*Computer Systems Science & Engineering*
DOI: 10.32604/csse.2023.033420
***Article***

Tech Science Press

# Applying Wide & Deep Learning Model for Android Malware Classification

**Le Duc Thuan[1,2,*], Pham Van Huong[2], Hoang Van Hiep[1] and Nguyen Kim Khanh[1]**

[1]Ha Noi University of Science and Technology, Ha Noi, 100000, Viet Nam
[2]Academy of Cryptography Techniques, Ha Noi, 100000, Viet Nam
*Corresponding Author: Le Duc Thuan. Email: thuanld@actvn.edu.vn
Received: 16 June 2022; Accepted: 26 July 2022

**Abstract:** Android malware has exploded in popularity in recent years, due to the platform's dominance of the mobile market. With the advancement of deep learning technology, numerous deep learning-based works have been proposed for the classification of Android malware. Deep learning technology is designed to handle a large amount of raw and continuous data, such as image content data. However, it is incompatible with discrete features, i.e., features gathered from multiple sources. Furthermore, if the feature set is already well-extracted and sparsely distributed, this technology is less effective than traditional machine learning. On the other hand, a wide learning model can expand the feature set to enhance the classification accuracy. To maximize the benefits of both methods, this study proposes combining the components of deep learning based on multi-branch CNNs (Convolutional Network Neural) with wide learning method. The feature set is evaluated and dynamically partitioned according to its meaning and generalizability to subsets when used as input to the model's wide or deep component. The proposed model, partition, and feature set quality are all evaluated using the K-fold cross validation method on a composite dataset with three types of features: API, permission, and raw image. The accuracy with Wide and Deep CNN (WDCNN) model is 98.64%, improved by 1.38% compared to RNN (Recurrent Neural Network) model.

**Keywords:** Wide and deep (W&D) learning; convolutional neural network; image feature; raw features; generalized features

## 1 Introduction

Today, as the fourth industrial revolution accelerates, smart devices pervade every sector of the economy and society. Android devices continue to dominate the market. According to [1], Android 10 was released in March 2019 and includes numerous enhancements to the application's security and usability. This version of Android has demonstrated superiority and is rapidly growing in popularity. Android 10 held a 43.13% market share in the Android operating system as of January 2021. Android's rapid development, particularly its Android 10 version, has resulted in a significant increase in malicious code on this platform.

Fig. 1 illustrates the number of new malwares discovered on Android devices. With the current emphasis on research and application of artificial intelligence, malwares are constantly improving their techniques, distribution methods, and attack methods. According to [2], each day, AV-TEST registers over 450,000 new malicious programs and potentially unwanted applications (PUA). Normally, we analyze and classify new malware based on its signature and handling characteristics. Unfortunately, Android malware detection and prevention are more challenging due to the variety of environments and Android devices.

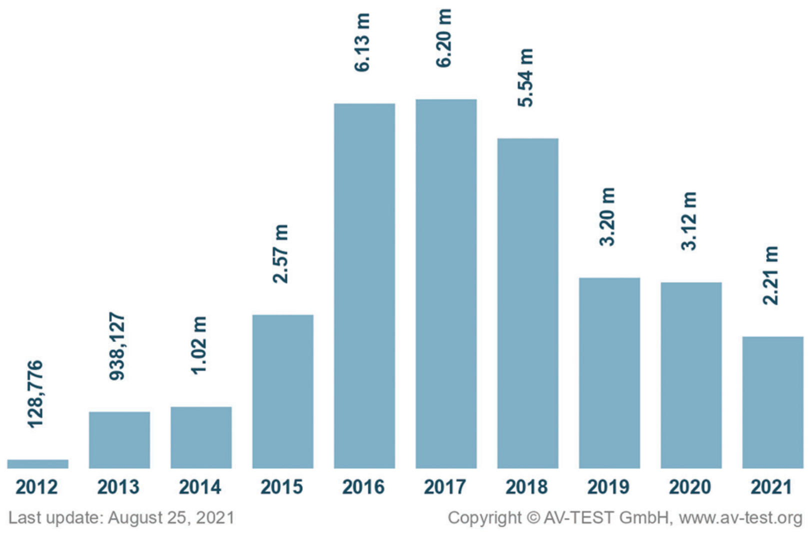


**Figure 1:** The new malware on Android

Numerous methods for determining whether an Android application is benign or malicious have been proposing. These methods can be classified into three main categories: traditional signature-based techniques, traditional machine learning techniques, and deep learning techniques.

The signature-based method is quite simple to implement in real time. This method, however, requires periodic database updating to ensure that the most recent malware types are included. For the traditional machine learning method, feature extraction is very important and must be done manually. The selection of which malware features to be extracted is critical, as these manually extracted features may affect the final classification rate. However, as malware types evolve at a tremendous rate, manually extracting features may become obsolete and unsuitable for classifying new malware. Prominently from 2017 until now, deep learning models have been increasingly applied to malware detection on Android framework, such as DBN (Deep Belief Network) [3–5], DNN (deep neural network) [6,7], RNN, LSTM (Long Short-Term Memory) [8,9], DAE (Deep AutoEncoder) [10], CNN [11–16]. These models allow learning features from the malware dataset themselves, and they give positive classification results on individual test datasets. Among deep learning models, the most widely used is the CNN, which has been successfully applied to image recognition problems where all the necessary information exists within the image. Based on this theory, a study has applied deep learning for malware classification on the Windows platform by converting the whole malware file into an *"image"* [17]. Since all information about a Windows malware is typically stored in a single PE file, the deep learning model can therefore extract useful information by learning features from the PE file's *"image"* and thus produce good classification results. However, applying deep learning to Android malware classification is not that straightforward due to the fact that an Android APK file is not a single file but rather a collection of sub files with varying structures. Physically converting the APK file into *"image"* may not work well due to the lack of connection information among sub files inside the APK. This also explains why the CNN model has a low classification rate on the Drebin Android malware dataset [18].

Recently, the Wide and Deep (W&D) model has been successfully applied to flower classification and power forecasting. This model is well suited for aggregate datasets gathered from multiple sources. In this paper, we propose a WDCNN model for Android malware classification. In our W&D (Wide and Deep) model, the deep component is responsible for extracting generalized malware features with minimal feature engineering, whereas the wide component is responsible for memorizing specific Android malware features such as the used API list, the requested permission list, etc., To prove the feasibility of our proposed model, we compare our model to (i) the deep learning-only model, (ii) the wide learning-only model, and (iii) the state-of-the-art RNN model in terms of performance. Experimental results show that our model achieves the best classification rate on the same datasets compare to other models.

The rest of this study is organized as follows: Section 2 discusses about the related works; Section 3 shows the overall mathematical model of the problem, the synthesis of feature set for W&D model to classification malware; Section 4 presents our experimental results and evaluation; Section 5 summarizes the achieved results, limitations, and some directions for further research and development.

## 2 Related Works

This section summarizes the related works, including issues related to feature extraction, classification model development, and evaluation method development. There are two primary methods for detecting malware: dynamic analysis and static analysis. Dynamic analysis is concerned with analyzing and investigating malware's behavior during execution. Dynamic analysis requires an isolated environment (a sandbox) to execute, collect, and investigate malicious behavior, and thus requires a significant amount of time and resources. Static analysis is performed without executing the malware. It analyzes the reverse code, machine code, or virtual machine code directly. As a result, static analysis is frequently used to solve several malware classification problems and is more suited to the current rapid growth of malware. This work proposes to use static analysis for Android malware classification. Features extracted by static method can be divided into two types:

- **Features extracted in form of "image":** these are raw features, represented by a color or gray-scale image. The "image" feature can be obtained by converting from a separate classes.dex file, or from the classes.dex file combined with the Androidmanifest.xml file, or from the entire APK file.
- **Features extracted in form of "string":** these are the features extracted based on the investigation of strings in the APK file, e.g., application requested permissions and API calls, memory access, etc.

