The ISC Int'l Journal of Information Security
ISeCure

November 2022, Volume 14, Number 3 (pp. 81–92)
http://www.isecure-journal.org

Presented at the ISCISC'2022 in Rasht, Iran.

Lightweight Identification of Android Malware with Knowledge Distillation and Deep Learning Approach ☆

Somayeh Mozafari [1], and Amir Jalaly Bidgoly [1,*]
[1] *Department of Electrical and Computer Engineering, University of Qom, Qom, Iran.*

ARTICLE INFO.

Keywords:
Android, Deep Learning, Ensemble Learning, Knowledge Distillation, Lightning, Malware Detection

Type:
Research Article

doi:
10.22042/isecure.2022.14.3.9

dor:
20.1001.1.20082045.2022.14.3.9.3

ABSTRACT

Today, with the advancement of science and technology, the use of smartphones has become very common, and the Android operating system has been able to gain lots of popularity in the meantime. However, these devices face many security challenges, including malware. Malware may cause many problems in both the security and privacy of users. So far, the state-of-the-art method in malware detection is based on deep learning, however, this approach requires a lot of computing resources and leads to high battery usage, which is unacceptable in smartphone devices. This paper proposes the knowledge distillation approach for lightening android malware detection. To this end, first, a heavy model is taught and then with the knowledge distillation approach, its knowledge is transferred to a light model called student. To simplify the learning process, soft labels are used here. The resulting model, although slightly less accurate in identification, has a much smaller size than the heavier model. Moreover, ensemble learning was proposed to recover the dropped accuracy. We have tested the proposed approach on CISC datasets including dynamic and static features, and the results show that the proposed method is not only able to lighten the model up to 99%, but also maintain the accuracy of the lightened model to the extent of the heavy model.

1 Introduction

The Android operating system has become the first choice for many smartphone companies. About 86.6% of the phones that were sold in 2019 used the Android operating system [1]. Research has shown that by the end of 2020, there were about 2.8 million apps in the official Google Stores. On the other hand, some issues led to security threats in mobile devices such as open source, request permissions at the installation time, and the lack of user awareness. According to published statistics, in 2018, the total number of known malicious apps was approaching 100 million. By the end of June 2018, the number of all known malicious apps had totaled 94.2 million and, on average, mobile malware is detected every 10 seconds. Due to the growing volume of Android malware, security solutions should be provided to reduce the damage to mobile phones and maintain user security.

* Corresponding author.
☆ The ISCISC'2022 program committee effort is highly acknowledged for reviewing this paper.
Email addresses: s.mozafarih@gmail.com, jalaly@qom.ac.ir
ISSN: 2008-2045

Researchers have proposed many approaches for Android malware detection. They have used a variety of features, including static, dynamic, and hybrid to analyze and detect malware samples. So far, deep learning methods have shown the best results in this regard. Deep neural networks (DNNs) are emerging learning models, trained to approximate nonlinear functions between inputs and outputs that are used to distinguish malware from benign applications. With the help of a huge set of layers and trainable parameters, these networks may train any distinguishable pattern. Different models of deep neural networks are used to detect Android malware. Recently, converting application features to images and then use of convolutional networks has been applied [1], which is the same as the model used in this research. Creating a graph by the APIs call of an Android application and using the LSTM network is also one of the malware detection solutions [2]. The use of various features of the application using fully connected networks has also led to high accuracy in detecting Android malware [3].

In general, deep learning methods on the android phone requires high computational resources and battery usage. Due to the limited mobile resources, these solutions are not practical. In research in 2018 [4], the authors evaluated the ability of mobile phones to run deep neural networks. The research shows that mobile phone faces many challenges to run a deep neural network due to their limited resources (see Table 1).

A practical smartphone malware detection solution should consider the limited resources of the phones while keeping detection accuracy as high as possible. In this paper, we have proposed to use knowledge distillation techniques for a light DNN-based android malware detection model. The results show that this approach can keep the accuracy of the model near to a heavy model while reducing its size up to 99% in terms of parameters. Moreover, the paper shows that ensembling the lightweight models with different distillation settings leads to a new lightweight model with accuracy exactly equal to the heavy one.

This paper is organized as follows: In Section 2, the related works are reviewed. In Section 3, the required background theories are presented. Section 4 describes the proposed approach. The experimental study, dataset, and evaluation results are presented in Section 5. Finally, the paper ends in Section 6 with the conclusion and future works.

2 Related Work

Regarding android malware detection, many studies and research have been done. In all studies, features of applications are first extracted. To extract features, there are three different approaches for Android malware detection, static, dynamic, and hybrid [5]. In methods that are based on static analysis, attributes are extracted by analyzing the application source code, or reverse engineering without running the program. In methods based on dynamic analysis, semantic features are used. Features are extracted by monitoring the application while it is running on a real device or virtual environment and in a controlled environment. The hybrid methods use a combination of both features. After extracting the required features and preprocessing, the data set is created, then malware can be identified using various approaches. A common approach in this regard is the use of machine learning methods. This approach is divided into two categories: traditional machine learning and deep learning, and each of them is divided into separate subcategories.

In 2020, Salah *et al.* [6] first extracted the features statically and tried to reduce the dimensions of the features with machine learning algorithms. Then, using machine learning algorithms (logistic regression and ADABOOST vector algorithms) on the Drebin dataset, they reach 99% accuracy in detecting malware. Compared to other algorithms, the SVM algorithm has the best performance with 349 features. The feature set includes URLs, API calls, and Android application manifest information.

In 2018, Rana *et al.* [7] evaluated four tree-based machine learning algorithms for detecting Android malware, along with a feature selection method. In their experiments, 11,120 programs were used from the Drebin dataset, of which 5,560 contained samples of malware, and the rest were benign. Random forest classification has been found to perform better than the previously reported best result (approximately 94% accuracy obtained by the SVM algorithm) with 97.24% accuracy.

