2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress

Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning

Samaneh Mahdavifar*, Andi Fitriah Abdul Kadir†, Rasool Fatemi*, Dima Alhadidi‡, Ali A. Ghorbani*
*Canadian Institute for Cybersecurity (CIC), Faculty of Computer Science, University of New Brunswick, Fredericton, NB, Canada, *(smahdavi, mfatemi, ghorbani)@unb.ca*
†Kulliyyah of Information & Communication Technology, International Islamic University Malaysia, Kuala Lumpur, Malaysia, *andifitriah@iium.edu.my*
‡School of Computer Science, University of Windsor, Windsor, ON, Canada, *dima.alhadidi@uwindsor.ca*

***Abstract*—Due to the significant threat of Android mobile malware, its detection has become increasingly important. Despite the academic and industrial attempts, devising a robust and efficient solution for Android malware detection and category classification is still an open problem. Supervised machine learning has been used to solve this issue. However, it is far to be perfect because it requires a significant amount of malicious and benign code to be identified and labeled beforehand. Since labeled data is expensive and difficult to get while unlabeled data is abundant and cheap in this context, we resort to a semi-supervised learning technique for deep neural networks, namely pseudo-label, which we train using a set of labeled and unlabeled instances. We use dynamic analysis to craft dynamic behavior profiles as feature vectors. Furthermore, we develop a new dataset, namely CICMalDroid2020, which includes 17,341 most recent samples of five different Android apps categories: Adware, Banking, SMS, Riskware, and Benign. Our offered dataset comprises the most complete captured static and dynamic features among publicly available datasets. We evaluate our proposed model on CICMalDroid2020 and conduct a comparison with Label Propagation (LP), a well-known semi-supervised machine learning technique, and other common machine learning algorithms. The experimental results show that the model can classify Android apps with respect to malware category with F_1-Score of 97.84 percent and a false positive rate of 2.76 percent, considerably higher than LP. These results demonstrate the robustness of our model despite the small number of labeled instances.**

***Index Terms*—Malware, Category Classification, Android, Dynamic Analysis, Semi-Supervised Learning, Deep Learning, Dynamic Behavior Profiles.**

I. Introduction

Android mobile devices have become ubiquitous, with trends showing that the recent pace of adoption is unlikely to slow down. Android popularity has its cost. The popularity has spurred an alarming growth in Android malware. Although many malware detection systems have been proposed [1]–[3], there are common limitations to the existing solutions. First, detecting malicious apps is not enough anymore. Specifying the category of Android malware is an important step to understand exactly the threat that we are vulnerable to. The additional information about the threat can be of great benefit to prioritizing mitigation techniques. Second, the static analysis uses statically-extracted signatures (behaviors/features) to detect and analyze malware [4]–[6]. Static-based malware detection is effective against most common types of malware but is ineffective against advanced malware programs that utilize sophisticated evasion detection techniques such as polymorphism (modifying code to defeat signature-based tools) or obfuscation (hiding evidence of malicious activities). That is why the detection rate of some static-based malware detection approaches decreased when analyzing recent malware instances. In contrast with the static analysis, the dynamic analysis uses a behavior-based approach to determine the functionality of the malware by studying the actions performed by the given malware after executing it in a sandbox environment. Third, machine learning techniques to detect malware depend on several features of both malicious and benign software. Accordingly, they need a high number of labeled instances. For a real-world problem such as malware analysis, it is quite challenging to label data because it is a time-consuming process, and in some cases, pieces of malware can avoid detection. The cost associated with the labeling process thus may render a fully labeled training set infeasible, whereas the acquisition of unlabeled data is relatively inexpensive. Semi-supervised learning is one type of machine learning technique that is very useful when a limited amount of labeled data exists for each class. Forth, machine learning models have shown to be insufficient to solve real-world problems with intrinsic complexity and massive amounts of data since they depend on a manual feature extraction process. On the other hand, deep learning algorithms take care of extracting abstract and flexible features automatically from raw data that help generalization in the classification.

In this paper, we propose a simple, yet practical and efficient framework for Android malware category classification based on mining system call-centric dynamically observed behaviors which are largely broken down into three categories of system calls, basic binders, and composite behaviors. The framework is based on a deep neural network that takes as input dynamically observed behaviors crafted by dynamic analysis to enable malware category classification by applying a semi-supervised technique. To the best of our knowledge, this is the first work that applies a semi-supervised deep neural network for dynamic malware category classification. Specifically, our contributions can be summarized as follows:

- We generate and release a dataset, namely CICMal-

978-1-7281-6609-4/20/$31.00 ©2020 IEEE
DOI 10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00094

Droid2020, that has the following four properties: (1) *Big*. It has more than 17,341 Android samples. (2) *Recent*. It includes recent and sophisticated Android samples until 2018, (3) *Diverse*. It has samples spanning between five distinct categories: Adware, Banking, SMS malware, Riskware, and Benign, and (4) *Comprehensive*. It includes the most complete captured static and dynamic features compared with publicly available datasets.