Feature extraction in the form of images is also an area of interest to many research groups. In [19], Byeongho Kang et al. proposed to convert the file classes.dex to byte code form and then put it into the Random Forest (RF) model. The method was tested on a 1,300 malwares dataset, of which 26 malware families were taken from the Android Malware Genome project. The average accuracy was 94%. Of these 26 malware families, seven malware families obtain 100% classification accuracy, while the lowest malware family is over 60%.

In [20], the authors convert Android malware into gray-scale images and use K-Nearest Neighbors (KNN) to classify them. Each malware corresponds to a vector of 8-bit unsigned integers and is converted into a 2-dimensional array; each array element represents a gray level pixel, with a value in range 0–255. The classification results for 9,458 malware samples, including 25 families, achieved an accuracy of 97.18%. With the approach in research [20], it gives high classification results, but it is easy to be exploited or deceived when malicious codes are added with redundant code or simply meaningless comments. With a similar approach, FM Darus and authors extracted .DEX files from .apk files, then converted them to 8-bit grayscale and fed them into a machine learning model for malware classification and evaluation [21]. To extract features of 8-bit grayscale images, the author uses the GIST descriptor.

The experiment was conducted on 300 malware and 300 benign samples for binary classification; the results with the RF algorithm was 84.14%, KNN was 80.69% and the Decision Tree (DT) was 78.62%.

All the above studies [19–21] used conventional machine learning models to classify malware based on converting bytecode files to images. The problem is that this approach leads to information being missing from non-APK files, such as information about permissions, services, and intents stored in the file AndroidManifest.xml.

Besides traditional machine learning models, deep learning models have also been studied and applied in many fields, with typical models such as CNN, DNN, LSTM and RNN. Among the deep learning models, the CNN model is the most commonly used method. This model has high feature gener- alization ability, suitable for classification problems on large datasets. In [22], Xiao et al., collected malware from the three largest AMD datasets, namely Airpush, Mecor and Youmi with a total of 6,234 malware samples, then combined them with 4,406 benign files downloaded from Google Play. The authors converted the classes.dex files to binary code files, then to RGB color images. These color image data were fed into the CNN model, which produced a 93% classification rate. Similarly, in [23], Arp et al., converted .DEX file to RGB image for classification. However, the author converts every element in the .DEX file including the header section, string ids, type ids, etc., and the entire data section into RGB image. The size of the image is limited by the length of the file. Experimental results on the AMD dataset give an accuracy of 96%. The results of the paper are high when using deep learning. However, the study only used three large malware families of the AMD dataset. It is not clear what the results will be if using all 71 families of malicious code in AMD.

In [24], the authors directly converted malware samples into gray level images and used histogram features to classify them with CNN. Experimental results on a dataset containing 50,000 samples, of which 25,447 benign samples and 24,553 malware samples, including 71 families, achieved an accuracy of 92.9%. To further evaluate how to combine color image features with CNN, in [18], the authors proposed a capsule network based on dynamic routing in which the feature set is collected by converting the malware binary files to color images [18]. The performance of the Capsule network is compared to that of CNN on two malware datasets: Windows and Android. The Android dataset was collected from a part of Drebin dataset with 20 families of malware, including 4,000 malware samples and 6,000 benign samples. The classification rate achieved on Windows malware is 96.5% with Capsule network and 96.8% with CNN. The results achieved on the Android dataset with the Capsule network and the CNN network is 99.3% and 79.3% respectively. These experimental results show that applying CNN with image features directly converted from binary to color images is not effective for Android malware. The difference is that Windows malware executes on real machine code while Android malware executes on virtual machine code.

There are some works applying deep learning models with the features extracted in the form of strings. Lee et al., proposed an anti-obfuscation classification method for malware Android applications integrating Recurrent Neural Network and Convolutional Neural Network, with anti-obfuscation ability and lightness [9]. This method extracts the application package name, authentication data, permission, and intention features from multiple short strings. The sample dataset consists of 1,152,750 benign samples and 1,279,389 malware samples. In addition to the CNN model, this study uses a deep learning model called RNN. This method reduces the training time of the RNN model and achieves a 97.7% true positive rate with a false positive rate of 0.01. Research using deep learning gives high results in the test data set. However, the study only uses two classes of clean code and malicious code to evaluate with numerous files.

The article [13] proposes a multi-mode malware detection method based on multiple convolutional neural networks, which uses permissions, APIs, and URL features to train subnetworks. It uses a backtracking method to solve the limitations of malware detection's poor interpretability based on neural networks. The backtracking method selects the most important features that make vital contributions to classification decision-making. This method reduces the detection time and achieves 96.54% accuracy on the data set composed of 10,948 benign samples and 8,652 malware samples. In [25] Ganesh et al., used the permissions extracted from the AndroidManifest.xml file as features and applied the CNN model for training. The dataset was collected manually with 2000 malware files and 500 benign files. The average classification result is 93%. Our previous work proposes improving the feature set based on the Apriori algorithm. The features are in the form of strings, including permission and API from DREBIN [23] malware and 7,140 benign files [26]. CNN was used as a learning model, and the classification rate reached 96.71%.

For summarizing, CNN model with "image" feature seems not good for Android malware classification. It can be seen that the CNN model has high classification accuracy for the data in the form of *"image"*. However, it seems that only the CNN model with image dataset is not sufficient for Android malware classification. Because it depends a lot on how the data is arranged and the noise in the code. If we apply CNN to the malware feature in the form of *"string"* such as permission list, API list, etc., the CNN performance is almost the same as that of other traditional machine learning models. Therefore, it is necessary to find a better machine learning model than CNN.

In recent research, W&D model has been applied to synthetic feature sets [27–32]. In [27], Binh et al., applied a wide and deep model to predict type 2 diabetes. Zheng et al., and the authors used W&D Convolutional Neural Network to detect power theft [29]. In [30], Lee et al., also used a wide and deep model with a CNN deep learning component to classify the growth status of plants. The W&D models in the above research achieved better results compared to conventional machine learning methods. To address the challenge posed by the problem of classifying Android malware with large datasets and aggregate feature sets, this study proposes a classification method using W&D model. This model is suitable for Android malware classification since the feature set of Android can be simplified as two groups: synthetic features (bytecode data in the form of *"image"*) and complement features (in the form of *"strings"*).

## 3 Proposed Scheme

### *3.1 Mathematical Model*

The model aims to combine the fast classification ability of wide learning with the generalization ability of deep learning. Each input feature set will be split into two corresponding subsets. To implement the deep learning part, we use multiple CNNs. Each CNN can be configured to use both convolutional layers and pooling layers, or to use only convolutional layers (including convolution and filtering). This makes it easier to generalize features and reduce dimension. The deep and wide feature set partitioning needs to be done concurrently. To build the mathematical model, we first give the following definitions:

**Definition 1 (Initial feature set)**: The initial feature set, denoted by $F_0$, is the set containing all features included in the W&D learning model.

**Definition 2 (Wide feature set):** The wide feature set, denoted by $F_w$, is a subset of $F_0$, used for the wide learning component in the W&D learning model.

**Definition 3 (Deep feature set):** The deep feature set, denoted by $F_d$, is a subset of $\mathrm{F}_0$, used to generalize features in the deep learning component of the W&D learning model.

On the basis of the above definitions, we build an overall mathematical model of the problem as shown in Eq. (1).

$$\begin{cases} \varepsilon: F_0 \rightarrow L \\ f_p: F_0 \rightarrow \{F_w, F_d\} \\ F_0 = F_w \cup F_d \\ \varepsilon_1: F_w \rightarrow \upsilon_1 \\ \varepsilon_2: F_d \rightarrow \upsilon_2 \\ f_c: \{\upsilon\} \rightarrow L \end{cases} \tag{1}$$

where,

- $L$ is a set of labels, including benign labels and malware labels.
- $f_p$ is a partition function, to divide the set $F_0$ into $F_d$ and $F_w$.
- $\varepsilon_1$ is the mapping from wide feature set to vector $\upsilon_1$.
- $\varepsilon_2$ is the mapping from wide feature set to vector $\upsilon_2$.
- $\upsilon$ is the composite vector for train in W&D.