Tao *et al.* [8] proposed a plan that, by adapting application permissions and sensitive APIs, creates a sequence of APIs and extracts patterns in Android malware. It then detects malware using machine learning algorithms. The detection power is the F1 score, and 0.98% using the random forest algorithm. It should be noted that the attribute vector has 31 properties that consist of sensitive APIs. The data include 18,531 benign programs and 33,615 samples. The second approach is deep learning, which is one of the research topics that has been widely studied in the field of machine learning. Deep learning has become popular in recent years due to its excellent results in many research areas, as well as in the field of Android malware detection.

In 2019, Ma *et al.* [9] created a program flow con-

Table 1. Mobile resource consumption bTests results are in milliseconds [4]

Model	SoC	RAM	Android	Test1	Test2	Test3	Test4	Test5	Test6	Test7	Test8
Huawei P20 Pro	HiSilicon Kirin 970	6GB	8.1	144	130	2634	279	241	4390	779	193
OnePlus 6	Snapdragon 845/DSP	8GB	9.0	24	892	1365	928	1999	2885	303	1244
HTC 1312+	Snapdragon 845	6GB	8.0	60	620	1433	1229	2792	3542	329	1485
Samsung Galaxy S9+	Exynos 9810 Octa	6GB	8.0	148	1208	1572	958	1672	2430	612	1230
Samsung Galaxy S8	Exynos 8895 Octa	4GB	8.0	134	731	1512	1197	2519	3039	428	1422
Motorola 22 Force	Snapdragon 835	6GB	8.0	85	823	1894	1513	3568	4302	381	1944
OnePlus 3T	Snapdragon 821	6GB	8.0	106	776	1937	1707	3624	4427	365	1982
Lenovo ZUK 22 Pro	Snapdragon 820	6GB	8.0	115	909	2099	1747	3683	4363	313	2030
Google Pixel 2	Snapdragon 835	4GB	9.0	143	1264	1953	1168	2104	4219	394	1360
Google Pixel	Snapdragon 821	4GB	9.0	116	867	1838	1287	2489	4125	365	1568
Nokia 7 plus	Snapdragon 660	4GB	9.0	136	944	2132	1320	2519	4641	475	1509
Asus Zenfone 5	Snapdragon 636	4GB	8.0	110	1055	2405	1910	4271	4877	515	2330
Google Pixel C	Nvidia Tegra X1	3GB	8.0	105	1064	2585	2104	4546	5036	429	2439
Huawei Honor 8 Pro	HiSilicon Kirin 960	6GB	8.0	121	1720	3163	1943	4791	5719	1082	2764
Sony XA2 Ultra	Snapdragon 630	4GB	8.0	170	1653	3424	2638	5497	6338	685	3166
Meizu Pro 7 Plus	Mediatek Helio X30	6GB	7.0	327	3357	4550	2215	4971	5502	1666	2651
BlacicBerry Keyone	Snapdragon 625	4GB	7.1	160	1695	3525	2780	6150	7164	780	3628
Sony X Compact	Snapdragon 650	3GB	8.0	111	1804	3566	2469	5789	6846	835	3527
Xiaomi Redmi 5	Snapdragon 450	3GB	7.1	188	1753	3707	3020	6144	7144	751	3580
Huawei Nexus 6P	Snapdragon 810	3GB	8.0	106	1962	4113	3389	8155	9805	930	4733
Meizu MX6	Mediatek Helio X20	4GB	7.1	183	2217	4981	3906	9245	10551	936	4870
HTC U Play	Mediatek Helio P10	3GB	6.0	239	2061	4303	3563	7537	10116	989	4368
Xiaomi Redmi 4X	Snapdragon 435	3GB	7.1	246	2640	5428	4155	8575	9979	1229	5030
Samsung Galaxy .37	Exynos 7870 Octa	3GB	7.0	278	2092	4648	3881	8495	9644	941	4699
LG Nexus 5	Snapdragon 800	2GB	4.4	332	2182	5080	5732	9625	12375	1299	5948
Asus Zenfone 2	Intel Atom 23580	2GB	5.0	1507	2433	6188	4337	12878	15128	1176	6947
Motorola Moth C	Mediatek MT6737	1GB	7.0	414	3394	7761	6356	14760	16721	1668	7856
Samsung Galaxy S3	Exynos 4412 Quad	1GB	4.3	553	4640	10321	7587	17187	21904	2059	9291
Fly Nimbus 15	Spreadtrum SC9832	1GB	7.0	538	5103	12618	7594	19174	22758	2094	9935
Huawei Ascend P1	TI OMAP 4460	1GB	4.1	482	7613	25105	12667	30743	35417	4015	18836

trol diagram to obtain the API, and then based on the API information three data sets are created, binary data set, iteration frequency data set and time series data set. Based on these three data sets, three detection models were developed to identify Android malware regarding API contacts, API frequency, and API time sequence aspects. Finally, a model for adaptation was developed. In the first model, all APIs are extracted from the graph and generated based on a reference set of binary vectors. With c4.5 algorithm detects malware with 96.81% accuracy. In the second model, which is based on the number of iterations of APIs, using the breath first search algorithm, the number of iterations of each API is calculated and then based on the reference set of the final vector and these vectors are given to a deep neural network with a different number of layers. The average accuracy is 97.70% for malware detection. In the third model, using the depth-first search algorithm (DFS), all executable paths of the program are extracted and then from each path, the last API, which is a system call, is extracted and given to the LSTM network as a numerical sequence based on the reference set. Finally, with about 99% accuracy, it detects Android malware. Experiments have been performed on 10,010 benign programs and 10,683 malicious programs. This study uses the ANDROZOO and AMD datasets to extract the properties.

In Zaho *et al* [10], the opcode sequence, from the Drebin dataset for malware, extracted and one-hot matrix into a deep neural network with convolutional layers. Accuracy in this approach is 99%, which is 2-11% higher than surface learning methods with the same data set. Opcodes extracted from smali files. The opcode properties are created from the Dalvik instructions that come from the smali file. A binary matrix is made up of sequences. This matrix is then multiplied by a random matrix, which is the embedded space, and a representation of the input matrix is prepared for the use of convolutional layers. This model has used pooling layers in the k-max strategy. This paper also uses traditional learning algorithms and finally concludes that the use of convolutional layers has the best efficiency.