- We adopt CopperDroid, a Virtual Machine Introspection (VMI)-based dynamic analysis system [7], to automatically reconstruct the Inter-Process Communication (IPC) and Remote Procedure Call (RPC) interactions as well as complex Android objects by detailing the invocation of system calls.
- We analyze the dynamically observed behaviors of malware samples, including sequences of system calls, basic binders, and composite behaviors using neural networks. We use a deep neural network that takes as input the frequencies of the dynamic behaviors to enable malware category classification by applying a semi-supervised technique. For this purpose, we utilize pseudo-label [8], which is an efficient method for training deep neural networks in a semi-supervised fashion.
- We analyze the results of experiments and find that the model can detect and categorize malware with an F_1-Score of 97.84 percent and a false positive rate of 2.76 percent. The numbers demonstrate the robustness of our model despite the small number of labeled instances. Furthermore, we show that our semi-supervised deep learning model significantly outperforms LP, which is a popular semi-supervised machine learning algorithm, and other common machine learning algorithms such as Random Forest (RF), Decision Tree (DT), Support Vector Machine (SVM), and k-Nearest Neighbor (k-NN). Moreover, it is superior to a supervised deep neural network in the existence of only $(1\% - 10\%)$ of labeled training samples.

The rest of the paper is organized as follows. Related work is discussed in Section II. In Section III, we present Pseudo-label algorithm. Our proposed framework is detailed in Section IV. Section V discusses the performance analysis. Finally, Section VI concludes the paper and includes some future research directions.

II. Related Work

The rising number of Android malware has become a concern for both academia and industry. Machine learning algorithms such as Naive Bayes (NB), SVM, DT, and k-NN are commonly used by academic and industrial researchers in detecting Android malware [2], [18]. Deep learning, on the other hand, is a branch of machine learning that attempts to study high-level features effectively and efficiently from the original data [19]. In this section, we review some of the attempts that focused on deep learning algorithms in Android malware detection.

One of the first attempts was conducted by Yuan *et al.* [9] where they utilized more than 200 dynamic and static features for malware binary detection. The proposed framework employed Deep Belief Network (DBN) based on stacked Restricted Boltzmann Machine (RMB) in two phases of unsupervised pre-training and supervised fine-tuning. In 2016, Hou *et al.* [6] proposed DroidDelver, an Android malware detection system using DBN Based on API call blocks. They categorized the API calls of the Smali code which belong to the same method into a block. Another work by Nix *et. al.* [10] focused on the family classification of Android malware and applications using system API-call sequences and studied the effectiveness of Convolutional Neural Networks (CNNs) and Long Short Term Memory (LSTM). In another approach, Huang *et al.* [11] aimed of automating Android feature extraction where they transformed Android app bytecode into Red, Green, Blue (RGB) color code. The results of encoded color images are used as input features to a CNN that is trained with over one million malware samples and one million benign samples. In a similar way, Wang *et al.* [12] presented a hybrid model based on deep AutoEncoder (AE). CNN has been proposed to detect Android malware in which a deep AE is used as a pre-training method for a CNN. Karbab *et al.* [13] proposed an Android malware detection and family attribution framework in which they extract raw sequences of API calls from dex assembly code. The vectorized API calls are then trained using a CNN. In [15], important values of an Android Package kit (APK) code are visualized on an image and then fed into a CNN to detect mutated Android malware. Xiao *et al.* [14] considered one system call sequence of an Android malware as a sentence and applied two LSTM language models to train malware and benign samples. To classify an APK under analysis, they measure the similarity score of the sequence using the two trained networks. A more recent paper in 2019 [5] proposed a multimodal deep learning method that employed seven features extracted by analyzing Android files to be used in Android malware detection.

In contrast to the previous studies as summarized in Table I, we propose a simple, yet effective and efficient framework for Android malware category classification using deep neural networks that resort to dynamic analysis to avoid obfuscation and semi-supervised learning to avoid expensive labeling process. There are some studies [16], [17] that target the semi-supervised learning for malware detection using machine learning techniques, however, our approach utilizes deep learning to classify Android malware into different categories (Table I).

