approach demonstrates strong performance, the approach is complex and computationally intensive.

Bakir et al. [22] presented a novel malware detection method called DroidEncoder, which utilized an auto-encoder structure for feature extraction. Their approach employed three different autoencoders (ANN-based, CNN-based, and VGG19-based) to extract features, which were then used to train multiple machine-learning algorithms. The result demonstrated the effectiveness of their method across various metrics which emphasized the importance of feature extraction in machine learning methods for malware detection.

Rehman et al. [23] proposed an efficient hybrid framework that combined both signature-based and heuristic-based analysis for detecting malware in Android apps. They analyzed mobile apps by extracting key files like manifest files and binaries. They used machine learning techniques, including Support Vector Machine (SVM), Decision Tree, W-J48, and K-nearest neighbors (KNN) for classification. Their findings showed that SVM was especially good at analyzing the binaries, while KNN performed better with manifest files.

The survey by Nawshin et al. [24] provided a comprehensive survey of the current landscape of mobile malware detection, focusing on the integration of federated learning (FL) and its implications for user privacy and security. The authors discussed conventional machine learning and deep learning models used for malware detection. They emphasized the challenges these models face, particularly in terms of privacy leakage when training on sensitive user data. However, although FL offers significant advantages in terms of privacy and real-time detection, it also faces challenges related to security threats such as data leakage, Byzantine attacks, data and model poisoning, etc. The survey provided a comprehensive analysis of mobile malware detection with traditional machine learning and deep learning approaches and federated learning.

3.1. Overall gap analysis

Overall Gap Analysis from the existing literature is summarized below:

- Some of the research [5,11] employed unique data preprocessing techniques to convert network traffic data into images for classification. However, these methods modify original data which may potentially lead to biased or partial classifications.
- Several existing works [5,6] reported remarkably high accuracy levels (e.g., 99.12% and 98.90% in multiclass classification respectively). However, such high accuracy can raise concerns about potential overfitting, where models perform exceptionally well on the training data but may fail to generalize well to new, unseen data.
- It has been found in several works [6,12] that they focused mainly on static analysis of network characteristics to detect malware on Android, which might miss the benefits of dynamic analysis. A combination of both could lead to a more effective detection system.
- Deep learning architectures such as CNN and CNN-LSTM were used in [6,12] which can be considered somewhat outdated for Android malware classification tasks. In contrast, our inclusion of ViT in Android malware detection showcases our commitment to advancing the state-of-the-art in malware detection.

The objective of our proposed approach is to address some of these gaps in Android malware detection through deep learning techniques.

4. Proposed methodology

The research approach used for detecting Android malware through network traffic data involves several processes. Each step is essential to the methodology and aids in the systematic analysis of network traffic to identify malicious Android applications. The proposed methodology is shown in Fig. 1.

4.1. Data acquisition

For this research, we utilized an Android Malware dataset to acquire the necessary network traffic data. The CICAndMal2017 dataset [9] that we used for our analysis is classified into five categories — Adware, Ransomware, Scareware, SMS Malware, and Benign. The CSV files used in the dataset are derived from raw pcap files, which have been processed to include essential network features utilizing the CIC-FlowMeterV3 tool [10]. This dataset thus offers a robust and extensive basis for our research. A significant imbalance is noticeable in the ratio of benign to malicious applications within the dataset. Training a classifier on this data could result in a biased classifier. However, We transformed the dataset into an image dataset and utilized data augmentation techniques [25] to address this issue of imbalance during our model's training, as detailed in Section 5.

4.2. Data preprocessing

During the data preprocessing phase, we carefully refined our dataset to improve the quality of our analysis. We selected the necessary features for our research, drawing on knowledge from existing literature [13,14]. From the original 80 features for each

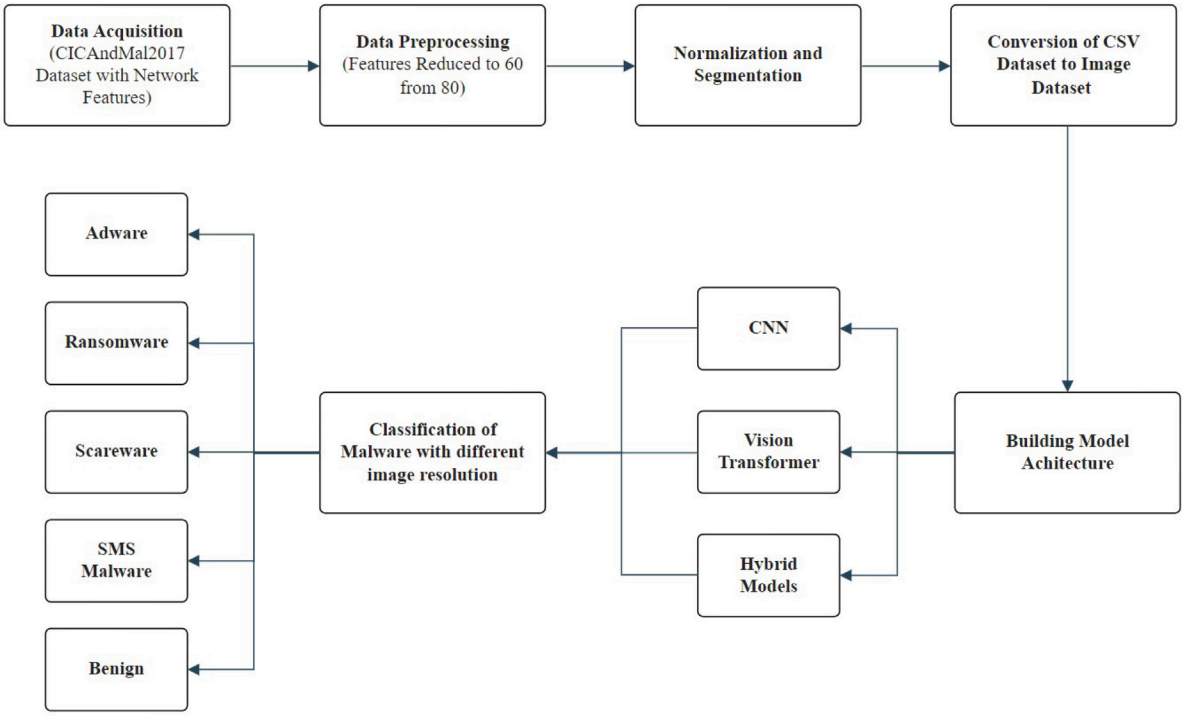


Fig. 1. Proposed methodology for android malware detection.

network traffic flow, we chose 60 that were most relevant for Android malware detection. This selection process involved several crucial steps of feature elimination, as mentioned below.

Static Features: Features such as fID (Flow ID), SrcID (Source ID), DesID (Destination ID), SrcPort (Source Port), DesPort (Destination Port), Protocol, and Timestamp were removed. Those features, although valuable in network analysis, were not relevant to Android malware detection.

Handling Missing or Malformed Data: We rigorously removed any records with incomplete or incorrect data, such as NaN (Not a number) or infinite values, to maintain the dataset's integrity and uniformity.