Rahali *et al.* [1] have proposed an image-based deep neural network method for classifying and identifying android malware taken from a malware database with

12 categories of malware. Their method, by using deep learning and converting android applications to images, has been able to achieve 93% accuracy in classifying and identifying the android malware family.

Alzaylaee *et al.* [11] proposed DLdroid project that first extracted the features using both static and dynamic methods. 178 features dynamically consisting of API and intent calls and about 300 permissions that are statically extracted, then create a CSV file and deliver to a deep neural network. The total number of applications is 31,125, of which 11,200 are android malware and the rest are benign. The accuracy of their method reaches 99.6%.

Pektas *et al.* [12] in 2020 followed all the execution paths of a program and drew an API call graph for all execution paths and performed a preprocessing on the obtained graphs. Then a numerical vector is considered as the embedding vector and normalization is performed. Then a deep neural network (convolution) was run on the model and detected malware with 98.86% accuracy on AMD, Drebin, and Androzoo datasets.

In 2019, Kim *et al.* [13] first extracted permissions and components and environment information from raw data, including: manifest, dex file, several sensitive API calls, smali file and number of calls per function. For each of them, a binary was created and finally, based on the similarity of these vectors with properties extracted from the malware, a single vector was created and given to the neural model with dense layers. This method can detect malware with 98% accuracy.

In 2019, Huang and Kao [14] proposed a method that extracts Dalvik bit codes and converts them to RGB color as fixed-size color images, then passes them to the convolutional neural network. The accuracy of this paper has reached about 99% as well as in terms of time, this analysis takes 0.4 seconds.

In 2019, Fallah and Bidgoly [15] investigated several well-known methods of machine learning for smartphone malware detection by network traffic. In the paper, supervised machine learning methods are considered. They benchmarked the methods using different features, such as the required features count, the network traffic volume, the malware family identification, and the new malware family detection. The results show that using these methods with appropriate features and network traffic volume would achieve the F1-measure of malware detection by a percentage of about 90%.

Giff *et al.* [16] created two data sets using static analysis, attribute properties, and data extraction from the use feature tag, then combine them and train with a deep neural network. The proposed model achieves 94.5% accuracy in detecting android malware. Sample malware has been collected from the genome database.

In [17], the authors proposed an explainable high-accuracy Android malware detection, called PAIRED. PAIRED used 35 static features to produce a prediction of whether an application is malicious or benign. The results showed that the PAIRED achieved an accuracy of 0.9802 with an FN rate of 0.0090. The novelty of this study is summarized by the creation of a lightweight system (by static features) with high accuracy, on the Drebin-215 dataset. The model was also explained using SHAP values to increase trust and understand the internal operations of the proposed classifier model.

In [18] MSFDroid, a multi-source fast Android malware detection model was proposed. This model uses information from the Android app files combining information entropy, file headers, and manifest files, to build the base models for ensemble learning. Meanwhile, this study proposed a soft voting method by adjusting the weights of each base model and thus improving the performance. The results show that it achieves a trade-off between performance and efficiency, which enables efficient and accurate detection of Android malware for static Android malware detection.

In [19] the authors proposed a malware detection model called SeqNet. This model could be trained with little memory on the raw binary data. By avoiding contextual confusion and reducing semantic loss, SeqNet kept the detection accuracy by reducing the number of parameters to 136K. Compared to existing models, SeqNet has a smaller size with enough detection accuracy.

In [20], the authors presented an Android malware detection system based on deep learning that uses static features to distinguish between malicious and benign applications. To reduce the feature dimensions, they have used the feature engineering approach, which utilizes a multilevel feature reduction and elimination process to create a detection model lightweight. The proposed detection system achieves accuracy and precision of about 98% and reduces the size of the feature from 3,73,458 to 105 features. Their evaluation dataset contains malware and benign applications with the same number of samples of both.

In all research that has tried to detect Android malware by deep learning, they have been able to detect android malware with high accuracy. In general, deep learning methods on the android phone requires

high computational resources and a lot of use of the mobile phone battery. In a study conducted in [4], the authors evaluated the ability of mobile phones to run deep neural networks. In this paper, 4 types of mobile processor platforms (Qualcomm, HiSilicon, MediaTek, and Samsung) and about 200 types of mobile phones have been used. Also, 8 different categories of AI-Benchmark criteria that are designed for deep neural networks, have been used for the test. The authors concluded that although mobile phone resources (CPU, GPU) are sufficient for everyday use, they can still be some challenges to running a deep neural network. Also, battery consumption has been significant in performing these tests. Table 1 shows the detailed result.

3 Background Theories

3.1 Deep Neural Network

Today there is a huge amount of data, and deep learning has become more popular than any other machine learning algorithm. Fully connected neural networks, convolutional neural networks (CNN), recurrent neural networks (RNN), and long short-term memory (LSTM) networks are examples of deep neural networks. Most of them are trained by an algorithm called backpropagation [21].

The DNNs contain several neurons in each layer. There are connections or edges between the nodes of each layer and adjacent layers. All of these connections are weighted. The neural network consists of three parts, the input layer, the hidden layers, and the output layer. and each layer has an activation function. It is trained by giving inputs and comparing the result with the expected output. The two values should be close to each other. The whole part can be seen as a trainable parametrized function that tries to reach the optimal point of a problem using the gradient descent approach. Among of DNNs, convolutional networks have had significant success in the field of image processing and language processing [22]. These networks include important components such as convolutional layers and pooling layers.

Convolutional layers extract important features of the image. In this layer, feature mapping is performed. To this end, a filter is applied to all parts of the image, internal multiplication is done between the two, and each of the pixels of this filter is internally conjunction by the image pixels, and the sum of the multiplication of all pixels with their corresponding pixels creates a new matrix. This way, a matrix with smaller dimensions is created. CNN has shown significant results in detecting malware [14, 23]. The current work also uses these networks.