III. Pseudo-label

Pseudo-label [8] is an efficient method for training deep neural networks in a semi-supervised fashion. In this approach, for every weight update, the class which has the maximum predicted probability is selected as the labels for unlabeled data. Then the deep network is trained with labeled and unlabeled data simultaneously in a supervised way. It is equivalent to Entropy Regularization that aims at creating a decision boundary in low-density regions by minimizing the conditional

TABLE I
RELATED WORK SUMMARY OF ANDROID MALWARE DETECTION

Year	Authors	Analysis Type	Deep Model Type	Functionality
2014	Yuan *et al.* [9]	Static and dynamic	Supervised	Binary Detection
2016	Hou *et al.* [6]	Static	Supervised	Binary Detection
2017	Nix *et. al.* [10]	Static	Supervised	Detection and Family Classification
2018	Huang *et. al.* [11]	Static	Supervised	Binary Detection
2018	Wang *et al.* [12]	Static	Supervised	Binary Detection
2018	Karbab *et al.* [13]	Static	Supervised	Detection and Family Classification
2019	Kim *et al.* [5]	Static	Supervised	Binary Detection
2019	Xiao *et al.* [14]	Dynamic	Supervised	Binary Detection
2019	Yen *et al.* [15]	Static	Supervised	Binary Detection
2020	Our approach	Dynamic	Semi-supervised	Detection and Category Classification
Year	**Authors**	**Analysis Type**	**Semi-supervised Model**	**Functionality**
2015	Ma *et al.* [16]	Static	Machine Learning	Binary Detection
2017	Chen *et al.* [17]	Dynamic	Machine Learning	Binary Detection
2020	Our approach	Dynamic	Deep Learning	Detection and Category Classification

entropy of class probabilities for unlabeled data. Pseudo-label algorithm comprises of three main stages: (a) training the model with labeled data, (b) using the trained model in stage one to predict labels for the unlabeled data, and (c) re-training the model using labeled and unlabeled data simultaneously in a supervised fashion. Let x be an unlabeled sample of an initial dataset T, pseudo-label of x, y'_i, is calculated by picking up the class with maximum probability as follows [8]:

$$y'_i = \begin{cases} 1 & \text{if } i = \arg\max_{i'} p_{i'}(x) \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

The objective function that should be minimized in the fine-tuning stage is defined as:

$$L = \frac{1}{n}\sum_{m=1}^{n}\sum_{i=1}^{C} L(y_i^m, p_i^m) + \alpha(t)\frac{1}{n'}\sum_{m=1}^{n'}\sum_{i=1}^{C} L({y'}_i^m, {p'}_i^m), \quad (2)$$

where n and n' are the number of labeled and unlabeled samples of each mini-batch in Gradient Descent, respectively. C is the number of classes. p_i^m and y_i^m are the output units and label of sample m in labeled data, respectively, and ${p'}_i^m$ and ${y'}_i^m$ are the output units and pseudo-label of sample m in unlabeled data, respectively. To prevent the optimization process from getting stuck in poor local minima, $\alpha(t)$ is gradually increasing using a deterministic annealing procedure [8].

IV. PROPOSED FRAMEWORK

In this section, we detail the steps followed to dynamically detect and classify Android apps into five categories, as shown in Fig. 1.

A. Data Collection

We managed to collect more than 17,341 Android samples from several sources including VirusTotal service, Contagio security blog [20], AMD [3], and other datasets used by recent research contributions [13], [21], [22]. The samples were collected from December 2017 to December 2018. It is significant for cybersecurity researchers to classify Android apps with respect to the malware category for taking proper countermeasures and mitigation strategies. Hence, our dataset is intentionally spanning between five distinct categories: Adware, Banking malware, SMS malware, Riskware, and Benign.

To foster more research in this area, we release the accumulated dataset (CICMalDroid2020) to the research community [1]. Each malware category is briefly described as follows:

- **Adware.** Mobile Adware refers to the advertising material (i.e., ads) that typically hides inside the legitimate apps which have been infected by malware (available on the third-party market). Because the ad library used by the malware repeats a series of steps to keep the ads running, Adware continuously pops up ads (even if the victim tries to force-close the app). Adware can infect and root-infect a device, forcing it to download specific Adware types and allowing attackers to steal personal information.
- **Banking Malware.** Mobile Banking malware is a specialized malware designed to gain access to the user's online banking accounts by mimicking the original banking applications or banking web interface. Most of the mobile Banking malware are Trojan-based, which is designed to infiltrate devices, to steal sensitive details, i.e., bank login and password, and to send the stolen information to a command and control (C&C) server [21].
- **SMS Malware.** SMS malware exploits the SMS service as its medium of operation to intercept SMS payload for conducting attacks. The attackers first upload malware to their hosting sites to be linked with the SMS. They use the C&C server for controlling their attack instructions, i.e., send malicious SMS, intercept SMS, and steal data.
- **Mobile Riskware.** Riskware refers to legitimate programs that can cause damage if malicious users exploit them. Consequently, it can turn into any other form of malware such as Adware or Ransomware, which extends functionalities by installing newly infected applications. Uniquely, this category only has a single variant, mostly labeled as "Riskware" by VirusTotal.
- **Benign.** All other applications that are not in categories above are considered benign which means that the appli-