Constant Features: Features with constant values across the dataset, including Bwd PSH Flags, Fwd URG Flags, Bwd URG Flags, FIN Flag Count, PSH Flag Count, ECE Flag Count, Fwd Avg Bytes/Bulk, Fwd Avg Packets/Bulk, Fwd Avg Bulk Rate, Bwd Avg Bytes/Bulk, Bwd Avg Packets/Bulk, and Bwd Avg Bulk Rate, were removed because they offered no value in distinguishing between different data points for our models.

Duplicate Feature Removal: When we encountered duplicate features, we kept the original and removed the unnecessary copies. Notable examples include RST Flag Count, Fwd Header Length, Subflow Fwd Packets, Subflow Fwd Bytes, Subflow Bwd Packets, and Subflow Bwd Bytes.

These preprocessing steps were instrumental in reducing dimensionality, eliminating noise, and refining the dataset to facilitate the effective training of machine learning models for Android malware detection.

4.3. Data conversion

During the data conversion phase, we introduced an innovative method to transform network flow data into image format, as depicted in Fig. 2. The method includes 04 steps such as preprocessing, normalization, segmentation, and image generation. This allowed us to apply deep learning algorithms originally developed for image analysis. The transformation process involved the following key steps:

Normalization: We began by normalizing the preprocessed data using a min–max scaler, which adjusted the dataset values to a range between 0 (black) and 255 (white). This normalization was vital to ensure data consistency and to improve the clarity of the images created from the data. The normalization has been applied to the dataset using (1).

$$X_{\text{normalized}} = \left(\frac{X - X_{\min}}{X_{\max} - X_{\min}} \right) \times 255 \tag{1}$$

Segmentation: As the dataset has 60 features (N features), we divide the dataset into segments, each containing 180 samples (3N samples). We did this because we wanted to split these 180 data points into three equal parts, with each part containing 60

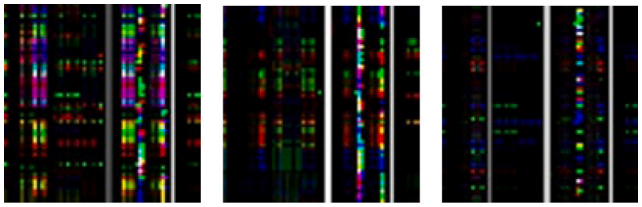
Input: RawTraffic.csv
Output: Set of N x N sized RGB images
Steps:

1. **Pre-processing:**
   (a) Load the data from RawTraffic.csv.
   (b) Select N features after necessary preprocessing.
2. **Normalization:**
   (a) Normalize the data in the range of 0 to 255.
3. **Segmentation:**
   (a) Divide the normalized data into segments, each containing 3N samples.
   (b) Discard the last segment if it contains fewer than 3N samples.
4. **RGB Image Generation:**
   (a) **foreach** segment **do**
   - Channel 1: Convert the first N samples in the segment into image matrix 1.
   - Channel 2: Convert the next N samples in the segment into image matrix 2.
   - Channel 3: Convert the last N samples in the segment into image matrix 3.

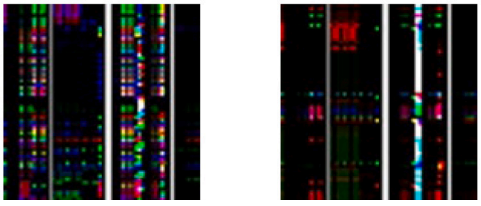
   (b) Map the three image matrices to the Red, Green, and Blue channels, respectively, to generate an RGB image of size N x N.
   End for

End

Fig. 2. Algorithm for image generation from the CSV of network traffic data.



(a) Adware (b) Ransomware (c) Scareware



(d) SMSMalware (e) Benign

Fig. 3. Generated images after the data conversion phase.

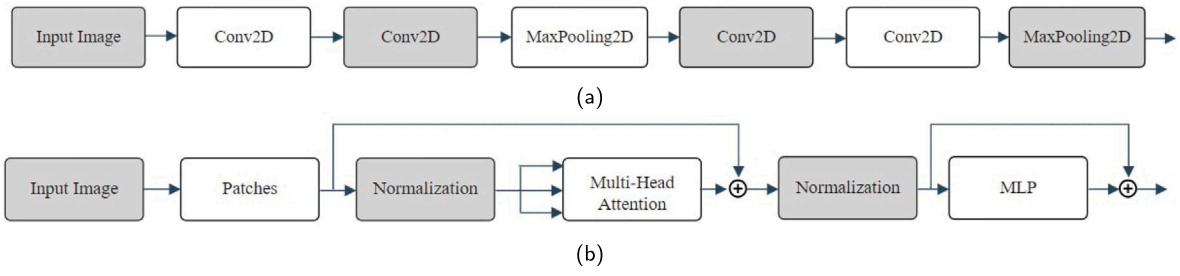
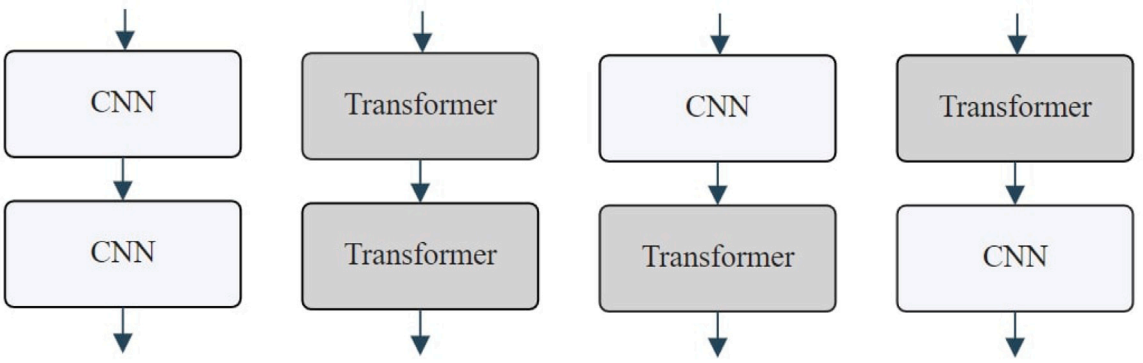
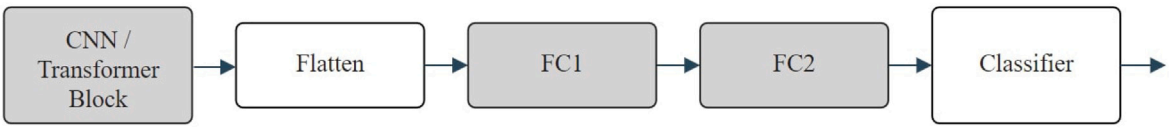


Fig. 4. The architectural designs of the (a) convolutional block framework and (b) transformer block framework.



(a) Model Structure (Block i, Block ii, Block iii and Block iv respectively).



(b) Classification head of the models.