3.2 Ensemble Learning

The combined machine learning model or ensemble model is one of the machine learning methods that help achieve a more accurate model. It is shown that when weaker models are properly combined, they can produce more accurate or stable models [24]. This technique, which is called ensemble learning, may help to get results that are more efficient. In machine learning models, the choice of algorithm is very important in obtaining good results. The model selection depends on many variables in the problem such as the amount of data, the dimensions of the data and the distribution. In hybrid machine learning methods, basic models are combined as components to create more complex models. Most of the time, these models do not perform well results by themselves, because they have a high bias or variance [25].

To create a hybrid machine learning method, we should first select the basic models. In many cases, especially the bagging method, a single basic learning algorithm is used, so we have several identical basic models that are learning in different ways, which are called homogeneous hybrid models. In another method, different types of basic learning algorithms are used, which are called the heterogeneous hybrid model.

There are three general ways to combine basic models: Bagging method: In this method, homogeneous basic models are used, they are trained independently of each other in parallel, and then combined with the deterministic averaging process [26]. Boosting method: This method also uses homogeneous basic models that are trained sequentially and with an adaptive method (so that a basic model depends on its previous model) and combined with a definite strategy [27]. Stacking method: This method uses heterogeneous base models that are trained in parallel and combined with training a meta-model of the predicted output method of base models [28].

Overall, bagging methods are more focused on creating a hybrid model with less variance than its base models. While boosting and stacking methods try to create a stronger model with less bias than their base models (variance may even be reduced) [24].

When we teach a model, whether it is a matter of classification or regression, we obtain a function that takes an input and returns an output according to the instruction set. The idea of the bagging method is simple, several independent models are trained and combined with their prediction to obtain a model with less variance.

There are several methods for combining basic models that have been learned in parallel. For regression

problems, the output of these models is averaged to obtain the output of the hybrid model. For classification issues, the output class of each base model is considered as one vote, and the class that wins the majority of votes will be the output of the composite model. This method is called hard selection. For classification problems, we can consider the probability for each of the classes returned by the models and average these probabilities and keep the class with the highest average probability. This method is called soft selection. The methods of averaging or voting can be simple or weighted.

3.3 Knowledge Distillation Model

Deep neural networks in recent years will be successful in almost field. Even, though these models are huge and have many parameters, so they cannot be used on devices with limited resources. Knowledge distillation is referring to the idea of model compression to teach a lighter network than a heavy network. As shown in Figure 1, the ability to generalize the heavy model to a small model can be used by the heavy model as a soft label to teach the small model using the softmax function along with the temperature. To transfer knowledge, the same training data set is used that was used to teach the heavy model. In transferring knowledge, the heavier model or the teacher model is pre-trained, then a light model is created. The loss function of the student model is the difference between the prediction of the student model and the hard labels.

A loss function for distiller with a temperature that calculates the difference between soft labels of the light model and soft labels of the heavy model. The back-propagation operation is repeated and the model is learned in such a way that this difference is minimized and the knowledge of the student model is completely transferred to the student model without losing any information. When the heavier model is a large set of simpler models, the arithmetic mean or geometric mean of their separate predictive distributions can be used as a soft label. Soft goals provide more information in each training case than hard labels. The activation function in a "neural network" is used to normalize the output of the network and convert it to a probability distribution. Normalization is performed relative to the predicted output classes. As shown in Equation 1, the softmax function takes the input of a vector, consisting of k, of real numbers, and converts it to a probability distribution consisting of k probabilities commensurate with the representation of the input numbers. This means that some components of the vector (z) may be negative or more than one before using softmax. But after using the function, each component is in the range (0, 1) so that the sum of them is equal to 1 and input with larger values will be more likely.

$$\vartheta(z_i) = \frac{e_i^z}{\sum_{j=1}^{k} e_j^z} \tag{1}$$

for $i = 1, \ldots, k$ and $z = (z_1, \ldots, z_k) \in \mathbb{R}^k$.

When using distillation, the softmax function changes as shown in Equation 2. Here T is used as the temperature. Increasing the temperature allows us to soften the output layers and thus allows us to extract possible outputs. When $T = 1$ the softmax function is standard. As T grows, the probability distribution that generated by the softmax function becomes smoother by providing more information about which model class is more similar to the predicted class. This is called "dark knowledge" embedded in the model. The heavy model has a softmax function with temperature in the output layer and the student model has two softmax outputs with temperature and another output with the standard softmax function in the output layer [29].

$$\frac{exp(\frac{z_i}{T})}{\sum_j exp(\frac{z_j}{T})} \tag{2}$$

4 Proposed Methodology

In this section, the proposed framework has been introduced, including the soft label construction as well as the bagging strategy for assembling.

4.1 The Overview of the Proposed Framework

Since the anatomy analysis of malware needs to take a deep look at the relationships between the features to reveal the malicious behavior and identify its patterns using fewer computing resources, we tried to achieve these by implementing feature selection, deep learning, distillation and finally bagging ensemble. Figure 3 shows the architecture of our methodology, first, a tree classifier is applied to the features to select the most important ones. Part two is the deep learning model, where the input is 2D images that are created using features to be fitted in the convolutional layers. Then distillation (teacher-student) is used to lightweight the processing and finally we used a bagging ensemble that can achieve an accuracy as high as the teacher model. Figure 3 shows the proposed framework.

4.2 Feature Selection

Carefully selecting features to teach the learning algorithm is important because data sets, combined with a set of wrong features, can lead to unreliable results.