[1] https://www.unb.ca/cic/datasets/maldroid-2020.html

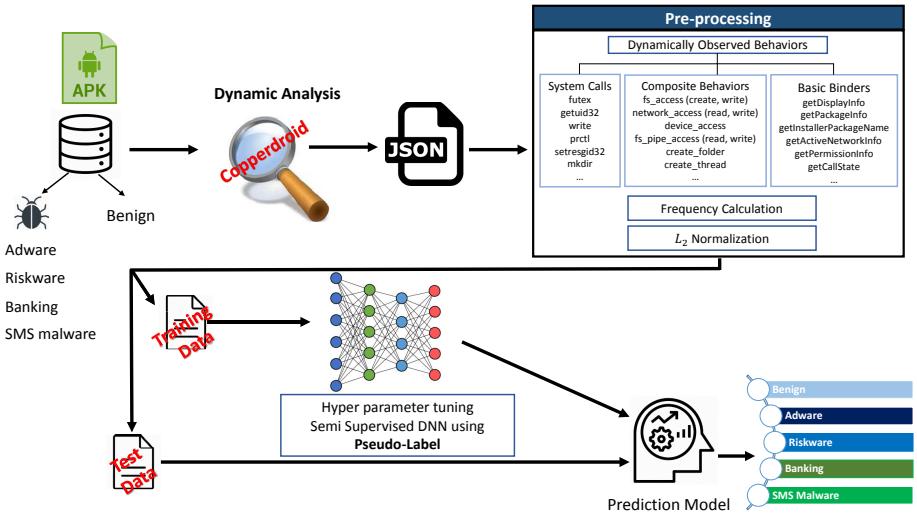


Fig. 1. An Overview of the Proposed Android Malware Semi-Supervised Categorization.

cation is not malicious. To verify the maliciousness, we scanned all the benign samples with VirusTotal.

B. Data Analysis

We analyzed our collected data dynamically using CopperDroid [7], a VMI-based dynamic analysis system, to automatically reconstruct low-level OS-specific and high-level Android-specific behaviors of Android samples. Out of 17,341 samples, 13,077 samples ran successfully while the rest failed due to errors such as time-out, invalid APK files, and memory allocation failures. All the APK files are first executed in CopperDroid and the run-time behaviors are recorded in log files. The output analysis results of CopperDroid are available in JSON format for easy parsing and additional auxiliary information. They include statically extracted information, e.g, intents; permissions and services; frequency counts for different file types; incidents of obfuscation, and sensitive API invocations; dynamically observed behaviors which are largely broken down into three categories of system calls, binder calls and composite behaviors; and the PCAP of all the network traffic captured during the analysis [7].

C. Pre-processing

Due to the high resiliency of system calls against malware evasion techniques and their platform independence nature, we employed dynamically observed behaviors of Android samples, which are being reconstructed from the system level observation point. Since system call traces alone would not disclose enough valuable insights into malware behavior, they have been combined with binder information and automatic complex Android object reconstruction.

In the pre-processing module, we first extracted three categories of dynamically observed behaviors including system calls, binder calls, and composite behaviors from the captured log files that assist us in automatically reconstructing system call semantics, including IPC, RPC, and complex Android objects. Binder is a base class used for realizing an optimized IPC and lightweight RPC mechanism. Composite behaviors such as *fs_access(create, write)*, *network_access(read, write)*, and *fs_pipe_access(read, write)* aggregate commonly associated low-level system calls. The following are the binder calls, namely *getDisplayInfo, registerCallback* and composite behavior *fs_access(write)* that are obtained from the CopperDroid analysis in the JSON format.


```
{
  "method": "getDisplayInfo",
  "interfaceGroup": "android.hardware.display",
  "low": [
    {
      "methodName": "android.hardware.display.
          IDisplayManager.getDisplayInfo(displayId = 0)",
      "type": "BINDER",
      "id": 414,
      "ts": "1526579390.815"
    }
  ],
  "interface": "android.hardware.display.IDisplayManager",
  "class": "BINDER",
  "arguments": [
    "displayId = 0"
  ]
},
{
  "method": "registerCallback",
  "interfaceGroup": "android.hardware.display",
  "low": [
    {
      "methodName": "android.hardware.display.
          IDisplayManager.registerCallback(callback = [
          IDisplayManagerCallback N/A])",
      "type": "BINDER",
      "id": 416,
      "ts": "1526579390.816"
    }
  ],
  "interface": "android.hardware.display.IDisplayManager",
  "class": "BINDER",
  "arguments": [
    "callback = [IDisplayManagerCallback N/A]"
  ]
},
```

```
{
"classType": 0,
"operationFlags": 2,
"procname": "com.cheaptravelnetwork.cheapflights",
"low": [
  {
    "sysname": "open",
    "blob": "{'flags': 131074, 'mode': 1, 'filename': u'/
        sys/qemu_trace/process_name'}",
    "type": "SYSCALL",
    "id": 471,
    "ts": "1530103411.788"
  },
  {
    "sysname": "write",
    "xref": 471,
    "ts": "1530103411.788",
    "blob": "{'size': 36L, 'filename': u'/sys/qemu_trace/
        process_name'}",
    "type": "SYSCALL",
    "id": 472
  },
  {
    "sysname": "close",
    "xref": 471,
    "ts": "1530103411.788",
    "blob": "{'filename': u'/sys/qemu_trace/process_name'
        }",
    "type": "SYSCALL",
    "id": 473
  }
],
"tid": 1162,
"class": "FS ACCESS(WRITE)"
},
```