Fig. 5. Implementation of the models.

samples. This setup allows us to create three separate image channels (RGB), where each image is represented by a square matrix with 60 rows and 60 columns, resembling a grid. This segmentation facilitated the efficient transformation of network data into an image format while retaining the data's chronological structure.

Image Generation: For every segment containing 180 samples, we generated an equivalent RGB image matrices with dimensions of 60 × 60. The samples were sequentially distributed to three separate channels with 60 samples assigned to each channel. The image matrices corresponding to the three channels were then mapped into the RGB channels of an image. This step was crucial for capturing the key features of the network flow data in a single image representation.

Disposal of segment: we removed any final segment from the CICAndMal2017 dataset [9] that contained less than 180 samples to maintain the integrity and uniformity of our newly created image dataset.

Overall, our data conversion process yielded a dataset comprising 14,534 individual image samples, 6697 benign, and 7837 malware samples. Fig. 3 presents one image sample from each category. Each image incorporates comprehensive network traffic data, which we then use as input for our deep-learning models.

4.4. Model architecture

This research explores various model architectures for Android malware detection, with a focus on two primary building blocks: the Convolutional Neural Network (CNN) block and the Transformer block. We also used hybrid models for Android Malware classification.

CNN Block: The CNN block contains two convolutional layers that use 10 filters in the initial Conv2D layers and 16 filters in the subsequent layers. This choice is based on common practices in literature for capturing detailed spatial features in image data efficiently. The CNN block also uses 3 × 3 kernel size, without adding padding or an activation function during these stages. Then,

Table 1
Data augmentation parameters and values.

Data augmentation parameter	Parameter value	Action
Width shifting	0.2	This randomly shifts the images horizontally up to 20%
Height shifting	0.2	This randomly shifts the images vertically up to 20%
Shear transformation	0.1	This stretches the image diagonally by up to 10%
Zooming	0.1	This randomly zooms in or out from the center by up to 10%
Horizontal flip	True	This randomly flips the images horizontally
Vertical flip	True	This randomly flips the images vertically

Table 2
Parameter specifications during the training of the models.

Parameter	Values
Conv.filters	10
Patch size	10
Dropouts rate	0.1, 0.5
Projection dimension	64
Activation function for CNN layers	LeakyReLU
Activation function for output layer	Softmax
Epochs	15
Batch size	32
Optimizer	AdamW
Error function	Categorical CrossEntropy
Learning rate	0.0001
Weight decay	0.000001

a max pooling layer with a 2×2 filter reduces the dimensions of the feature maps, which helps reduce computational complexity and the risk of overfitting The structure of the CNN model can be seen in Fig. 4a.

Transformer Block: Fig. 4b illustrates the Transformer block, where the input first goes through a normalization process to make the data even and balanced. Following this, it enters a multi-head attention layer with four separate attention units, each with a projection dimension of 64. This dimension is typically effective for encoding the information in each patch while maintaining a balance between model complexity and performance. The output from the multi-head attention is then merged back with the initial block input through a skip connection to enhance gradient flow and facilitate better information transfer. Following this, another layer of normalization is applied. The normalized data is then passed on to a multilayer perceptron (MLP) which consists of two fully connected layers, featuring 128 and 64 neurons respectively. The activation function used in each of these layers is the Gaussian Error Linear Unit (GELU). Finally, the output from these connected layers is added back to their input through another skip connection. This process ensures that the flow through the block maintains and refines the information from the input image.

4.5. Strengths of proposed method

There are several strengths of our proposed methodology:

- We carefully picked the features for our model by investigating previous studies [13,14] to ensure that we are focusing on the most important parts of network traffic. The original CICAndMal2017 dataset [9] contained more than 80 features and we were able to reduce the features to 60 only.
- We employed an efficient algorithm to convert the network traffic dataset into an image dataset. Unnecessary trimming or padding of bits was avoided, making our images both precise and efficient.
- Our proposed methodology includes Vision Transformers for Android malware detection by analyzing network traffic data. Unlike CNNs, Vision Transformers (ViTs) effectively capture long-range dependencies through self-attention mechanisms, enabling a thorough grasp of the global context. To the best of our knowledge, Android malware detection based on network traffic data using ViT has not been done before.
- We have created several hybrid models that combined the strengths of Convolutional Neural Networks and Vision Transformers. CNN excels at capturing local features, whereas Vision Transformers are known for capturing global context within images. To the best of our knowledge, this innovative approach of combining CNN and ViT has never been conducted for Android Malware classification.

5. Implementation

In our research, we have implemented eight different models to identify their effectiveness for Android malware detection. The implementation of the models can be found in the GitHub repository [26]. Fig. 5 illustrates the main designs of the different models we have implemented. These models can be categorized into four main groups based on their structures:

Convolutional Blocks Only: Model 1 features a single CNN block, focusing on spatial feature extraction. Model 2 has two CNN blocks for a deeper look at these features. Both models use block (i) as shown in Fig. 5a.

Transformer Blocks Only: Model 3 incorporates one Transformer block, emphasizing the significance of contextual understanding. Model 4 features two Transformer blocks which may enhance its ability to capture complex relationships. Both models use Block (ii) as shown in Fig. 5a.

Hybrid Structures: Models 5, 6, 7, and 8 adopt the hybrid architecture, combining both Convolutional and Transformer blocks. Models 5 and 7 include a combination as shown in Block iii and Block iv respectively. For Model 6, the arrangement starts with a Convolutional Block (Block i) followed by a Transformer Block (Block ii). In contrast, Model 8 begins with a Transformer Block (Block ii) and then adds a Convolutional Block (Block i).

Classification Head: Across all models, a consistent classification head was maintained as shown in Fig. 5b. The output from the CNN and Transformer blocks is then flattened as the input of a fully connected layer. The first fully connected layer (FC1) takes the flattened vector and performs a linear operation followed by a non-linear operation. The activation function used in the non-linear operation is LeakyReLU. The second fully connected layer (FC2) works just like the first one. It takes the output from FC1 and processes it further. Having multiple fully connected layers allows the network to learn more complex relationships in the data. The final output layer is also fully connected which serves as the classifier. As multi-class classification was conducted in this research, the final output layer used the activation function softmax. As all models use the same classification head to classify the data, it allows us to compare them fairly.

During the model training process, we deliberately chose to evaluate image resolutions at 60×60 (low), 120×120 (medium), and 180×180 (high) to cover a range of detail levels and observe how they affect model performance. It is to be noted that higher resolutions significantly increase computational resources. By limiting the resolution to 180×180, we aimed to strike a balance between model performance and computational resources. Additionally, preliminary experiments indicated that increasing the resolution above 180×180 offered marginal improvements in accuracy, which did not justify the additional computational costs.

We used various data augmentation techniques [25] to address the issue of data imbalance and to prevent overfitting, as shown in Table 1. These methods include shifting height and width, flipping horizontally and vertically, zooming, and shear transformation. They introduce variety to the training data through systematic transformations. The parameters, parameters value, and specific actions taken for each augmentation technique are comprehensively listed in Table 1. We split the dataset into a 70–30 train-test ratio, with 10,171 samples for training and 4363 for testing. This split ensures that the test set represents most of the dataset's variance. We trained the models for 15 epochs with a batch size of 32, a learning rate of 0.0001, and a weight decay of 0.000001.