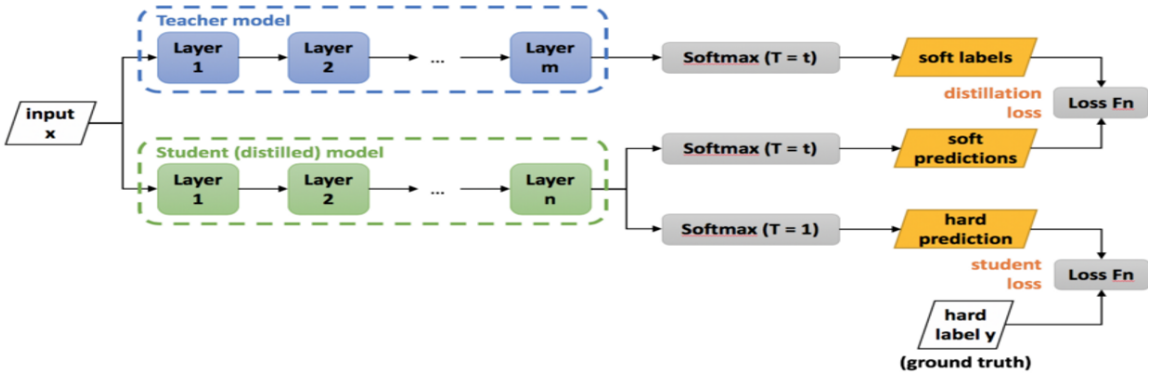

Figure 1. The knowledge distillation model

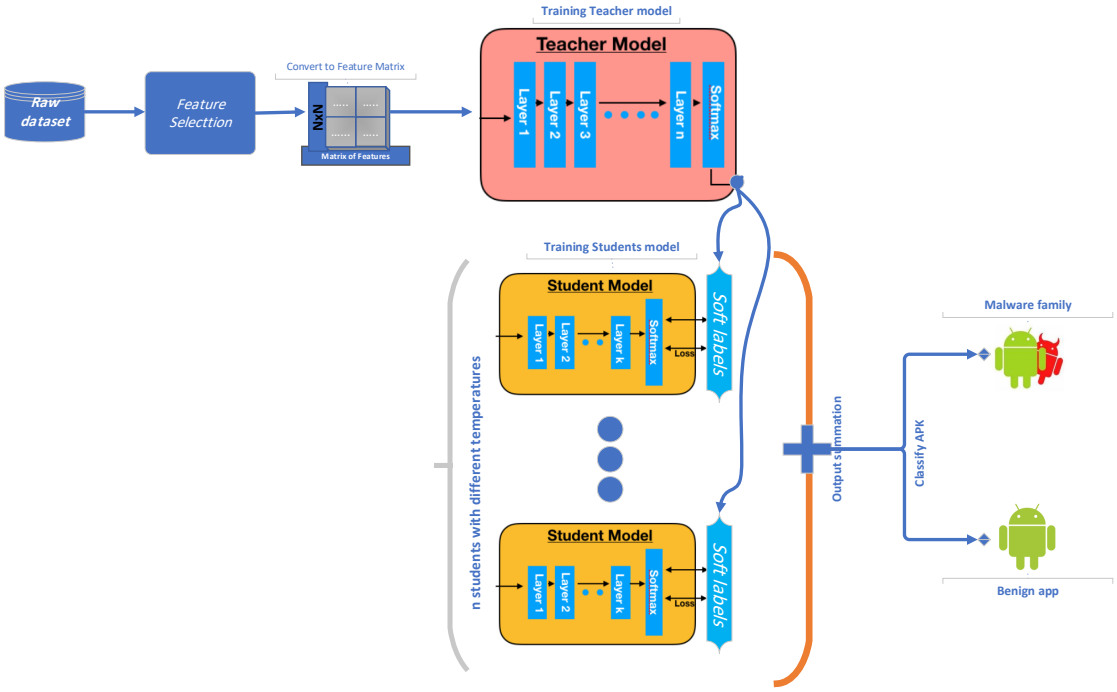

Figure 2. The proposed framework

That is why we applied feature extraction to the data to increase the performance of the proposed model. We used by Extremely Randomized Trees Classifier (Extra Trees Classifier) to select features. This algorithm is a group learning technique that aggregates the results of multiple uncorrelated decisions. Trees collected in a "forest" are classified to achieve the desired output. During forest construction, for each feature, a normalized value called the Gini coefficient is calculated. The features are sorted in descending order based on this parameter, and we keep the k important of them and delete the rest. The Gini index is a metric for the distribution of a parameter across a set of numbers. A higher Gini index shows greater inequality.

4.3 Base Classifier Learning

This section describes the CNN model layers. The model proposed in [1] is used as the heavy model. The input layer on CNN contains the two-dimensional matrix of features. We have used the feature selection method described in the previous step. These features are in the form of a one-dimensional vector that we have transformed into a two-dimensional matrix. Intuitively, all application features are converted to 2D gray images. After that, there are three convolutional layers with pooling as the feature extraction layers. As shown in **??**, the first convolution layer consists of 32 filters with a size of 3×3. The second layer contains 64 filters with a size of 3×3 and the next convolution layer includes 128 filters with a size

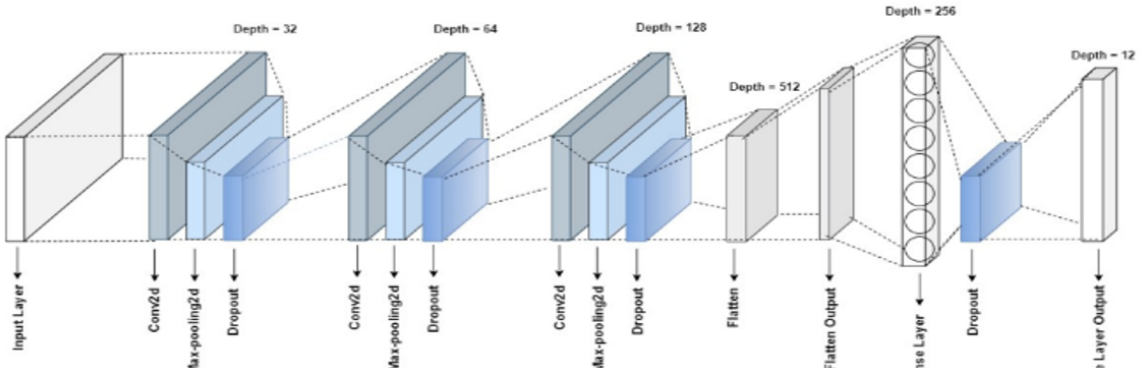

Figure 3. The CNN model [1]

of 3×3. The layers are followed by a dropout layer to prevent overfitting. Finally, the last layers of the model are two dense layers that are responsible for decision-making.