## *3.2 Data Partitioning Scheme*

To evaluate and partition the original feature set into a deep feature set and a wide feature set, we give the following definitions:

**Definition 4 (Raw feature):** A raw feature, denoted by *r*, is a feature that does not represent or does not fully mean a behavior, operation, or attribute of malware. For example, one byte in the *".dex"* file, i.e., one pixel in the converted *"image"* of the *".dex"* file.

**Definition 5 (General features)**: The general feature, denoted by α, is a feature that represents a behavior, operation, or property of malware. For example, a permission or an API.

**Definition 6 (Group-level general features)**: A group-level generic feature, denoted by *g*, is a feature that represents a group of malware behaviors, operations, or attributes. For example, the memory access permission group and file manipulation API group can be understood as group-level generic features.

The proposed solution must also tackle the problem of set division. We need to divide set $F$ into $F_d$ and $F_w$. Because a raw feature often doesn't have a full meaning, it needs to be generalized to form a generalizable feature or a group-level generalizable feature to reduce on the number of dimensions. Thus, the features are put into the set $F_d$. Group-level generic features are often divided into $F_w$ because these features take on the meaning and generalizability of the malware. However, when the system has a large group-level generalizable feature set, it is still possible to include $F_d$ to reduce the number of dimensions. Depending on the problem context and the level of generality, the general features and the group level features can be included in $F_d$ or $F_w$.

**The partition of the feature set**

A partition is a division of the initial feature set into a wide feature set and a deep feature set that is suitable for the problem context and properties of the feature set. To partition the feature set, we build the following Algorithm 1.

The scope of this article is focused on the initial feature set, which is composed of three subsets of features: the permissions set, APIs set, and image file converted from the byte code in the **.dex* format file. As previously stated, the set of pixels in the image file represents the raw feature set, as pixels do not fully describe a malware behavior, operation, or attribute. Permissions and APIs are two broad categories of features, each of which represents a malware behavior or operation. The deep component will receive the raw features, while the wide component will receive the general features. To implement Algorithm 1, within the scope of the paper, we choose $F_d$ as the raw feature set and $F_w$ as the set of all generalized features including permission and API. On the basis of this feature set division, the W&D model in the study is described in the next section.

**Algorithm 1 Partition algorithm**

**Input**

- $F_0$: original feature set
- $A$: Set of behavior/operation/attribute
- $G$: Set of behavior/operation/attribute groups
- $R$: Set of rules for division.

**Ouput** $F_w, F_d$

```
1:  Set $F_A \leftarrow \varnothing, F_G \leftarrow \varnothing, F_R \leftarrow \varnothing$
2:  while $F_0 \neq \varnothing$ do
3:      Put a featue $f_i$ from $F_0$
4:      if $F_i$ is satisfied by the set $G$ then
5:          Set $F_G \leftarrow F_G \cup \{f_i\}$
6:      else
7:          if $F_i$ is satisfied by the set $A$ then
8:              Set $F_G \leftarrow F_G \cup \{f_i\}$
9:          else
10:             Set $F_R \leftarrow F_R \cup \{f_i\}$
11:         end if
12:     end if
13:     Remove $f_i$ from $F_0$
14: end while
15: Set $F_d \leftarrow F_R$
16: while $F_A \neq \varnothing$ do
17:     Put a feature $f_i$ from $F_A$
18:     if $f_i$ is satisfied by the set $R$ then
19:         Set $F_w \leftarrow F_w \cup \{f_i\}$
20:     else
21:         Set $F_d \leftarrow F_d \cup \{f_i\}$
22:     end if
23:     Remobe $F_i$ from $F_A$
24: end while
25: while $F_G \neq \varnothing$ do
26:     Put a feature $f_i$ from $F_G$
27:     if $f_i$ is satisfied by the set $R$ then
28:         Set $F_w \leftarrow F_w \cup \{f_i\}$
29:     else
30:         Set $F_d \leftarrow F_d \cup \{f_i\}$
31:     end if
32:     Remobe $F_i$ from $F_G$
33: end while
34: Return $F_w, F_d$
```

## 3.3 WDCNN Implementation

In this part, we propose our WDCNN model, including the detailed parameters as shown in Fig. 2. First, the sample dataset, a set of *.Apk* files, is extracted to produce an original feature set consisting of API calls, permissions, and grey image pixel features. Each gray image is generated from a bytecode file extracted from an *.Apk* file. According to Algorithm 1, the original feature set is divided into two subsets: $F_w$ and $F_d$. $F_w$ includes general features such as API calls and permission features; $F_d$ consists of grey image pixel features. $F_w$ is put in the wide component of the model. $F_d$ is put in the deep component of the model using CNN. This model includes two components: wide and deep component as follows

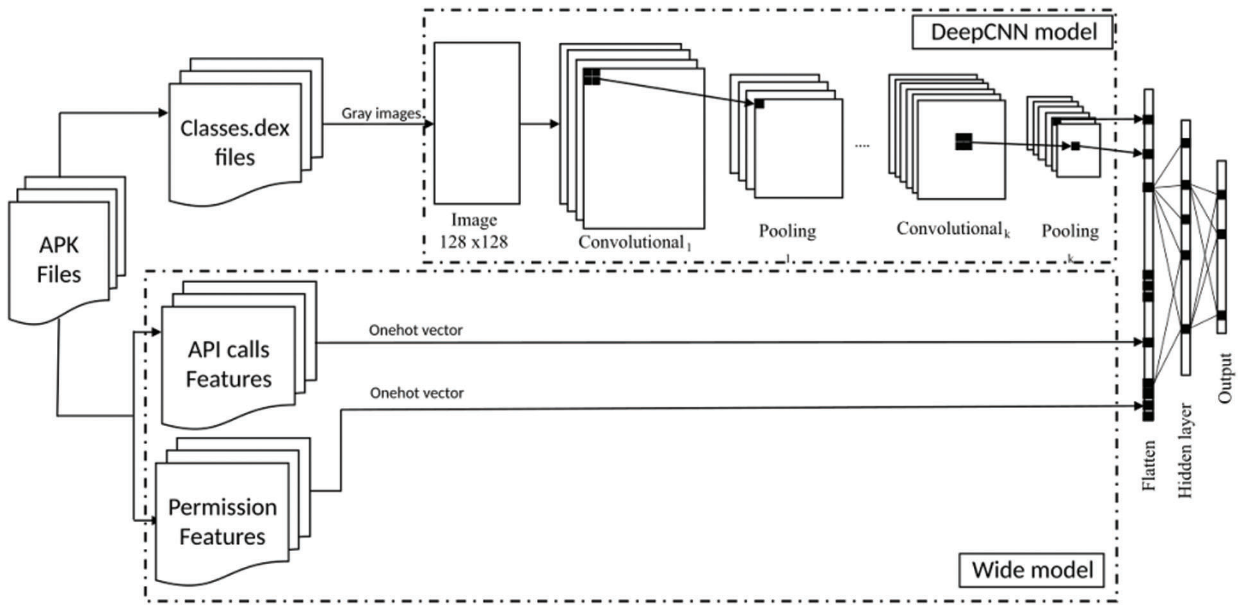


**Figure 2:** WDCNN model operation diagram

- **The deep component** The deep learning component can help derive new features (highly generalizable deep learning model) based on an internal structure consisting of convolutional and pooling layers. The raw *"image"* feature (features in $F_d$) described in Definition 4 is used as the input to our DeepCNN model. It has an input matrix of 128 × 128, four convolutional layers and pooling layers, which are used to generalize the features. In the first layer, the convolution has 32 filters used to create 32 matrices. The size of max-pooling is set at a 2 × 2 matrix, the size of each output convolutional matrix is reduced by 4 times, resulting in a 64 × 64 matrix. In the second layer, using 32 filters and the max-pooling of 2 × 2, the number of matrices is 32, but the size of a matrix is reduced by 4 times to the 32 × 32 matrix. On to the third layer, 64 filters and the max-pooling of 2 × 2 are used to create 64 matrices of 16 × 16. The number of filters and the size of the max-pooling in the fourth layer are similar to those in layer 3, so the output is 64 matrices of size 8 × 8. Finally, in the flattening layer, the outputs of the fourth layer are converted to a vector of 4069 neural units. This vector is the output of the deep component. The detailed implementation steps are shown in the left of Fig. 3.
- **Wide model component** The wide component is a generalized linear model used for large-scale regression and classification problems [28]. This component is responsible for memorizing feature interactions. In this work, the wide component is the vector of API and Permission features. Scine there are too many API calls in original dataset, we take only top (1000) of the most popular ones in the dataset.
  The API features are the top (1000) features from the original dataset, and all permission features are in the original data set. They are part of the wide component. The neurons of the DeepCNN model and the Wide model were combined as input to a dense layer of 1,024 neurons, which produced the output layer as a set of labels. The detailed implementation steps are shown in the right of Fig. 3.