We also made several key adjustments to our training methodology to improve our models' generalization and robustness. We tested different weight decay values, ranging from 0.01 to 0.0000001, and settled on 0.000001 to prevent overfitting. We also increased the dropout rate from 0.1 to 0.5 in the final MLP layer before classification to reduce dependency on specific features.

Initially, we conducted training over 20 epochs but observed overfitting in some models. Using early stopping, we consistently found optimal performance between 12 and 15 epochs. To address this and to establish a uniform baseline for comparison, we decided to standardize the training to 15 epochs across all models. This approach ensured that we maintained consistency and minimized overfitting while facilitating a fair comparison of model performance.

All the parameter specifications are listed in Table 2. These configurations were meticulously chosen to ensure effective model training, validation, and optimization.

6. Results

In this section, we will present the experimental results of our proposed models. We have recorded the accuracy of each model with different image resolutions. Confusion matrices have been generated for different image resolutions of each model. We have found in the literature that precision, recall, accuracy, and F1-score are four commonly used performance metrics. We have also conducted latency evaluation and percentile accuracy of the models. To get a thorough understanding of the results, we have graphically plotted our findings.

6.1. Performance metrics

We use Accuracy, Precision, Recall, and F1-score to evaluate the performance of our proposed models. Accuracy, Precision, Recall, and F1-score are the performance metrics used to evaluate our proposed models. It is important to note that True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) are essential to calculate performance metrics.

True Positive (TP): Correct Prediction, Actually Positive

True Negative (TN): Correct Prediction, Actually Negative

False Positive (FP): Incorrect Prediction, Actually Negative

False Negative (FN): Incorrect Prediction, Actually Positive

The performance metrics are computed using the following equations:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2}$$

Table 3
Accuracy for 60 × 60 image.

Model	A Model Structure	Accuracy
1.	CNN Block	84.36
2.	2 x CNN Block	86.95
3.	ViT Block	91.49
4.	2 x ViT Block	92.78
5.	CNN Block + ViT Block	89.27
6.	2 x CNN Block + 2 x ViT Block	96.28
7.	ViT Block + CNN Block	77.42
8.	2 x ViT Block + 2 x CNN Block	80.81

Table 4
Accuracy for 120 × 120 image.

Model	A Model Structure	Accuracy
1.	CNN Block	94.54
2.	2 x CNN Block	85.26
3.	ViT Block	99.10
4.	2 x ViT Block	98.78
5.	CNN Block + ViT Block	93.60
6.	2 x CNN Block + 2 x ViT Block	97.84
7.	ViT Block + CNN Block	84.57
8.	2 x ViT Block + 2 x CNN Block	95.64

Table 5
Accuracy for 180 × 180 image.

Model	A Model Structure	Accuracy
1.	CNN Block	95.39
2.	2 x CNN Block	87.71
3.	ViT Block	95.18
4.	2 x ViT Block	98.44
5.	CNN Block + ViT Block	93.94
6.	**2 x CNN Block + 2 x ViT Block**	**99.61**
7.	ViT Block + CNN Block	97.61
8.	2 x ViT Block + 2 x CNN Block	98.34

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3}$$

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

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5}$$

6.2. Experimental results

We have evaluated all of the models under the same parameters to get a comparative analysis of the models. We have observed the precision, recall, F1-score, and accuracy of each of the models. A comparative analysis of the experimental results will be presented in the following subsections.

6.2.1. Accuracy

Tables 3, 4, and 5 present the accuracy of the proposed models for different image resolutions.

Model 6 (2 x CNN block + 2 x ViT block) has performed the best in 60 × 60 resolution at 96.28% accuracy while model 7 has got the worst (77.42%). In 120 × 120, model 3 (ViT block) and model 4 (2 x ViT block) models both have achieved excellent accuracy of 99.10% and 98.78% respectively. In contrast, model 2 (2 x CNN block) and model 7 (ViT block + CNN block) have achieved the low accuracy of 85.26% and 84.57% respectively. It is to be noted that model 7 has also performed the worst in 60 × 60 image resolution. The model with two CNN blocks and two transformer blocks (model 6) for the 180 × 180 resolution has attained the highest accuracy, as indicated in Table 5, obtaining an excellent score of 99.61%. In 180 × 180, most of the models have achieved significantly high accuracy except model 2 (87.71%).

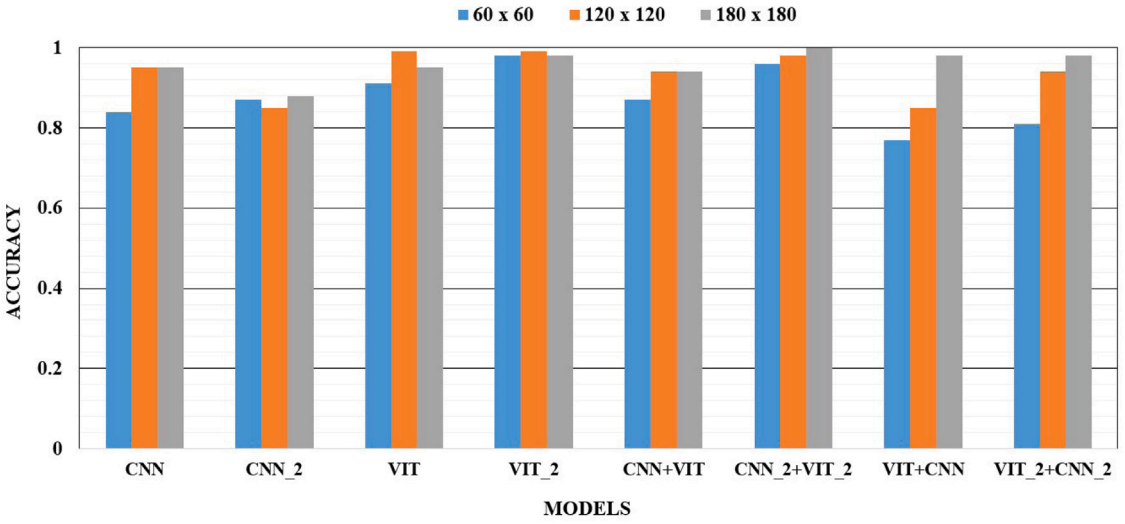


Fig. 6. Accuracy of the models for different image resolutions.

In Fig. 6, a comparative analysis of the models based on accuracy has been shown. It can be observed that ViT has outperformed CNN in terms of accuracy in most of the cases. Even in lower resolutions like 60 × 60, model 4 has achieved greater accuracy than models 1 & 2. It implies that transformer models can extract more valuable information even with lower-resolution images. However, we got the optimal results when we combined both CNN and ViT (model 6). Also, in most cases, the higher the image resolution, the higher the accuracy. But the trade-off here is the prediction speed.