The first dense layer contains 256 neurons. The output of the model is a dense layer with 13 classes (12 classes of malware and 1 class of benign). Assigns a probability value to each of the outputs so, the highest probability is the predicted class.

4.4 Distillation

By distilling knowledge and creating the student-teacher model, the knowledge of the heavy model (so-called teacher model) is transferred to the light models (named students) without losing the information of the teacher model. The student model is similar to the teacher model, except that the number of filters in all convolution layers is equal to 8. The first layer of dense also has 8 neurons.

Algorithm 1 The pseudo-code of students bagging

Input:
Data: Matrix of App features
Model: Teacher Model
x: Suspicious app features
Output:
Predicted Class of x (Y:Set of All Classes)

$Students \leftarrow \emptyset$
for $k \in [0..100]$ **do**
    $student_k \leftarrow$ train student on Model with temperature k
    $Students+ = student_k$
end for
▷ Generate Bagging Model
for $student_i \in Students$ **do**
    $out_i \leftarrow student_i(x)$
end for
return $argmax_{y \in Y} \Sigma_{i=1}^{len(Students)} p(y = out_i)$

4.5 Bagging

Group learning involves methods that combine the predictions of several models. These predictions can work better than a single case. In this study, distillation was performed at different temperatures. At each temperature, the output is an array with 13 entries. After distillation with different temperatures, peer-to-peer array entries are summed and then the arg-max array of each program sample is returned as output. Algorithm 1 shows the pseudo-code of bagging.

5 Experiment and Results

5.1 Datasets

Experiments have been performed on two datasets: 1) static and 2) dynamic feature datasets.

- Static dataset: The static features dataset that is used in our experiment is CCCS-CIC-AndMal-2020, [1]. It includes families of malware in addition to benign android applications which includes 400K android apps. The dataset has 12 malware categories including Adware, Backdoor, FileInfector, Potentially Unwanted Apps (PUA), Ransomware, Riskware, Scareware, Trojan, Trojan-Banker, Trojan-Dropper, TrojanSMS, Trojan-Spy. The dataset has already been used in machine learning-based malware detection research.
- Dynamic dataset We used the CCCS-CIC-AndMal-2020 [30] for the dynamic dataset. The features of this dataset are extracted from programs that run in simulated environments. Using a data set containing 24,175 applications containing 13 families of malware. The number of features extracted for each program is equal to 142. By creating a convolutional network and creating a teacher or complex model.

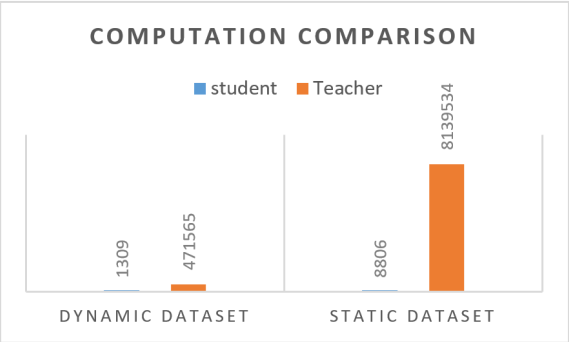

Figure 4. Computation comparison in two datasets

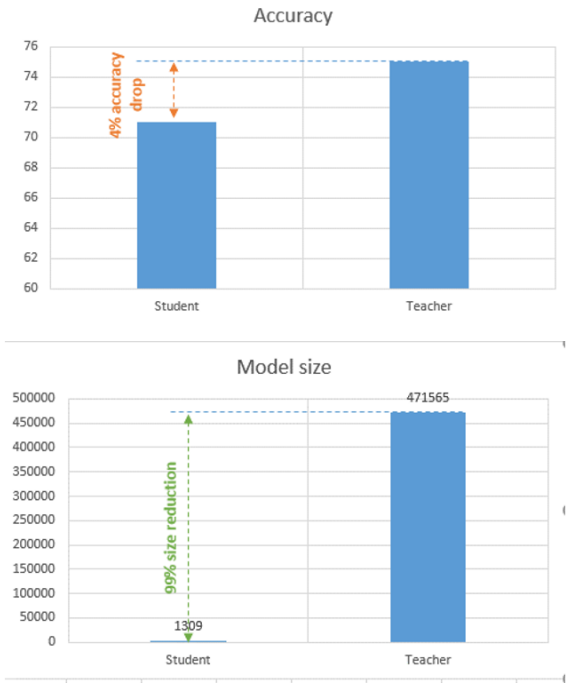

Figure 5. Computation vs. Accuracy reduction (dynamic dataset)

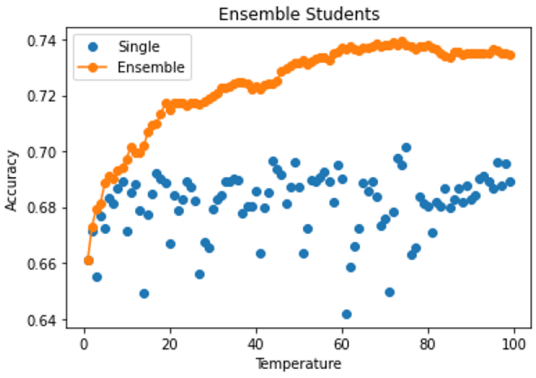

Figure 6. Bagging students with different temperatures

5.2 Knowledge Distillation Results

The models have been implemented in Python using Keras and TensorFlow. The following hyper-parameters are configured to perform the experiments: Activation (Hidden layers): RELU, Optimizer: Adam, Epoch: 50, Batch Size: 16 The first experiment is malware detection using the static feature dataset, which includes 13 families of malware. In this experiment, about 900 features were selected as the best features from 2200 features using the decision tree. Then, using the convolutional layer and dense layers and using knowledge distillation operation to reduce the model calculations, the accuracy reached 92% accuracy. As can be seen in Table 2, Knowledge distillation was able to close the accuracy of the light model to nearly the same as the heavy model. The light model, without knowledge distillation, has an accuracy of about 89%, while using distillation its accuracy increased up to 92%. The accuracy of the heavier model is 93%, but it has 100 times more parameters. The results clearly show the effectiveness of knowledge distillation of the lightning deep model while maintaining their performance.