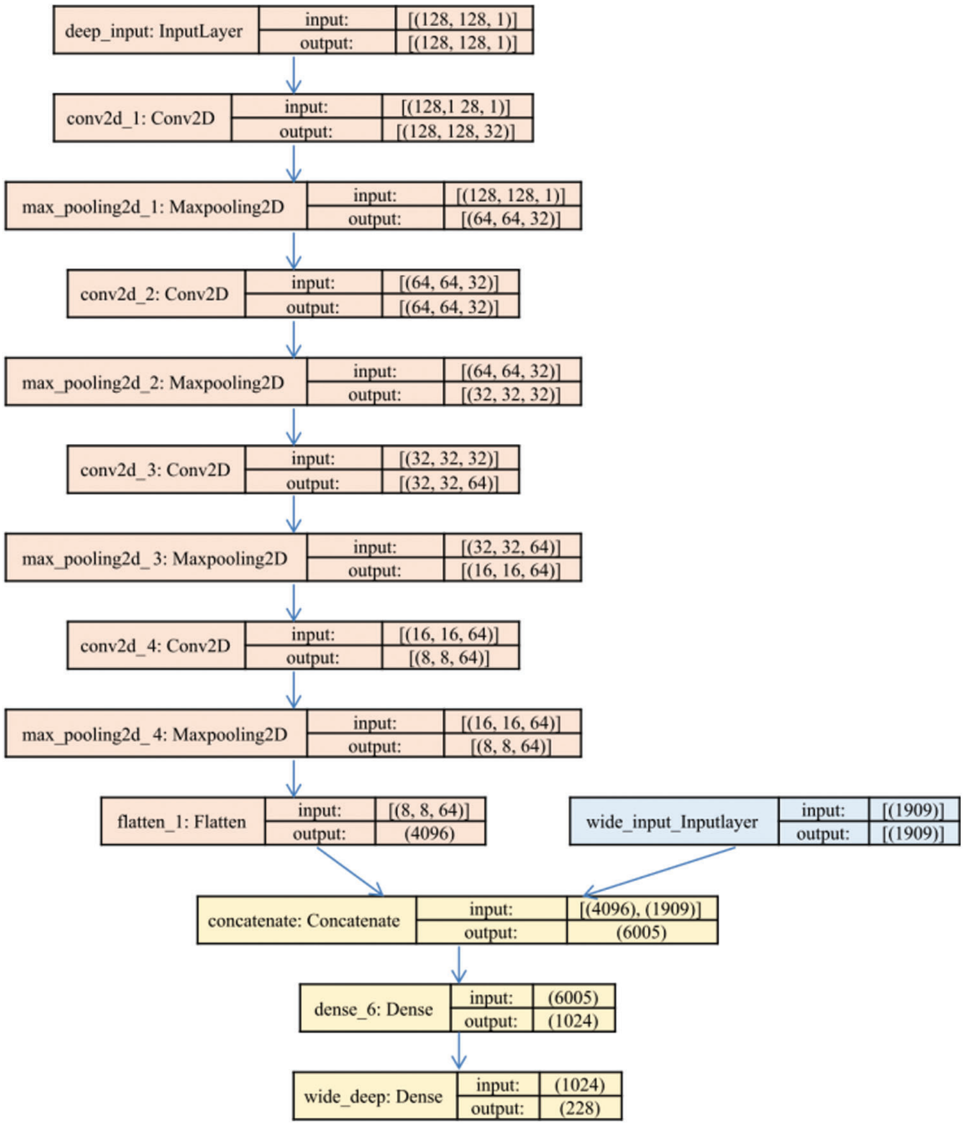


**Figure 3:** Structure and parameters of the WDCNN model

## 4 Experiment Results and Evaluation

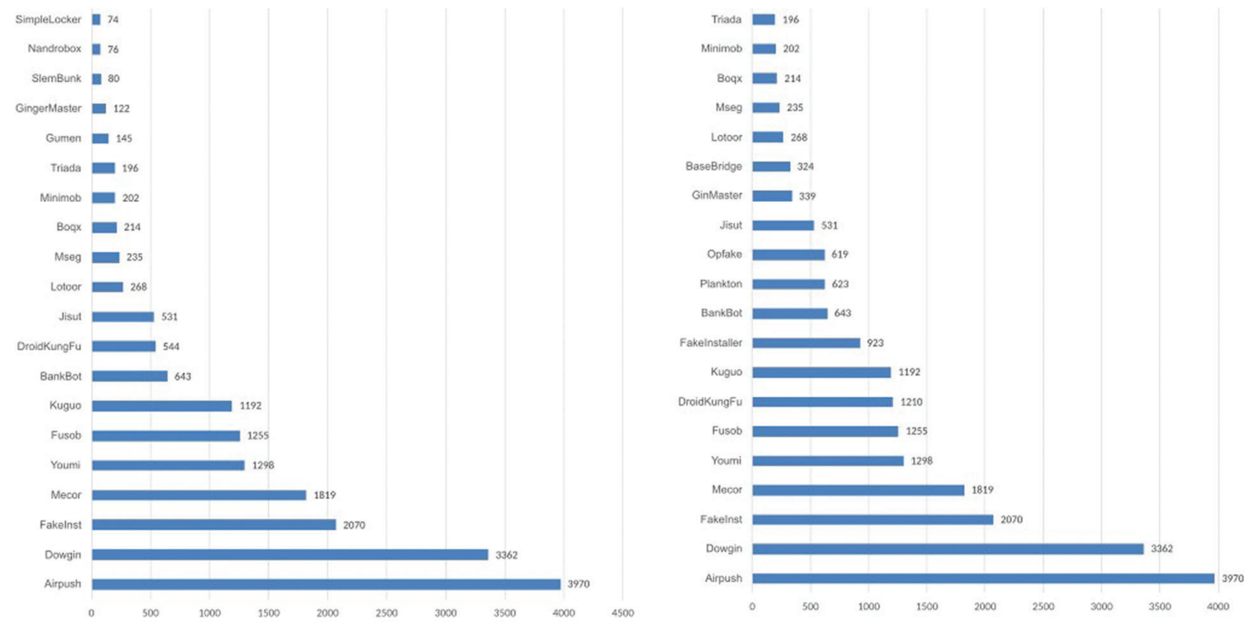
### *4.1 Experimental Data and Program*

**Experimental data**

Drebin [23] and AMD [33] are two widely used datasets for Android malware classification. However, there are very few benign samples in these two datasets. Therefore, more benign samples are collected from archive.org [26], a large and free application database. Two test datasets, dubbed *"Simple dataset"* and *"Complex dataset"*, were created in this study. The Simple dataset is comprised of all malware in the AMD dataset blended with all benign files downloaded from archived.org [26]. The Complex dataset was created by combining malware files from the AMD, Drebin, and benign datasets. Because the AMD and Drebin datasets contain 16 identical malware families, those malware families were reduced when combined. These datasets, however, have the following limitations:

- The malware files are not equally distributed into families; some families are much more dominant than others. Thus, the total number of files in the *top (10)* malware families is 16,684 files, and those of the next 10 families are 1,612 files (number of malware files in AMD and Drebin suite is 17,742 and 3,551, respectively).
- There are some malware families having few sample files, i.e., less than 10 files. The Simple dataset has nine malware families having a number of sample files less than 10, while the Complex dataset has 127 malware families similarly.