The validation accuracy for each model for 15 epochs can be seen in Fig. 7. We have seen a fluctuation in several models. Usually, running more training epochs can make these results more stable. However, preliminary experiments indicated that additional training would not yield better generalization. Since most models converged and did their best within 15 epochs, there is no need to continue training them for longer. Moreover, our primary goal was to prevent overfitting, which can occur with excessive training. Furthermore, limiting the number of epochs also allowed us to use our computational resources more efficiently.

6.2.2. Confusion matrix

As we have proposed eight models, 24 confusion matrices are generated for three different image resolutions. We have presented six confusion matrices that have achieved the highest and the lowest accuracy for three different image resolutions in Figs. 8, 9, and 10.

A confusion matrix can help identify incorrect predictions visually. For image size 60 × 60, model 6 has performed the best, and model 7 has performed the worst. In model 6, most of the incorrect predictions belong to the scareware class. Model 6 has misclassified 131 samples of scareware as benign which has impacted the accuracy. Model 7 fails to correctly predict a single scareware sample. Both scareware and SMSMalware are misclassified as benign in model 7.

For image size 120 × 120, model 3 achieved the highest accuracy, and model 7 achieved the lowest accuracy. Model 3 mostly misclassified scareware and SMSMalware as benign respectively whereas model 7 incorrectly predicted 617 scareware samples as benign.

For image size 180 × 180, model 6 achieved an extraordinary accuracy of 99.61% whereas model 2 achieved the lowest accuracy of all the models (87.71%). Model 2 predicted 448 benign samples as scareware in 180 × 180.

From the confusion matrices, it can be concluded that the models have failed to classify mostly between scareware and benign samples. Scareware often generates network traffic that is similar to benign applications. Moreover, Scareware might mimic benign application behaviors to avoid detection, such as displaying legitimate-looking warnings or prompts that do not differ significantly from those of regular apps. Furthermore, this could be due to identical network flows shared by both classes, causing the images to look similar, as shown in Fig. 3.

Adversarial training to expose the model to tricky cases, eliminating problematic features that lead to similar network flows, and integrating hybrid analysis techniques that combine static and dynamic analysis can be used to better distinguish between scareware and benign samples.

6.2.3. Precision, recall and f1-score

The performance metric accuracy may be misleading when an imbalanced dataset is used. In such cases, precision, recall, and f1-score may offer a detailed and deeper insight into classification performance.

Precision provides insights into the accuracy of a model's positive predictions. A model with high precision means that it got significantly more relevant results than irrelevant ones. Recall is another performance metric that measures the number of true

Fig. 7. Validation accuracy graph of the models for 15 epochs.

positive predictions relative to an actual number of positive ones. A high recall means that the false negatives are low. There is frequently a trade-off between recall and precision. Recall may decrease with increased precision and vice versa. To resolve this issue,

Fig. 8. Best and Worst confusion matrices for 60 × 60.

Fig. 9. Best and Worst confusion matrices for 120 × 120.

Fig. 10. Best and Worst confusion matrices for 180 × 180.

an f1-score can be used which combines recall and precision. This metric provides a more complete picture of model performance and is resistant to extreme values, particularly in cases where the distribution of classes is not uniform.

Table 6 indicates that the models generally perform well as many Average Precision, Average Recall, and Average F1 scores are close or equal to one. This implies high accuracy and low false positive/negative rates. Based on precision, recall, and f1-score, Models 3 and 6 perform the best, and Model 7 performs worst overall.

It is interesting to note that performance does not always improve with additional blocks of CNN or ViT. For example, Model 1 which has 1 CNN block outperformed Model 2 which has 2 CNN blocks. Model 3 and Model 4 have achieved significant scores in all metrics and resolutions. Although the best of all the models seems to be a hybrid one, model 6. However, another hybrid one, model 7 underperformed in terms of recall, precision, and f1-score in 60 × 60 and 120 × 120. This implies that not all hybrid models are well-suited for Android malware classification.

6.2.4. Latency evaluation

We have evaluated the latency of our models with the sample of the scareware class. We have recorded the time taken for the models to make a prediction. All the models predicted correctly. In Fig. 11, a comparative analysis of the models based on latency has been shown.

Model 2 (2 x CNN block) had the lowest latency in 180 × 180 resolution with 386.45 ms and Model 4 (2 ViT blocks) had the highest latency in 60 × 60 resolution with 1103.66 ms.

Model 2 was the fastest across all resolutions, making it the most time-efficient. Models with ViT blocks, like Model 3 and Model 4, had significantly higher latencies, especially at lower resolutions. This may be due to the complex processing in Vision

Table 6
Results of experiment.

Model	Res	Precision	Recall	F1-Score
Model 1	60	0.89	0.87	0.85
	120	0.95	0.96	0.95
	180	0.95	0.97	0.96
Model 2	60	0.92	0.80	0.84
	120	0.90	0.89	0.87
	180	0.91	0.91	0.90
Model 3	60	0.96	0.87	0.90
	120	0.99	0.99	0.99
	180	0.98	0.93	0.95
Model 4	60	0.95	0.90	0.92
	120	0.99	0.98	0.98
	180	0.99	0.98	0.98
Model 5	60	0.91	0.88	0.88
	120	0.97	0.91	0.93
	180	0.98	0.91	0.93
Model 6	60	0.98	0.95	0.96
	120	0.99	0.96	0.98
	180	**0.99**	**0.99**	**0.99**
Model 7	60	0.69	0.66	0.66
	120	0.94	0.79	0.78
	180	0.99	0.95	0.97
Model 8	60	0.72	0.73	0.72
	120	0.97	0.93	0.95
	180	0.97	0.99	0.98

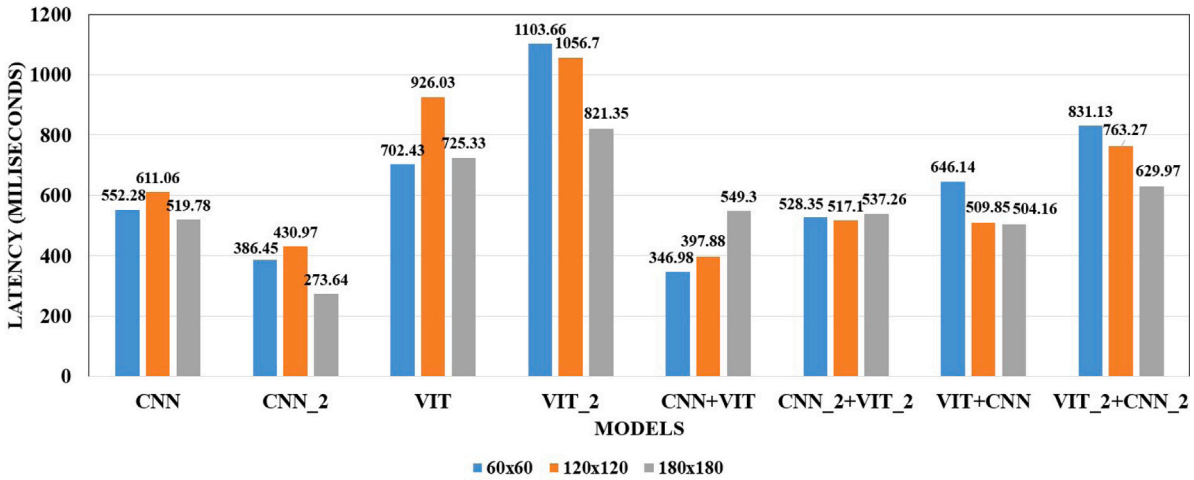


Fig. 11. Latency evaluation of the models.