To extract the feature vectors, we first parse the JSON file related to each Android APK to extract the system calls, binders, and composite behaviors that are invoked as the APK is run in CopperDroid. After that, we create feature vectors of size 470 for each APK file by putting all distinct low-level behaviors of all APK files together and setting each feature to the frequency of invocation of that particular behavior. The feature vectors are then normalized into values between $[0, 1]$ using ℓ_2 normalization method which scales each feature vector such that the square root of the sum of squares of all the values, i.e., vector's ℓ_2-norm, equals one. Let $x = (x_1, x_2, \cdots, x_n)$ be a vector in the n-dimensional real vector space $\mathbb{R}^n$, the ℓ_2-norm of vector x, denoted by $|x|$, is defines as $|x| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$.

D. Training

In the training module, the normalized feature vector is fed into a deep neural network[2] which is then trained with pseudo-label algorithm. Using pseudo-label, the deep neural network is trained in a supervised fashion with labeled and unlabeled data simultaneously. Pseudo-labels that are recalculated every weight update are used along with the real labels to optimize the supervised loss function in each mini-batch.

E. Detection

The detection module takes the classification model and test data and classifies the input feature vector into one of the five categories. Finally, the corresponding classification measures are calculated and reported as output.

[2]In this paper, a deep neural network is basically a Feed-Forward Neural Network (FFNN).

V. Performance Analysis

In this section, we evaluate how *effectively* and *efficiently* the semi-supervised deep learning-based framework can classify Android APK files into one of the pre-defined categories.

A. Environment

We implemented the proposed semi-supervised deep learning-based framework using a PC with 3.60 GHz Core i7-4790 CPU and 32 GB RAM. We had the Tensorflow implementation of pseudo-label in Python, which is fast and robust [23], and we revised the code based on our needs for pre-processing, batch processing, and training convergence. The input layer of the deep neural network consists of 470 neurons, with reference to the number of input feature vectors as detailed in Section IV, and the output layer consists of five neurons equivalent to the number of categories. However, the number of hidden layers and hidden neurons in each layer need to be fine-tuned. We used the Sigmoid function as the activation function for all layers and Mini Batch Gradient Descent as the optimization algorithm. The mini-batch size was set to 100 for labeled samples, and the initial learning rate was set to 0.4. These parameters were determined using hyper-parameter tuning. As mentioned previously in Section III, the balancing coefficient, $\alpha(t)$ was slowly increased to dynamically adjust the network performance with the parameter settings of $\alpha_f = 1.5$, $T_1 = 100$, and $T_2 = 400$ (We refer the reader to [8] for more details about these parameters). We trained each deep network model until convergence or the number of 800 epochs was reached. We ran each set of experiments 20 times, and computed the average over all runs to find the evaluation metrics. To make sure that the training samples in each mini-batch are randomly selected, we shuffled the entire dataset at the beginning of each epoch.

B. Dataset

As discussed earlier in Section IV, we ran 17,341 APKs in CopperDroid to observe the behavior of each sample. Out of this number of samples, 13,077 ran successfully while the rest failed because of different error types including but not limited to bad unpack parameters, bad ASCII characters, bad CRC-32 values, and bad UTF-8 bytes. Then we loaded all analysis results where about 12% of the JSON files failed to be opened mostly due to "unterminated string". The final remaining Android samples in each category are as follows: Adware (1,253), Banking (2,100), SMS malware (3,904), Riskware (2,546), and Benign (1,795) (Table II).