Although the malware dataset is used in the Simple dataset and the Complex dataset is widely used in the research, the uneven addition will greatly affect the model evaluation. Fig. 4 shows an overview of the distribution of files in the *top (20)* malware families. Because these two datasets are not evenly distributed among the malware families, we divide each dataset into 3 sets: the full set, the *top (20)* set, and the *top (10)* set as shown in Tab. 1.



(a) Top 20malware family in Simple datasets (b) Top 20malware family in Complex dataset

**Figure 4:** Top (20) number of files in malware dataset

**Table 1:** Experimental datasets

| Dataset | Total simple | Benign | Malware | Malware source | Description |
|---|---|---|---|---|---|
| 1 | 26,029 | 6,730 | 19,299 | AMD | Full dataset |
| 2 | 31,467 | 6,730 | 24,737 | AMD & DREBIN | Full dataset |
| 3 | 24,953 | 6,730 | 18,222 | AMD | Top (20) dataset |
| 4 | 27,826 | 6,730 | 21,096 | AMD & DREBIN | Top (20) dataset |
| 5 | 22,883 | 6,730 | 16,153 | AMD | Top (10) dataset |
| 6 | 25,829 | 6,730 | 19,099 | AMD & DREBIN | Top 10 dataset |

**Experimental program**

To build the dataset and conduct the experiment, we implemented the programs as described in Tab. 2. The dataset and experimental program in the research were compiled and published on GitHub [34].

**Table 2:** Experimental program set

| Number | Program | Function |
|---|---|---|
| 1 | Program | Collect and create the experimental raw dataset |
| 2 | Program | Feature extraction |
| 3 | Program | Implement and execute the classification models |

## *4.2 Feature Extraction*

There are two types of features we need to extract from dataset: *"image"* features and *"string"* features. For the image feature, we convert all the *classes.dex* files into images with size 128 × 128. Thus, the dimension of an image feature is 16,384. The conversion is done by treat every three bytes of the *classes. dex* as a color pixel of the target image. The color image is then converted into a gray scale 128 × 128 image.

For the features in form of strings, permissions and APIs are the two most used in malware classification. If a program is malware, it may need to transfer some sensitive data from the victim's device to the outside, i.e., the hacker's server. To do that, the program must ask for some permissions related to the network such as: *INTERNET, ACCESS WIFI STATE*, etc. The malware may ask for some other permissions such as the permission to have access to the location: *ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION*, etc. Obviously, there is a relationship between permission requests and API calls in the malware. Therefore, in this work, we extract the *"string"* features including permissions and APIs:

- To extract requested permissions from APK files we read all the permission registered in the *AndroidManifest.xml* file.
- To get the APIs, we use the tool *"AKPtool"* [35] to read the *classess.dex* files. Then we extracted all the APIs used in the dataset, and make statistics about the APIs that appear in each file of the dataset. We take the *top (1000)* of the most used APIs. The number of files corresponding to the *top (1)* API is 22,082 and were reduced to 7,110 respectively to the 1000th API. This shows that the APIs in the *top (1000)* are good features, appearing on many files in the dataset.

Those extracted features were used as input to the WDCNN operation model as described in the previous section. The entire feature of our dataset is contained in a CSV file. In the CSV file, we arrange the above permission and API call characteristics into columns (the first column is the label), the corresponding rows are APK files. The cell will be filled with the number "1" if the feature is extracted in the file. Cells are filled with "0" if the feature is not present in the file.

## *4.3 Experimental Scenarios*

To prove the effectiveness of WDCNN, we did two experiments for two scenarios as follows:

**Scenario 1**: Compare the effectiveness of the WDCNN model to each component individually.

- Compare WDCNN model to the DeepCNN model only.
- Compare WDCNN model to the Wide model only.

To compare the performance of WDCNN to each component individually, we conduct the experiment on six datasets including Simple dataset full, Simple dataset *top (10)*, Simple dataset *top (20)*, Complex dataset full, Complex dataset *top (10)*, and Complex dataset *top (20)*. The experimental process on these six datasets follows the following three scripts:

- Script 1: putting Image features into the DeepCNN model.
- Script 2: putting Permission & API feature in Wide model.
- Script 3: putting Image into the DeepCNN component while putting Permission & API to the Wide component in our WDCNN model.

**Scenario 2**: Compare the effectiveness of WDCNN model to other machine learning models including KNN, RF, Logistic, DNN, and RNN. To make the comparison to be fair, we do follow experiments:

- Using the same feature set extracted from our Simple dataset and the Complex dataset for the WDCNN model and other machine learning models.
- Using an independent feature extraction scheme, which is proposed in research [7], and apply two malware datasets. We compare the performance of our proposed WDCNN model to the best model in the research [7] using the new feature set.

### *4.4 Results and Evaluation*

To evaluate the proposed method, we conduct experiments according to the model described in Fig. 5. The initial data set is feature extracted by 3 different methods to create a composite feature set including: Image feature set, API feature set, and permission feature set. Algorithm 1 was used to divide the aggregate feature set into $F_d$ and $F_w$. Then, these two feature sets are included in the built WDCNN model for evaluation. To compare with other machine learning models, we also installed experimental programs on the synthetic feature set with models: DNN, RF, KNN, Logistic, and RNN. Then, the experimental results could be evaluated using cross test method presented in the previous section.

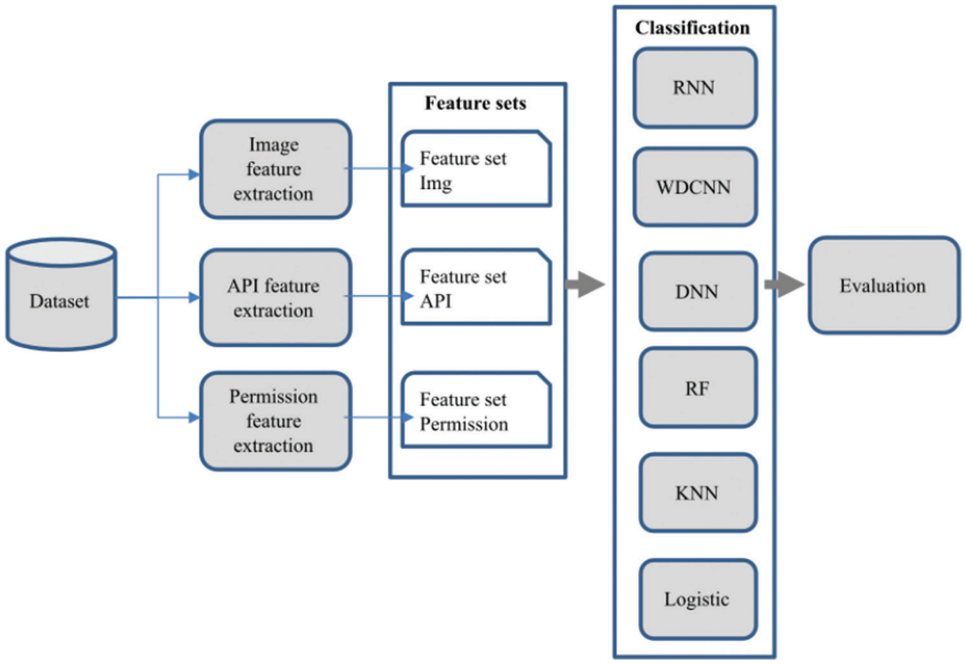


**Figure 5:** Experimental model

**Model Evaluation Method**

Since the number of malware family files are various as shown in Fig. 4, using only the average classification rate will not accurately reflect the classification result. Therefore, we propose to evaluate the accuracy of the classification for every single malware family. The metrics used for evaluating is shown in Tab. 3.