Table 2. Static data set results

	model parameters#	accuracy
BaseLightModel	8806	89%
TeacherModel	8139534	93%
DistilledModel	8806	92%

Table 3. Dynamic data set results

	model parameters#	accuracy
BaseLightModel	1309	64%
TeacherModel	471565	75%
DistilledModel	1309	71%

Another data set in our experiment is the dynamic feature dataset. The result of the mentioned dataset can be seen in Table 3. The Android malware family can be detected with 75% accuracy with the heavier model. Then, by distilling and transferring knowledge from the teacher model to the student model, 71% accuracy is achieved. The lightweight model without knowledge distillation detects Android malware with just 64% of accuracy. Figure 4 shows the Computation comparison of two datasets. Figure 5 shows the reduction in model size along with the reduction in accuracy. As can be seen, distillation was able to reduce the model size by 99%, while the accuracy just dropped by 4%. Using ensemble learning, we can increase the accuracy of student models for Android malware identification. To this end, the student models were trained with different temperatures between 1 and 100.

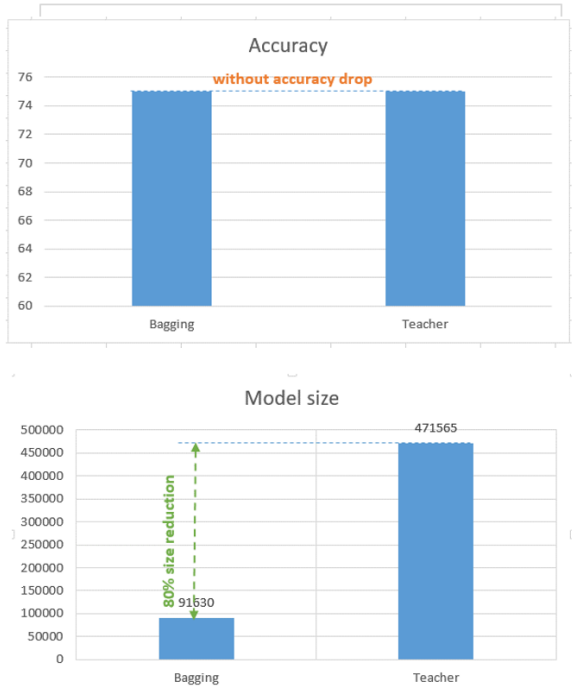

Figure 7. Computation vs. Accuracy reduction (bagging)

Finally, these 100 models are combined with the bagging method. The achieved model shows an accuracy of 75%, which is the same as the heavy model. Figure 6 shows the accuracy of a different number of ensemble models in comparison to the single student models. Each blue point is the accuracy of a single student with a temperature indicated on the X-axis. As can be seen in this figure, while none of the students have accuracy above 71%, the combination of up to 70 student models can increase the accuracy of the model up to 75% which is the accuracy of the heavier model. The size of this model is compared with the heavy model in Figure 7. As can be seen, the model size is still reduced by 80%, while there is no drop in the accuracy.

6 Conclusion

In this paper, we proposed a method for lightweight deep malware detection models on mobile phones. In the proposed method, first, a heavy model is taught and then with the knowledge distillation approach, its knowledge is transferred to a light model called student. In this approach, soft labels are used to simplify the learning process.

The resulting model, although slightly less accurate in identification, has a much smaller size than the heavier model. To continue, ensemble learning was used to recover the dropped accuracy. The results show that combining 70 students with different settings can lead to a model with accuracy exactly equal to the heavy model and at the same time a much smaller size. The proposed method is examined on two different datasets including dynamic and static features, and the results show that in both datasets the proposed method can maintain accuracy and at the same time significantly reduce the model size. However, the results showed that this method has a greater impact on more complex features such as dynamic features.

References

[1] Abir Rahali, Arash Habibi Lashkari, Gurdip Kaur, Laya Taheri, Francois Gagnon, and Frédéric Massicotte. DIDroid: Android malware classification and characterization using deep image learning. *ACM International Conference Proceeding Series*, pages 70–82, 2020.

[2] Xi Xiao, Shaofeng Zhang, Francesco Mercaldo, Guangwu Hu, and Arun Kumar Sangaiah. Android malware detection based on system call sequences and LSTM. *Multimedia Tools and Applications*, 78(4):3979–3999, feb 2019.

[3] Arvind Mahindru and Paramvir Singh. Dynamic Permissions based Android Malware Detection using Machine Learning Techniques Smartphones Security View project Android malware detection View project Dynamic Permissions based Android Malware Detection using Machine Learning Techniques. *dl.acm.org*, pages 202–210, feb 2017.

[4] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. AI Benchmark: Running deep neural networks on android smartphones. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11133 LNCS, 2019.

[5] Kimberly Tam, Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, and Lorenzo Cavallaro. The evolution of android malware and android analysis techniques. *ACM Computing Surveys*, 49(4), jan 2017.

[6] Ahmad Salah, Eman Shalabi, and Walid Khedr. A lightweight android malware classifier using novel feature selection methods. *Symmetry*, 12(5):858, may 2020.

[7] Md Shohel Rana, Sheikh Shah Mohammad Motiur Rahman, and Andrew H Sung. Evaluation of Tree Based Machine Learning Classifiers for Android Malware An Optimized Perona-Malik Anisotropic Diffusion Function for Denoising Medical Image View project Phishing URLs Detection View project Evaluation of Tree Based Machine Learning Classifiers for Android Malware Detection. *Proceedings*, 11056 LNAI:377–385, 2018.