TABLE II
Dataset Distribution

	Category					
	Adware	**Banking**	**SMS**	**Riskware**	**Benign**	**Total**
#Sample	1,253	2,100	3,904	2,546	1,795	11,598

Table III provides a comparison among publicly available Android malware datasets and our generated dataset based

on multiple criteria, such as size, age, and captured static or dynamic features. Apparently, CICMalDroid2020 dataset is the most recent and quite diverse in terms of Android malware categories. Regarding the broad range of static and dynamic features captured, CICMalDroid2020 dataset owns a fairly large collection of Android samples.

C. Experimental Results

In our experiments, we used recision (PR), recall (RC), F_1-Score (F_1), accuracy (ACC), false positive rate (FPR), false negative rate (FNR), and classification error to assess the overall classification performance. A false positive is a benign APK being detected as malicious, and a false negative is a malicious APK being detected as benign. To acquire the best neural network architecture, we tested our semi-supervised deep network with multiple numbers of hidden layers and neurons in each layer. We also examined the classification performance of our proposed framework under different train/test split ratios. Afterward, we picked the best architecture and train/test split ratio $(70\% - 30\%)$ and compared the performance of the semi-supervised deep neural network with the supervised deep neural network and some common machine learning algorithms including the semi-supervised approach (LP) while changing the number of labeled samples. We also reported the confusion matrix of our semi-supervised deep network per app category. Finally, we estimated the average runtime of the prediction stage.

1) Malware Classification Performance: We investigated the impact of neural network architecture on the classification performance of our malware detector. Table IV compares the test classification metrics of Pseudo-Label Deep Neural Network (PLDNN) with different numbers of hidden layers and hidden neurons. For this set of experiments, we set the number of labeled samples and train-test ratio to 1000 and $(70\% - 30\%)$, respectively. Even though some architectures demonstrate unsatisfactory FPR or FNR, all architectures produce an F_1-Score of higher than 94% that justifies the robustness of our proposed semi-supervised deep learning in existence of about 10% of the labeled training samples (1000). The deep network with seven layers and neurons of [470 400 250 150 30 10 5] is superior to other deep networks in terms of nearly all measures, and from now on, it is used as the deep network architecture for the subsequent experiments.

Table V compares the classification error of PLDNN with that of basic Deep Neural Network (DNN), semi-supervised approach LP, and four common machine learning classifiers, namely RF, DT, SVM, and k-NN on the Android malware dataset with 100, 300, 500, 1000, 5000, and all labeled training samples. We applied stratified 5-fold cross-validation for all machine learning algorithms. The results presented in the table show that by increasing the number of labeled samples the classification error of all algorithms decreases. For any number of labeled training samples, PLDNN remarkably outperforms other classifiers, particularly LP which is a well-known semi-supervised machine learning algorithm. Apparently, despite the small number of labeled samples, PLDNN achieves an acceptable classification error. For instance, for approximately 1% of the total number of labeled training samples (100 labeled samples), PLDNN achieves an ACC of 91.63% for malware categorization, justifying its stability even in the presence of scarce labeled data. For a higher amount of labeled samples, e.g., 5000, the gap between classification error of DNN and PLDNN is almost negligible and is decreased to 2.77% and 2.6% for DNN and PLDNN, respectively.

Table VI shows the confusion matrix of PLDNN for each Android app category. We have set the number of labeled training samples to 1000. We listed true positive rate (TPR), i.e., the fraction of predicted apps with respect to the total number of apps for each category. The diagonal illustrates the correct classification. SMS malware achieves a TPR of 100% while Riskware and Benign obtain 98% each. The highest rate of misprediction made for Adware which has been mostly mistaken for Benign (7%) and Riskware (5%). Adware apps inherently have high similarities with Benign and Riskware. Therefore, it makes sense to see the overlap between these three categories. Furthermore, the total number of Adware samples successfully were analyzed by CopperDroid is lower than in other categories. Although we tend to evenly distribute the samples to each category before the training stage, there exists a less diversified number of samples to be selected in each run of the algorithm for smaller categories. Overall, the figures of the confusion matrix demonstrate that PLDNN precisely discovers the underlying behavior of the Android malware with only 1000 labeled training samples. This shows that regardless of the small number of labeled samples, our semi-supervised approach exhibits highly competitive performance for malware categorization.