**Table 3:** Model evaluation parameters

| Term | Definition |
|---|---|
| $T\,Pc$ | The number of the malware in the family ***c*** classified correctly |
| $FPc$ | The number of the malware that is not labeled as family ***c***, but classified into the family ***c*** |
| $FNc$ | The number of the malware in the family ***c***, not classified into the family ***c*** |
| $TNc$ | The number of the malware, not in the family ***c***, classified correctly |

Correct classification of all malware families is calculated as Eq. (2):

$$TP = \sum_{c \in C} TP_c \tag{2}$$

Misclassification of all malware families is as Eq. (3):

$$FP = \sum_{c \in C} FP_c \tag{3}$$

Final false negative score is calculated as Eq. (4):

$$FN = \sum_{c \in C} FN_c \tag{4}$$

Final true negative is as Eq. (5):

$$TN = \sum_{c \in C} TN_c \tag{5}$$

Classification Macro Accuracy (Accuracy): percentage of malware which are correctly classified into their corresponding families as Eq. (6):

$$ACC = \frac{\sum_{c \in C} TP_c}{\sum_{c \in C} (TP_c + FN_c)} \tag{6}$$

Classification Micro Accuracy for Recall (Recall): Average correct classification of each malware family as Eq. (7):

$$\text{Recall} = \frac{\sum_{c \in C} \frac{TP_c}{TP_c + FN_c}}{|C|} \tag{7}$$

From Eqs. (2)–(7), we have:

- $c$: label of a family of malware.
- $C$: Label set.

In this experiment, we use the *K-fold* cross validation method with *10-fold*. The dataset is divided into 10 parts, 8 parts for training, 2 parts for testing, of which one part is testing during training (validation) and one part for final testing (test). The experimental process was conducted according to the described scenarios. Experimental results on dataset 1 are summarized in Tab. 4. Experimental results on dataset 2 are summarized in Tab. 5.

**Table 4:** Experimental results of Simple dataset

(a) *Input: Image; Model: DeepCNN*

| Set | Measure | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | AVG |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Full | Acuracy | 67.85 | 68.28 | 65.4 | 64.07 | 66.45 | 67.39 | 66.33 | 68.18 | 66.44 | 65.61 | **66.6** |
| | Recall | 54.57 | 42.96 | 55.9 | 44.41 | 52.44 | 45 | 46.88 | 47.56 | 50.45 | 46.32 | **48.65** |
| Top (20) | Acuracy | 68.37 | 69.29 | 65.72 | 64.87 | 66.92 | 68.17 | 66.96 | 68.85 | 67.85 | 66.32 | **67.33** |
| | Recall | 52.91 | 56.31 | 50.8 | 49.95 | 54.92 | 51.8 | 48.62 | 54.71 | 53.45 | 51 | **52.45** |
| Top (10) | Acuracy | 70.28 | 71.16 | 67.61 | 66.78 | 68.27 | 70.28 | 69.54 | 70.81 | 70.36 | 68.32 | **69.34** |
| | recall | 67.44 | 72.45 | 65.1 | 63.85 | 67.54 | 67.77 | 67.72 | 68.34 | 67.5 | 65.17 | **67.31** |

(b) *Input: Pemision+API; Model: Wide*

| Set | Measure | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | AVG |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Full | Acuracy | 96.21 | 98.45 | 97.64 | 97.56 | 97.25 | 98.37 | 98.45 | 97.48 | 97.06 | 98.38 | **97.68** |
| | Recall | 86.82 | 80.87 | 88.72 | 83.96 | 86.62 | 86.96 | 89.3 | 87.26 | 90.53 | 85.79 | **86.68** |
| Top (20) | Acuracy | 96.58 | 99.24 | 97.99 | 98.11 | 97.79 | 98.75 | 98.71 | 97.83 | 98.01 | 98.83 | **98.19** |
| | Recall | 94.9 | 97.22 | 94.37 | 95.37 | 92.45 | 95.74 | 96.34 | 94.49 | 94.8 | 95.35 | **95.1** |
| Top (10) | Acuracy | 96.94 | 99.61 | 98.64 | 98.69 | 98.56 | 99.3 | 99.08 | 98.21 | 98.66 | 99.43 | **98.71** |
| | recall | 97.52 | 99.66 | 98.01 | 98.63 | 98.61 | 99.26 | 98.87 | 98.52 | 97.82 | 99.4 | **98.63** |

(c) *WDCNN model with input: Image into model DeepCNN; Input: permission+API into model WideCNN*

| Set | Measure | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | AVG |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Full | Acuracy | 98.76 | 99.15 | 98.99 | 98.84 | 98.3 | 98.72 | 98.68 | 98.68 | 9.46 | 98.8 | **98.64** |
| | Recall | 90.01 | 88.89 | 93.3 | 86.98 | 91.48 | 89.78 | 86.22 | 90.64 | 91.54 | 83.84 | **89.27** |
| Top (20) | Acuracy | 99.16 | 99.6 | 99.2 | 99.24 | 98.51 | 99 | 99.08 | 99.16 | 98.4 | 99.28 | **99.06** |
| | Recall | 95.09 | 98.65 | 97.78 | 96.9 | 93.17 | 96.11 | 97.41 | 95.68 | 95.76 | 96.52 | **96.31** |
| Top (10) | Acuracy | 99.82 | 99.78 | 99.47 | 99.65 | 99.39 | 99.52 | 99.39 | 99.69 | 98.96 | 99.74 | **99.54** |
| | recall | 99.62 | 99.83 | 99.25 | 99.66 | 99.37 | 99.36 | 99.22 | 99.81 | 99.42 | 99.68 | **99.42** |

**Table 5:** Experimental results of Complex dataset

(a) *Input: Image; Model: DeepCNN*

| Set | Measure | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | AVG |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Full | Acuracy | 62.36 | 62.98 | 62.83 | 66.94 | 64.72 | 61.69 | 64.48 | 60.93 | 63.59 | 58.78 | **62.93** |
| | Recall | 36.2 | 36.87 | 46.81 | 45.51 | 43.23 | 41.68 | 46.28 | 39.38 | 43.04 | 27.49 | **40.65** |
| Top (20) | Acuracy | 65.3 | 65.41 | 64.83 | 68.94 | 67.06 | 63.57 | 66.31 | 62.63 | 67.85 | 62.67 | **65.45** |
| | Recall | 52.74 | 54.5 | 54.07 | 57.49 | 56.43 | 55.48 | 55.88 | 51.75 | 60.79 | 54.09 | **55.32** |
| Top (10) | Acuracy | 68.03 | 67.48 | 66.55 | 71.39 | 68.91 | 64.75 | 68.32 | 64.92 | 69.57 | 64.41 | **67.43** |
| | recall | 164.59 | 63.96 | 62.88 | 68.29 | 66.5 | 64.22 | 64.49 | 62.87 | 66.56 | 62.22 | **64.66** |

(b) *Input: Pemision+API; Model: Wide*

| Set | Measure | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | AVG |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Full | Acuracy | 94.31 | 93.98 | 94.33 | 94.84 | 95.02 | 94.65 | 95.97 | 95.63 | 93.87 | 92.73 | **94.53** |
| | Recall | 66.88 | 74.83 | 81.64 | 75.59 | 77.41 | 5.72 | 85.59 | 87.02 | 79.67 | 64 | **76.84** |
| Top (20) | Acuracy | 96.61 | 95.5 | 95.78 | 96.36 | 96.68 | 96 | 96.83 | 96.25 | 96.32 | 95.71 | **96.21** |
| | Recall | 93.811 | 88.5 | 91.2 | 92.23 | 95.1 | 93.22 | 94.65 | 94.05 | 92.3 | 93.39 | **92.94** |
| Top (10) | Acuracy | 96.89 | 96.6 | 96.18 | 96.51 | 96.81 | 96.18 | 96.89 | 96.18 | 96.76 | 96.55 | **96.45** |
| | recall | 95.44 | 92.4 | 92.71 | 92.57 | 95.52 | 92.69 | 93.42 | 93.37 | 92.46 | 92.58 | **93.32** |