Transformers. Hybrid models, such as Models 5, 6, 7, and 8, showed varied latencies. This suggests that combining CNN and ViT blocks can balance processing times. For instance, Model 6 (2 CNN blocks + 2 ViT blocks) had lower latency than pure ViT models but higher than CNN-only models.

From the latency evaluation, it can be concluded that models with only CNN blocks tend to have lower latency, while models incorporating only ViT blocks tend to have higher latency. However, the hybrid model 6 has shown promising results, offering a good balance between complexity and speed.

6.2.5. Percentile accuracy

We evaluated the accuracy of our models at the 95th and 99th percentiles. However, we did not include the percentile for all models due to space constraints. We presented the result only for the highest-performing model, Model 6 in Table 7.

Model 6 shows excellent accuracy for all malware types and image resolutions at the 95th percentile. For Adware, Ransomware, and SMSmalware, the model achieves 99.99% accuracy at 180 × 180 resolution. This highlights its strong ability to identify these malware types. For benign applications, the accuracy ranges from 99.52% to 99.97% which shows the model's effectiveness in

Table 7
Accuracy rates of model 6 at 95th and 99th percentile.

	95th Percentile			99th Percentile		
	60 × 60	**120 × 120**	**180 × 180**	**60 × 60**	**120 × 120**	**180 × 180**
Adware	99.99	99.99	99.99	99.99	99.99	99.99
Benign	99.52	99.87	99.97	99.66	99.90	99.98
Ransomware	99.93	99.99	99.99	99.98	99.99	99.99
Scareware	94.88	95.11	99.98	97.84	98.10	99.98
SMSmalware	99.87	99.98	99.99	99.95	99.99	99.99

Table 8
Comparison of android malware detection studies (Dataset: CICAndMal2017)

Author	Year	Technique	Accuracy	Precision	Recall	F1
Feng et al. [18]	2020	CNN & AutoEncoder (CACNN)	98.2	98.4	96.4	–
Sihag et al. [5]	2021	CNN	98.56	98.51	98.46	98.49
Shen et al. [19]	2022	CNN & LSTM (SelAttConvLstm)	99.12	99.18	99.05	99.11
Alkahtani et al. [20]	2022	CNN-LSTM	95.07	97.16	93.63	95.53
Zhang et al. [21]	2023	CNN	98.11	99.55	98.02	98.78
Proposed work	2023	CNN & ViT	99.61	99.30	99.08	99.19

Table 9
Comparison of android malware detection studies with statistical significance tests.

Author	Accuracy	t-statistics	*p*-value (t-stat)	Wilcoxon statistic	*p*-value (Wilcoxon)	Significant Difference
Feng et al. [18]	98.2	4.3374	0.0019	2.0	0.0059	Yes
Sihag et al. [5]	98.56	2.6920	0.0247	6.0	0.0273	Yes
Shen et al. [19]	99.12	0.1325	0.8975	22.0	0.6250	No
Alkahtani et al. [20]	95.07	18.6429	0.0	0.0	0.0020	Yes
Zhang et al. [21]	98.11	4.7487	0.0010	2.0	0.0059	Yes

distinguishing benign traffic. Scareware detection is slightly lower at lower resolutions, with 94.88% accuracy for 60 × 60 and 95.11% for 120 × 120. However, it improves to 99.98% at 180 × 180 resolution which indicates better performance with higher resolution images.

At the 99th percentile, the accuracy is similarly high. The model maintains 99.99% accuracy for Adware, Ransomware, and SMSmalware at 180 × 180 resolution. For benign applications, accuracy improves slightly, ranging from 99.66% to 99.98%. Scareware detection also improves at higher resolutions, with 97.84% accuracy at 60 × 60, 98.10% at 120 × 120, and 99.98% at 180 × 180 which shows that the model benefits from higher resolution images.

The 95th percentile shows that Model 6 performs very well for most malware types, with minor drops in accuracy for Scareware and Benign applications at lower resolutions. At the 99th percentile, the accuracy is slightly higher.

Overall, the results show that Model 6 consistently achieves high accuracy for various malware types which demonstrates its robustness and effectiveness in detecting malware. Model 6 is especially effective at higher resolutions, making it good at identifying both malware and benign applications.

6.2.6. Comparative analysis

Table 8 presents a comparison of the performance of our suggested method against earlier research. It demonstrates the effectiveness of our approach in detecting recent malware through network traffic analysis.

We also conducted statistical tests to validate the performance of our hybrid model (Model 6). We compared our model's accuracy with several studies mentioned in the literature reviews. We have used the one-sample t-test and the Wilcoxon signed-rank test [27] to determine if the differences in performance are statistically significant. We compared the accuracy from ten runs of our model with the reported accuracies from other research.

t-statistic: The t-statistic measures the difference between the sample mean and the population mean (or a fixed value) in terms of the standard error of the sample mean. It is calculated using (6).

$$t = \frac{\bar{X} - \mu}{\frac{s}{\sqrt{n}}} \tag{6}$$

- $\bar{X}$ is the sample mean
- μ is the population mean
- s is the standard deviation of the sample
- n is the sample size

The p-value associated with the t-statistic shows whether the difference between our model and the literature models is statistically significant.

- If t > 0 and p-value < 0.05, the model's mean accuracy is significantly higher than the literature value
- If t ≤ 0 and p-value < 0.05, the model's mean accuracy is significantly lower than the literature value.
- If p-value ≥ 0.05 there is no significant difference.

Wilcoxon statistic: The Wilcoxon statistic measures the sum of the ranks of the differences between paired observations which is calculated using (7).

$$W = \min\left(\sum_{+} R_i, \sum_{-} R_i\right) \tag{7}$$

- $\sum_{+} R_i$ is the sum of ranks of the positive differences.
- $\sum_{-} R_i$ is the sum of ranks of the negative differences.

The p-value associated with the Wilcoxon statistic helps to determine if the observed differences are statistically significant.

- If p-value < 0.05, the model's median accuracy is significantly different from the literature value.
- If p-value ≥ 0.05 there is no significant difference between the model's median accuracy and the literature value

Table 9 presents the results of the significance tests.

The t-test shows significant differences in model performance ($p < 0.05$) for Feng et al. [18], Sihag et al. [5], Alkahtani et al. [20], and Zhang et al. [21]. This indicates that our hybrid model performs significantly better than these models. The Wilcoxon signed-rank test also shows significant differences ($p < 0.05$) for the same literature accuracies, further confirming the robustness of our model's performance.

However, the results from Shen et al. [19] did not show a significant difference in performance. While our model performs better, the improvement is not statistically significant at the 95% confidence level.