In Fig. 2, the average training cost of DNN and PLDNN are compared according to the number of iterations. The

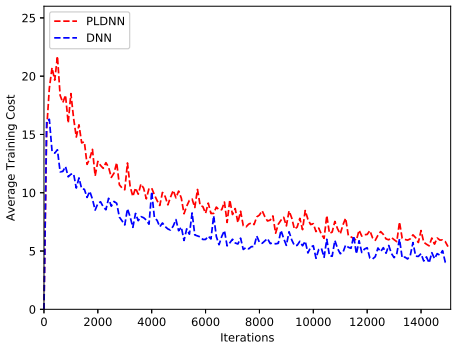


Fig. 2. Average Training Cost of PLDNN and DNN vs Total Number of Iterations

number of iterations is calculated by multiplying the number of epochs (1,500) and the number of labeled mini-batch (10), i.e., 15,000. As expected, the average cost of both methods constantly decreases as the number of iterations increases. However, the first iterations have massive improvements, but

TABLE III
COMPARISON AMONG PUBLICLY AVAILABLE ANDROID MALWARE DATASETS

Dataset Name	Published Year	Samples Collected	Size	Family/ Category	Captured Static Features							Captured Dynamic Features					
					Intents	Method Tags	Permissions	API Calls	File Types	Obfuscation	Components	Sys. Calls	Binder Calls	Composite Behaviors	API Calls	Network	Log
Genome [24]	2012	2011	1,260	Family	No	No	Yes	No	No	No	Yes	No	No	No	No	No	No
Drebin [1]	2014	2012	129,013	Family	Yes	No	Yes	Yes	No	No	Yes	No	No	No	No	No	No
AndroTracker [4]	2015	2013	55,733	Family	Yes	No	Yes	Yes	No	No	No	No	No	No	No	No	No
AndroProfiler [25]	2016	2013	9,483	Family	No	No	No	No	No	No	No	Yes	No	No	No	No	Yes
AMD [3]	2017	2016	24,650	Family	No	No	No	Yes	No	No	Yes	No	No	No	No	No	No
Maldozer [13]	2018	2016	71,000	Family	No	No	No	Yes	No	No	No	No	No	No	No	No	No
InvesAndMal [26]	2019	2017	5,491	Family & Category	Yes	No	Yes	No	No	No	No	No	No	No	Yes	Yes	No
CICMalDroid	**2020**	**2018**	**13,077**	**Category**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**No**	**Yes**	**Yes**	**Yes**	**No**	**Yes**	**Yes**

TABLE IV
CLASSIFICATION METRICS (%) OF DIFFERENT ARCHITECTURES OF PLDNN WITH LABELED SAMPLES=1000.

Layers	Neurons	PR	RC	F_1	ACC	FPR	FNR
7	[470 400 300 200 100 30 5]	98.2	98	98.1	97.05	6.25	2.0
	[470 400 270 150 30 15 5]	97.03	97.03	97.03	95.98	6.18	2.97
	[470 400 250 150 30 15 5]	97.86	95.09	96.45	95.67	3.38	4.91
	[470 400 250 150 30 10 5]	**99.16**	**96.54**	**97.84**	**96.7**	**2.76**	**3.46**
	[470 400 250 100 30 10 5]	98.16	96.2	97.17	96.12	4.05	3.8
	[470 350 250 150 50 20 5]	98.81	95.42	97.09	95.96	2.74	4.58
	[470 350 250 100 50 10 5]	97.48	95.09	96.27	94.77	5.99	4.91
6	[470 400 200 100 30 5]	96.96	97.36	97.16	95.79	8.67	2.64
	[470 400 200 50 20 5]	97.26	97.04	97.15	95.73	8.22	2.96
	[470 300 200 30 10 5]	96.8	97.38	97.09	95.7	8.99	2.62
	[470 300 150 50 10 5]	96.99	96.08	96.53	95.06	7.51	3.92
5	[470 400 150 50 5]	96.1	93.23	94.64	92.61	8.84	6.77
	[470 400 150 10 5]	97.17	94.5	95.82	94	7.34	5.5
	[470 300 150 50 5]	99.4	90.84	94.93	93.25	1.26	9.16
	[470 300 100 10 5]	99.14	90.59	94.67	93.11	1.63	9.41
4	[470 350 100 5]	96.54	91.51	93.95	91.22	9.6	8.49
	[470 350 50 5]	96.49	92.86	94.64	92.01	10.69	7.14
	[470 300 50 5]	98.36	91.24	94.66	92.96	3.31	8.76

TABLE V
CLASSIFICATION ERROR (%) ON THE ANDROID MALWARE DATASET WITH 100, 300, 500, 1000, 5000, AND ALL LABELED TRAINING SAMPLES.

Method	Labeled Training Samples					
	100	300	500	1000	5000	All
RF	29.18	21.56	17.41	14.29	8.03	6.56
DT	35.12	25.37	20.81	17.93	10.61	9.25
SVM	56.47	46.41	41.62	37.39	24.94	21.9
K-NN	48.54	37.74	32.96	27.5	17.42	14.75
LP	57.23	53.42	53.25	51.5	19.6	16.94
DNN	10.82	9.44	7.93	3.52	2.77	2.4
PLDNN	8.37	6.93	4.54	3.3	2.6	2.4

after a while, the cost slightly changes and get stabilized. Besides, the average cost of training PLDNN is higher than DNN due to expanding loss function of labeled samples by adding $(\alpha(t)\frac{1}{n'}\sum_{m=1}^{n'}\sum_{i=1}^{C}L(y'^{m}_{i},p'^{m}_{i}))$ term to the overall loss function L in Eq. 2. As we proceed with training, this gap gradually shrinks, and the two diagrams approximately converge after 15,000 iterations.