(c) *WDCNN model with input: Image into model DeepCNN; Input: permission+API into model WideCNN*

| Set | Measure | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | AVG |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Full | Acuracy | 94.63 | 95.65 | 94.98 | 95.06 | 95.21 | 95.2 | 96.07 | 95.6 | 94.85 | 93.52 | **95.08** |
| | Recall | 65.04 | 81.13 | 81.45 | 76.32 | 79.93 | 79.17 | 83.76 | 82.09 | 84.29 | 60.97 | **77.42** |
| Top (20) | Acuracy | 96.9 | 96.97 | 96.22 | 96.22 | 96.58 | 96.54 | 94.14 | 96.54 | 96.43 | 96.79 | **96.33** |
| | Recall | 93.67 | 93.34 | 93.09 | 92.95 | 91.88 | 92.76 | 94.8 | 91.18 | 93.08 | 95.85 | **93.26** |
| Top (10) | Acuracy | 97.31 | 97.35 | 96.34 | 96.55 | 97.06 | 97.06 | 99.04 | 97.06 | 96.93 | 96.6 | **97.13** |
| | recall | 94.57 | 94.64 | 93.03 | 93.68 | 93.8 | 94.32 | 94.4 | 92.97 | 94.67 | 94.6 | **94.07** |

Based on the experimental results presented in Tabs. 4 and 5, we conduct an evaluation of the characteristics, models and datasets.

**Assess the suitability of the model WDCNN**

Tabs. 4 and 5 show the average classification results of all malware families on two datasets (Simple and Complex datasets) for three methods (proposed WDCNN, DeepCNN, and the Wide model). It can be seen that the WDCNN shows the best performance compared to the others. The average accuracy and recall of DeepCNN is quite low, less than 70%. This leads to the conclusion that using only the image feature as an input for the CNN model is not suitable for Android malware classification. The average accuracy of the wide model with permission and API features is high and reaches 97.68% and 94.53% for Simple Full and Complex Full datasets, respectively. These results demonstrated the critical importance of performance and API features in classifying Android malware. The superior performance of WDCNN demonstrates that our proposed model is an ideal candidate for Android malware classification, as it retains the benefits of the wide model while utilizing deep learning to extract useful information from raw features.

**Evaluation on malware families**

As mentioned above, the malware families differ widely in the number of files. There are some malware families that only include three or four files, while the largest malware family includes 3,970 files, as shown in Fig. 4. Hence, using the average accuracy of all families may not reflect the real performance of each model. Instead, the recall metric will be measured for each family of the *top (10)* largest malware families and the *top (20)* largest malware families for classification. The proposed WDCNN shows the best performance on *top (10)* and *top (20)* of the Simple and Complex datasets, as shown in Fig. 6. In the program, we used early stopping to prevent the model from overfitting.

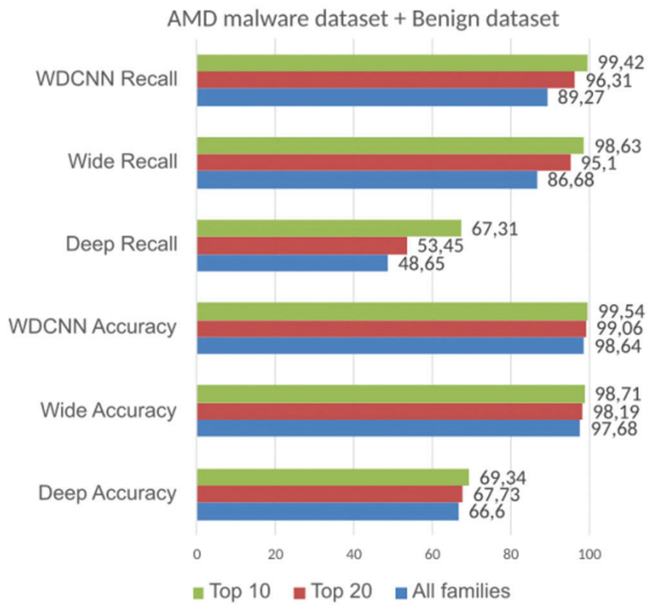


**Figure 6:** Classification of malware depending on the number of labels

**Comparing WDCNN with some other machine learning models**

*Case 1: Experiment conducted on the same feature set extracted from our dataset scenario:*

Experiment 2 was conducted on other models such as RNN [36], DNN [7], KNN, RF and Logistic using the same feature set extracted from Simple dataset (Scenario 2). After 10-fold experiment, the results were shown in Tab. 6. The results showed that our proposed WDCNN produced better results than the RNN model's (the best model mentioned in [36]) 1.38%, i.e., 98.64% compared to 97.26%. In [7], DNN gives the best result of 10 hidden layers (DNN (10)) when running with our dataset, the average classifier result is 94.5%, lower than the WDCNN model of 4.14%.

**Table 6:** Accuracy comparison of models Features: Images 128 × 128 + permission + API

| | KNN | Logistic | RF(10) | RF(50) | DNN | RNN | WDCNN |
|---|---|---|---|---|---|---|---|
| Accuracy | 39.61 | 62.98 | 77.72 | 84.8 | 89.46 | 97.26 | 98.64 |

*Case 2: Experiment conducted using the same feature extraction scheme:*

We chose the feature extraction scheme mentioned in [7]. The scheme converts every **.Apk* file to binary, then to a 256-pixel image using a histogram. The converted image was then treated as input to the DNN and DeepCNN models. On the other hand, we combine this 256-pixel image with permission and API features into the WDCNN model and the DNN (10) model. The results are shown in Tab. 7. We found that with the 256-pixel image features obtained as in [7], the result of the DeepCNN model component was 8.16% higher than that of DNN (10). Both had very low accuracy though (less than 50%). When combined with the permission and API features, the WDCNN model gives a result of 97.73%, which is 1.26% higher than that of the DNN (10) model. In addition, for the recall metric, the WDCNN model is much better than DNN (10), 85.65% compared to 47.8%. This shows that the average correct recognition classifier of WDCNN is much higher than that of DNN (10).

**Table 7:** Experimental datasets

| Features | DNN | | Deep CNN | | WDCNN | |
|---|---|---|---|---|---|---|
| | Accuracy | Recall | Accuracy | Recall | Accuracy | Recall |
| 256 pixel images | 37.43 | 2.99 | 44.59 | 9.11 | – | – |
| 256 pixel images + permission | 96.47 | 47.8 | – | – | 97.73 | 85.65 |

Based on the results shown in Tabs. 6 and 7, although the features are taken in different ways, the proposed WDCNN model always gives better performance compared to other models, especially recall metric (the correct mean value of each label).

**Discussion about the impact of different features to the final classification rate**

In our experiments, we have used three types of features, including: *"image"* feature, which is converted from *classes.dex* file in the APK malware file, the *"permission"* feature, and the API feature. To evaluate the quality of each feature, each feature was separately put into the CNN model to test for the final classification rate. For the image feature, we convert the file *classess.dex* to a RGB color image using a similar algorithm to that used in [22]. The size of each image is 128 × 128. Based on experimental results, we found that using only the *"image"* feature is inadequate for Android malware classification. The results of the CNN model with the Simple and Complex datasets, in particular, were quite low, at 66.6% and 62.93%, respectively. On the other hand, the CNN model's results when only the API feature was used and when only the permission feature was used were 92.97% and 95.54%, respectively. While the image feature alone does not produce satisfactory results, when combined with other features, it can significantly improve classification performance.

## 5 Conclusion

The WDCNN model is a combination of deep learning and traditional machine learning. This model is extremely effective when used with aggregated datasets from a variety of different sources and features with varying degrees of generalization. The purpose of this study was to propose and develop a method for classifying Android malware that utilizes a WDCNN learning model. The proposed WDCNN architecture consists of a DeepCNN and a wide component. The input for the deep component is a *"Image"* feature converted from the *classes.dex* file, while the input for the wide component is a permissions and API list. This configuration contributes to the final classification accuracy and recall metric improvements. The proposed WDCNN model outperforms the DeepCNN model or the wide model alone in experiments.