6.3. Discussion

The experimental results from the proposed models demonstrate the impact of model architecture and image resolution on the performance of Android malware classification. Based on the experimental results from our proposed models, here are the key findings:

- In most cases, vision transformers (ViT) have outperformed traditional convolutional neural networks (CNN) for Android malware classification. This could suggest that transformers may be more capable of extracting valuable information from images, even at lower resolutions. This is mainly because ViTs and CNNs handle images differently. CNNs look at images piece by piece, focusing on small areas at a time. However, ViTs look at the whole image all at once from the first layer, so they are better at understanding how different parts of an image relate to each other.
- In terms of accuracy, a hybrid model, model 6, which contains two CNN blocks followed by two ViT blocks, stands out by achieving excellent accuracy at 180 × 180 resolution. This model has also achieved significant accuracy in lower resolutions as well, which indicates its robustness across different image resolutions.
- In contrast, another hybrid model, model 7, has performed the worst at 60 × 60 and 120 × 120 resolution and achieved the lowest accuracy among all the models for different image resolutions. This indicates that all hybrid approaches may not be equally effective for Android Malware detection.
- The precision, recall, and F1-scores across models are generally high which indicates low rates of false positives and negatives. This is crucial for malware detection because errors in malware detection can lead to significant consequences.
- The models struggled to distinguish between scareware and benign samples as reflected in the confusion matrices in the result section. Both may have similar frequencies and types of network requests, making it challenging for the model to distinguish between them based on traffic data alone.
- The latency evaluation shows a trade-off between model complexity and processing speed. Models with only CNN blocks have lower latency, while models with only ViT blocks have higher latency. Model 6 (2 CNN blocks + 2 ViT blocks) has lower latency than pure ViT models (Model 4) but higher than CNN-only models (Model 2), balancing complexity and speed.
- The 99th and 95th percentile accuracy rates indicate that Model 6 is highly effective in classifying different types of malware, especially at higher resolutions. The 99th percentile accuracy is generally equal to or slightly higher than the 95th percentile rates. The high accuracy rates at both percentiles demonstrate the model's consistent and reliable performance.
- The significance test results confirm that our model's accuracy is not only higher but also statistically significant to several well-known models in the literature. The hybrid model's ability to combine local and global features using CNN and ViT makes it a strong method for Android malware detection.
- In most cases, the higher the image resolution, the higher the accuracy of the models. However, this process can be quite time-consuming, especially for hybrid models.

7. Conclusion and future work

We introduced a novel approach for Android malware detection by transforming network traffic data into images and employing advanced deep learning models, including Convolutional Neural Networks (CNN), Vision Transformers (ViT), and hybrid models integrating both architectures. We have shown that image-based analysis of network features is highly effective in distinguishing between malicious and benign applications. Our results indicate that a hybrid model combining both CNN and ViT can produce better results for Android malware detection. The proposed detection method has demonstrated its potential to offer an accurate means of identifying harmful applications. Though our primary focus has been on network traffic features, we acknowledge the possibility of further improving the classification by integrating other static features of Android. Also, the model's performance can be validated across diverse datasets in the future which will ensure its effectiveness with different network traffic data. Furthermore, dynamic analysis for Android malware detection using network traffic features is a promising direction for future research.

CRediT authorship contribution statement

Md. Shadman Wasif: Conceptualization, Methodology, Software, Writing – original draft. **Md. Palash Miah:** Formal analysis, Methodology, Investigation, Writing – original draft. **Md. Shohrab Hossain:** Project administration, Data curation, Resources, Writing – review & editing. **Mohammed J.F. Alenazi:** Data curation, Validation, Writing – review & editing. **Mohammed Atiquzzaman:** Supervision, Validation, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The authors extend their appreciation to Researcher Supporting Project number (RSPD2025R582), King Saud University, Riyadh, Saudi Arabia.

Data availability

Data will be made available on request.

References

[1] Curry D. Google play store statistics (2024). 2024, https://www.businessofapps.com/data/google-play-statistics/. (Accessed 13 August 2024).
[2] Ceci L. Number of available applications in the google play store from december 2009 to december 2023. 2023, https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/. (Accessed 13 August 2024).
[3] Kivva A. It threat evolution in q2 2023. mobile statistics. 2023, https://securelist.com/it-threat-evolution-q2-2023-mobile-statistics/110427/. (Accessed 13 August 2024).
[4] Qiu J, Zhang J, Luo W, Pan L, Nepal S, Xiang Y. A survey of android malware detection with deep neural models. ACM Computing Surveys 2020;53(6):1–36.
[5] Sihag V, Choudhary G, Vardhan M, Singh P, Seo JT. Picandro: packet inspection-based android malware detection. Security and Communication Networks 2021;2021(1):9099476.
[6] Gohari M, Hashemi S, Abdi L. Android malware detection and classification based on network traffic using deep learning. In: 2021 7th International Conference on Web Research (ICWR). IEEE; 2021, p. 71–7.
[7] LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE 1998;86(11):2278–324.
[8] Dosovitskiy A, Beyer L, Kolesnikov A, Weissenborn D, Zhai X, Unterthiner T, Dehghani M, Minderer M, Heigold G, Gelly S, et al. An image is worth 16×16 words: Transformers for image recognition at scale. 2020, arXiv preprint arXiv:2010.11929.
[9] Lashkari AH, Kadir AFA, Taheri L, Ghorbani AA. Toward developing a systematic approach to generate benchmark android malware datasets and classification. In: 2018 International Carnahan Conference on Security Technology (ICCST). IEEE; 2018, p. 1–7.
[10] Lashkari AH, Gil GD, Mamun MSI, Ghorbani AA. Characterization of tor traffic using time-based features. In: International Conference on Information Systems Security and Privacy, Vol. 2. SciTePress; 2017, p. 253–62.
[11] Ding Y, Zhang X, Hu J, Xu W. Android malware detection method based on bytecode image. Journal of Ambient Intelligence and Humanized Computing 2020;1–10.
[12] Bakour K, Ünver HM. DeepVisDroid: Android malware detection by hybridizing image-based features with deep learning techniques. Neural Computing and Applications 2021;33:11499–516.
[13] Abuthawabeh MKA, Mahmoud KW. Android malware detection and categorization based on conversation-level network traffic features. In: 2019 International Arab Conference on Information Technology (ACIT). IEEE; 2019, p. 42–7.
[14] Zulkifli A, Hamid IRA, Shah WM, Abdullah Z. Android malware detection based on network traffic using decision tree algorithm. In: Recent advances on soft computing and data mining: Proceedings of the third international conference on soft computing and data mining (sCDM 2018), johor, Malaysia, February 06–07, 2018. Springer; 2018, p. 485–94.
[15] Arp D, Spreitzenbarth M, Hubner M, Gascon H, Rieck K, Siemens C. Drebin: effective and explainable detection of android malware in your pocket. In: Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS). San Diego, CA, USA; 2014.
[16] Dump CM. Contagio dump set. 2013, https://contagiodump.blogspot.com/. (Accessed 13 August 2024).
[17] Kim J, Ban Y, Ko E, Cho H, Yi JH. MAPAS: A practical deep learning-based android malware detection system. International Journal of Information Security 2022;21:725–38.