[8] Guanhong Tao, Zibin Zheng, Ziying Guo, and Michael R. Lyu. MalPat: Mining Patterns of Malicious and Benign Android Apps via Permission-Related APIs. *IEEE Transactions on Reliability*, 67(1):355–369, mar 2018.

[9] Zhuo Ma, Haoran Ge, Yang Liu, Meng Zhao, and Jianfeng Ma. A Combination Method for Android Malware Detection Based on Control Flow Graphs and Machine Learning Algorithms. *IEEE Access*, 7, 2019.

[10] L Zhao, D Li, G Zheng, W Shi 2018 IEEE 18th International, and undefined 2018. Deep Neural Network Based on Android Mobile Malware Detection System Using Opcode Sequences. *ieeexplore.ieee.org*, 2018.

[11] MK Alzaylaee, SY Yerima, S Sezer Computers Security, and undefined. DL-Droid: Deep learning based android malware detection using real devices. *Elsevier*, 2020.

[12] Abdurrahman Pektaş and Tankut Acarman. Learning to detect Android malware via opcode sequences. *Neurocomputing*, 396:599–608, jul 2020.

[13] Taeguen Kim, Boojoong Kang, Mina Rho, Sakir Sezer, and Eul Gyu Im. A multimodal deep learning method for android malware detection using various features. *IEEE Transactions on Information Forensics and Security*, 14(3), 2019.

[14] Tonton Hsien De Huang and Hung Yu Kao. R2-D2: ColoR-inspired Convolutional NeuRal Network (CNN)-based AndroiD Malware Detections. In *Proceedings - 2018 IEEE International Conference on Big Data, Big Data 2018*, pages 2633–2642. Institute of Electrical and Electronics Engineers Inc., jan 2019.

[15] Somayyeh Fallah and Amir Jalaly Bidgoly. Benchmarking machine learning algorithms for android malware detection. *Jordanian Journal of Computers and Information Technology*, 5(3):216–230, 2019.

[16] J McGiff, WG Hatcher, and J Nguyen. Towards multimodal learning for android malware detection. *ieeexplore.ieee.org*.

[17] Mohammed M Alani and Senior Member. PAIRED : An Explainable Lightweight Android Malware Detection System. *IEEE Access*, 10(June):73214–73228, 2022.

[18] Tao Peng, Bochao Hu, Junping Liu, Junjie Huang, Zili Zhang, Ruhan He, and Xinrong Hu. A Lightweight Multi-Source Fast Android Malware Detection Model. *Applied Sciences*, 12(11):5394, 2022.

[19] Jiawei Xu and Lingyun Ying. SeqNet: An Efficient Neural Network for Automatic Malware Detection. 2022.

[20] Kavita Jain and Mayank Dave. Machine Learning-Based Lightweight Android Malware Detection System with Static Features. *Lecture Notes in Electrical Engineering*, 694:345–359, 2021.

[21] Li Deng, Geoffrey Hinton, and Brian Kingsbury. New types of deep neural network learning for speech recognition and related applications: an overview. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8599–8603. IEEE, may 2013.

[22] Joan Bruna and Stephane Mallat. Invariant scattering convolution networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1872–1886, 2013.

[23] Niall McLaughlin, Jesus Martinez Del Rincon, Boo Joong Kang, Suleiman Yerima, Paul Miller, Sakir Sezer, Yeganeh Safaei, Erik Trickel, Ziming Zhao, Adam Doupe, and Gail Joon Ahn. Deep android malware detection. In *CODASPY 2017 - Proceedings of the 7th ACM Conference on Data and Application Security and Privacy*, pages 301–308, New York, NY, USA, mar 2017. Association for Computing Machinery, Inc.

[24] Xibin Dong, Zhiwen Yu, Wenming Cao, Yifan Shi, and Qianli Ma. A survey on ensemble learning. *Frontiers of Computer Science*, 14(2):241–258, 2020.

[25] Tianchong Gao, Wei Peng, Devkishen Sisodia, Tanay Kumar Saha, Feng Li, and Mohammad Al Hasan. Android Malware Detection via Graphlet Sampling. *IEEE Transactions on Mobile Computing*, 18(12):2754–2767, dec 2019.

[26] Quan Sun and Bernhard Pfahringer. Bagging Ensemble Selection. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7106 LNAI:251–260, 2011.

[27] Jafar Tanha, Yousef Abdi, Negin Samadi, Nazila Razzaghi, and Mohammad Asadpour. Boosting methods for multi-class imbalanced data classification: an experimental review. 2020.

[28] Mohammad Amini, Jalal Rezaeenoor, and Esmaeil Hadavandi. Effective Intrusion Detection with a Neural Network Ensemble Using Fuzzy Clustering and Stacking Combination Method. *Journal of Computing and Security*, 1(4):293–305, 2014.

[29] Matilda Rhode, Pete Burnap, and Kevin Jones. Distillation for run-time malware process detection and automated process killing. feb 2019.

[30] David Sean Keyes, Beiqi Li, Gurdip Kaur, Arash Habibi Lashkari, Francois Gagnon, and Frederic Massicotte. EntropLyzer: Android Malware Classification and Characterization Using Entropy Analysis of Dynamic Characteristics. In *2021 Reconciling Data Analytics, Automation,*

Privacy, and Security: A Big Data Challenge (RDAAPS), pages 1–12. IEEE, may 2021.

Somayeh Mozafari received her B.Sc. degree in computer engineering from Shiraz Azad University, Iran, in 2010. She received her M.Sc. degree in Information Technology from Qom University in 2022. Her research interests include artificial intelligence and android malware detection.

Amir Jalaly Bidgoly received his M.Sc. degree in Software Engineering from the Iran University of Science and Technology (IUST) in 2009, and Ph.D. in Software Engineering from the University of Isfahan (Isfahan, Iran) in 2015. He is currently an Associate Professor with the Department of Computer Engineering at the University of Qom. His research interests include computer security and machine learning.