WDCNN and existing models can be compared in two ways to demonstrate that the proposed model is a good idea.

We have already published the implementation source code and dataset on GitHub; you can freely download them and evaluate the programs by visiting this link [34].

**Funding Statement:** The authors received no specific funding for this study.

**Conflicts of Interest:** The authors declare that they have no conflicts of interest to report regarding the present study.

## References

[1] Statcounter GlobalStats: Mobile & Tablet Android Version Market Share Worldwide. https://gs.statcounter.com/os-version-market-share/ android/mobile-tablet/worldwide last accessed: 24/8/2021.

[2] AV-test: Malware. https://www.av-test.org/en/statistics/malware/ last accessed: 25/8/2021.

[3] T. Chen, Q. Mao, M. Lv, H. Cheng and Y. Li, "Droidvecdeep: Android malware detection based on word2vec and deep belief network," *KSII Transactions on Internet and Information Systems*, vol. 13, pp. 2180–2197.

[4] X. Su, W. Shi, X. Qu, Y. Zheng and X. Liu, "DroidDeep: Using deep belief network to characterize and detect android malware," *Soft Computing*, vol. 24, no. 8, pp. 6017–6030, 2020.

[5] X. Qin, F. Zeng and Y. Zhang, "MSNdroid: The Android malware detector based on multi-class features and deep belief network," in *Proc. of the ACM Turing Celebration Conf.*, New York, NY, USA, pp. 1–5, 2019.

[6] M. K. Alzaylaee, S. Y. Yerima and S. Sezer, "DL-Droid: Deep learning based android malware detection using real devices," *Computers & Security*, vol. 89, no. 5, pp. 101663, 2020.

[7] F. Mercaldo and A. Santone, "Deep learning for image-based mobile malware detection," *Computer Virology and Hacking Techniques*, vol. 16, no. 2, pp. 157–171, 2020.

[8] Z. Ma, H. Ge, Z. Wang, Y. Liu and X. Liu, "Droidetec: Android malware detection and malicious code localization through deep learning," *ArXiv*, vol. abs/2002.0394, pp. 1–13, 2020.

[9] W. Y. Lee, J. Saxe and R. Harang, "SeqDroid: Obfuscated Android malware detection using stacked convolutional and recurrent neural networks," in *Deep Learning Applications for Cyber Security*, Springer Cham, pp. 197–210, 2019.

[10] W. Wang, M. Zhao and J. Wang, "Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 8, pp. 3035–3043, 2019.

[11] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei *et al.,* "IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture," *Computer Networks*, vol. 171, no. 1, pp. 107–138, 2020.

[12] D. Zhu, T. Xi, P. Jing, D. Wu, Q. Xia *et al.,* "A transparent and multimodal malware detection method for Android Apps," in *Proc. of the 22nd Int. ACM Conf. on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, New York, NY, United States, pp. 51–60, 2019.

[13] K. Zou, X. Luo, P. Liu, W. Wang and H. Wang, "ByteDroid: Android malware detection using deep learning on bytecode sequences, " in: *Trusted Computing and Information Security*, Singapore: Springer, pp. 159–176, 2020.

[14] Y. Sun, Y. Chen, Y. Pan and L. Wu, "Android malware family classification based on deep learning of code images," *International Journal of Computer Science*, vol. 46, no. 4, pp. 1–10, 2019.

[15] D. Li, L. Zhao, Q. Cheng, N. Lu and W. Shi, "Opcode sequence analysis of Android malware by a convolutional neural network," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 22, pp. 1–8, 2019.

[16] Z. Ren, H. Wu, Q. Ning, I. Hussain and B. Chen, "End-to-end malware detection for android IoT devices using deep learning," *Ad Hoc Networks*, vol. 101, no. 4, pp. 102098, 2020.

[17] S. L. S. Darshan and C. D. Jaidhar, "Windows malware detector using convolutional neural network based on visualization images," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 2, pp. 1057–1069, 2021.

[18] S. Wang, G. Zhou, J. Lu and F. Zhang, "A novel malware detection and classification method based on capsule network," in *Artificial Intelligence and Security*, Springer Cham, vol. 11632, pp. 573–584, 2019.

[19] B. Kang, B. Kang, J. Kim and G. E. Im, "Android malware classification method: Dalvik bytecode frequency analysis," in *Proc. of Adaptive and Convergent System*, New York, NY, United States, pp. 349–350, 2013.

[20] L. Nataraj, S. Karthikeyan, G. Jacob and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *Proc. of the 8th Int. Symp. on Visualization for Cyber Security*, Pittsburgh Pennsylvania, USA, pp. 1–7, 2011.

[21] F. M. Darus, N. A. A. Salleh and A. F. M. Ariffin, "Android malware detection using machine learning on image patterns," in *Cyber Resilience Conf.*, Putrajaya, Malaysia, pp. 1–2, 2018.

[22] X. Xiao and S. Yang, "An image-inspired and CNN-based Android malware detection approach," in *Proc. of IEEE/ACM Int. Conf. on Automated Software Engineering*, San Diego, CA, USA, pp. 1259–1261, 2019.

[23] D. Arp, M. Spreitzenbarth, M. Hu¨bner, H. Gascon and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket," in *Symp. on Network and Distributed System Security*, San Diego, California, USA, vol. 14, 2014.

[24] F. mercaldo and A. Santone, "Deep learning for image-based mobile malware detection," *Journal of Computer Virology and Hacking Techniques*, vol. 16, no. 2, pp. 157–171, 2020.

[25] M. Ganesh, P. Pednekar, P. Prabhuswamy, D. S. Nair, Y. Park *et al.,* "CNN-based Android malware detection," in *Proc. of Int. Conf. on Software Security and Assurance*, Altoona, PA, USA, pp. 60–65, 2017.

[26] Sketch the Cow, S.: Random .APK Collection (February 2018). https://archive. org/details/2018-02-random-apk-collection last accessed: 24/8/2020.

[27] N. P. Binh, P. N. Hung, T. Hop, N. Nhung, N. H. Quang *et al.,* "Predicting the onset of type 2 diabetes using wide and deep learning with electronic health records," *Computer Methods and Programs in Biomedicine*, vol. 182, no. 9, pp. 1–9, 2019.

[28] H. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra *et al.,* "Wide & deep learning for recommender systems," *CoRR abs/1606.07792*, 2016.

[29] Z. Zheng, Y. Yang, X. Niu, H. N. Dai and Y. Zhou, "Wide and deep convolutional neural networks for electricity-theft detection to secure smart grids," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 4, pp. 1606–1615, 2018.

[30] J. W. Lee and Y. C. Yoon, "Fine-grained plant identification using wide and deep learning model," in *Proc. of Int. Conf. on Platform Technology and Service*, Jeju, South, Korea, pp. 1–5, 2019.

[31] W. Yuan, H. Wang, B. Hu, L. Wang and Q. Wang, "Wide and deep model of multi-source information-aware recommender system," *IEEE Access*, vol. 6, pp. 49385–49398, 2018.

[32] M. Kim, S. Lee and J. Kim, "A wide & deep learning sharing input data for regression analysis," in *Int. Conf. on Big Data and Smart Computing*, Busan, South, Korea, pp. 8–12, 2020.

[33] F. Wei, Y. Li, S. Roy, X. Ou and W. Zhou, "Deep ground truth analysis of current android malware," *Springer Lecture Notes in Computer Science*, vol. 10327, pp. 252–276, 2017.

[34] Github: WDCNN–for-malware–Android. https://github.com/lethuan255/WDCNN-for-malware-Android last accessed: 5/9/2021.

[35] Apktool: A tool for reverse engineering Android apk files. https://ibotpeaches.github.io/Apktool/ last accessed: 24/8/2020.

[36] M. Rhode, P. Burnap and K. jones, "Early-stage malware prediction using recurrent neural networks," *Computers & Security*, vol. 77, no. 2, pp. 578–594, 2018.