[18] Feng J, Shen L, Chen Z, Wang Y, Li H. A two-layer deep learning method for Android malware detection using network traffic. IEEE Access 2020;8:125786–96.
[19] Shen L, Feng J, Chen Z, Sun Z, Liang D, Li H, Wang Y. Self-attention based convolutional-LSTM for android malware detection using network traffics grayscale image. Applied Intelligence 2023;53:683–705.
[20] Alkahtani H, Aldhyani TH. Artificial intelligence algorithms for malware detection in android-operated mobile devices. Sensors 2022;22(6):2268.
[21] Zhang X, Wang J, Xu J, Gu C. Detection of android malware based on deep forest and feature enhancement. IEEE Access 2023;11:29344–59.
[22] Bakır H, Bakır R. DroidEncoder: Malware detection using auto-encoder based feature extractor and machine learning algorithms. Computers and Electrical Engineering 2023;110:108804.
[23] Rehman Z-U, Khan SN, Muhammad K, Lee JW, Lv Z, Baik SW, Shah PA, Awan K, Mehmood I. Machine learning-assisted signature and heuristic-based detection of malwares in Android devices. Computers & Electrical Engineering 2018;69:828–41.
[24] Nawshin F, Gad R, Unal D, Al-Ali AK, Suganthan PN. Malware detection for mobile computing using secure and privacy-preserving machine learning approaches: A comprehensive survey. Computers and Electrical Engineering 2024;117:109233.
[25] Maharana K, Mondal S, Nemade B. A review: Data pre-processing and data augmentation techniques. Global Transitions Proceedings 2022;3(1):91–9.
[26] Wasif MS, Miah MP, Hossain MS. CNN-vit synergy: An approach to Android malware detection based on network traffic. 2023, https://github.com/shadman17/network_security_project_android_malware_detection. (Accessed 13 August 2023).
[27] Kitani M, Murakami H. One-sample location test based on the sign and wilcoxon signed-rank tests. Journal of Statistical Computation and Simulation 2022;92(3):610–22.

Md. Shadman Wasif is currently pursuing a Master of Science (M.Sc.) degree in Computer Science and Engineering from Bangladesh University of Engineering and Technology (BUET). Prior to this, he earned a Bachelor of Science (B.Sc.) degree in Computer Science and Engineering from the Military Institute of Science and Technology (MIST), where he established a strong academic foundation. His research interests focus on important and fast-growing areas, such as Cyber Security, Network Security, Software Engineering, Internet of Things (IoT), Machine Learning, Deep Learning etc.

Md. Palash Miah received the B.Sc. degree in Computer Science and Engineering from the Daffodil International University, Dhaka in 2022. He is currently conducting his M.Sc. in Computer Science and Engineering from Bangladesh University of Engineering and Technology, Dhaka. His current research interests include Network Security, Machine Learning, Natural Language Processing, Deep learning etc.

Md. Shohrab Hossain received his B.Sc. and M.Sc. in Computer Science and Engineering from Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh in the year 2003 and 2007, respectively. He obtained his Ph.D. degree from the School of Computer Science at the University of Oklahoma, Norman, OK, USA in December, 2012. During his Ph.D., he worked under NASA funded projects related to survivability, scalability and security of space networks. He is currently serving as a Professor in the Department of Computer Science and Engineering at Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh. His research interests include Mobile malware detections, cyber security, security in Software defined networking (SDN), security of mobile and ad hoc networks, and Internet of Things. He has published more than 100 technical research papers in leading journals and conferences including Journal of Computers & Security, IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Mobile Computing, Journal of Computer Communications, Computer Networks, IEEE Access, Journal of Network and Computer Applications, Journal of Telecommunication Systems, Wireless Personal Communication, PLOS ONE, IEEE GLOBECOM, IEEE ICC, IEEE MILCOM, IEEE HPSR, IEEE HPCC, etc. He has been serving as the TPC member of IEEE GLOBECOM, IEEE ICC, IEEE VTC, Wireless Personal Communication, (Springer), Journal of Network and Computer Applications (Elsevier), IEEE Wireless Communications.

Mohammed J.F. Alenazi earned his B.S., M.S., and Ph.D. degrees in computer engineering from the University of Kansas in 2010, 2012, and 2015, respectively. He is a Professor in computer engineering at King Saud University and a reviewer for several international journals (Senior Member, IEEE). His research interests span cybersecurity, focusing on network security, encryption, and vulnerability analysis, as well as machine learning, where he applies AI to enhance network security and performance. He also works on the design and analysis of resilient networks, network routing, and mobile ad hoc network (MANET) protocols. A member of ACM, his work contributes to the intersection of cybersecurity and machine learning for developing adaptive, threat-resistant systems.

Mohammed Atiquzzaman obtained his M.S. and Ph.D. in Electrical Engineering and Electronics from the University of Manchester (UK). He is currently holds the Edith Kinney Gaylord Presidential professorship in the School of Computer Science at the University of Oklahoma, and is a senior member of IEEE. Dr. Atiquzzaman is the Editor-in-Chief of Journal of Networks and Computer Applications, founding Editor-in-Chief of Vehicular Communications and has served/serving on the editorial boards of IEEE Communications Magazine, International Journal on Wireless and Optical Communications, Real Time Imaging journal, Journal of Communication Systems, Communication Networks and Distributed Systems and Journal of Sensor Networks. He also guest edited 12 special issues in various journals. Also, he has served as editor of IEEE Transactions on Mobile Computing and IEEE Journal on Selected Areas in Communications. He has served as co-chair of IEEE High Performance Switching and Routing Symposium (2011 and 2003) and has served as symposium co-chairs for IEEE Globecom (2006, 2007, 2014) and IEEE ICC (2007, 2009, 2011, 2012) conferences. He co-chaired ChinaComm (2008), and SPIE Next-Generation Communication and Sensor Networks (2006) and the SPIE Quality of Service over Next Generation Data Networks conferences (2001, 2002, 2003, 2005). He was the panels co-chair of INFOCOM05, and

is/has been in the program committee of numerous conferences such as INFOCOM, ICCCN, and Local Computer Networks. He serves on the review panels of funding agencies such as the National Science Foundation and National Research Council (Canada) and Australian Research Council (Australia). In recognition of his contribution to NASA research, he received the NASA Group Achievement Award for outstanding work to further NASA Glenn Research Centers effort in the area of Advanced Communications/Air Traffic Managements Fiber Optic Signal Distribution for Aeronautical Communications project. He is the co-author of the book Performance of TCP/IP over ATM networks and has over 300 refereed publications which are accessible at www.cs.ou.edu/ãtiq. His research interests are in communications switching, transport protocols, wireless and mobile networks, ad-hoc networks, satellite networks, Quality of Service, and optical communications. His research has been funded by National Science Foundation (NSF), National Aeronautics and Space Administration (NASA), U.S. Air Force, Cisco, Honeywell, Oklahoma Department of Transportation, Oklahoma Highway Safety Office through grants totaling over $7M.