2) Runtime Performance: In this section, we evaluated the efficiency of PLDNN by estimating the detection time. The

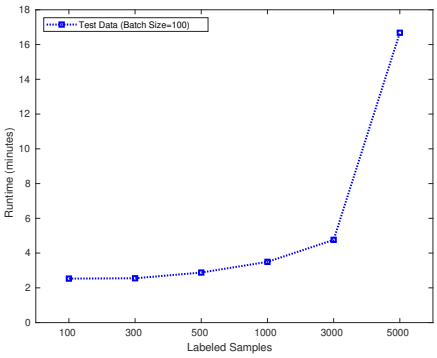
TABLE VI
CONFUSION MATRIX OF CATEGORY CLASSIFICATION

Real Category	Prediction Category				
	Adware	**Banking**	**SMS**	**Riskware**	**Benign**
Adware	0.85	0.02	0.02	0.05	0.07
Banking	0.0	0.96	0.03	0.01	0.0
SMS	0.0	0.0	1.0	0.0	0.0
Riskware	0.01	0.0	0.0	0.98	0.01
Benign	0.01	0.0	0.0	0.01	0.98

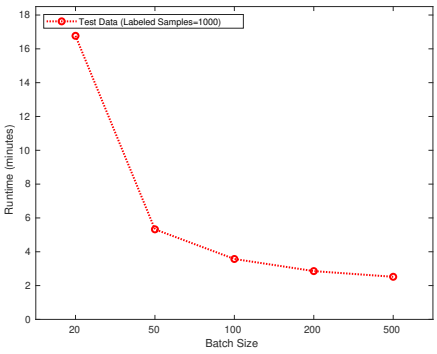
detection time is the total time needed to predict the category of the test samples, including normalization, training, and testing stages. The detection time of PLDNN for the test data based on different numbers of labeled samples and batch sizes are presented in Fig. 3(a) and 3(b), respectively. In Fig. 3(a), total runtime monotonically grows for the number of labeled samples between 100 to 3000, having values approximately between 2.5 to 4.5 minutes, and rises sharply for 5000 labeled samples (about 16.5 minutes). In contrast, the runtime in Fig. 3(b) follows an opposite trend and gradually declines by increasing the batch size. After the batch size of 20 (16.5 minutes), the runtime experiences a sudden drop to 5 minutes for a batch size of 50. Both runtime diagrams justify that PLDNN is very efficient, having an average prediction time of 60 milliseconds per sample for labeled samples=1000 and batch size=100. It is worth mentioning that the average prediction time per sample includes normalization, training, and testing stages, implying that the proposed Android malware detection system has a negligible testing runtime and could be easily deployed in computationally-limited devices.

VI. CONCLUSION

In this paper, we have proposed an effective and efficient Android malware category classification system based on semi-supervised deep neural networks. In spite of the small number of labeled training samples, the proposed detection system is effective and superior to supervised deep neural networks. This eliminates the need for a high number of labeled instances, which is very expensive to acquire in the domain of malware analysis. Additionally, it is efficient in terms of execution time, and it helps us to prioritize our mitigation



(a) Runtime vs Labeled Samples



(b) Runtime vs Batch Size

Fig. 3. Detection Runtime vs Number of Labeled Samples and Batch Size

techniques by specifying the category of the malware. We have offered a new 17,341 Android malware dataset which includes the most complete captured static and dynamic feature sets and spans between five distinct categories of Adware, Banking, SMS malware, Riskware, and Benign. As future work, we like to test if the proposed detection system can run on resource-limited IoT devices such as Raspberry Pi. Another avenue of research would be to utilize other types of features captured by CopperDroid. Our approach also could be enhanced using state-of-the-art deep learning models like RNN or CNN.

ACKNOWLEDGMENT

The authors would like to express their gratitude toward Dr. Lorenzo Cavallaro and Feargus Pendlebury (Systems Security Research Lab, King's College London) for generously analyzing a large number of Android APKs in CopperDroid.

REFERENCES

[1] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *NDSS*, 2014.

[2] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-aware Android malware classification using weighted contextual API dependency graphs," in *SIGSAC Conf. Comput. and Commun. Secur.* ACM, 2014, pp. 1105–1116.

[3] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current Android malware," in *Int. Conf. Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2017, pp. 252–276.

[4] H. Kang, J.-w. Jang, A. Mohaisen, and H. K. Kim, "Detecting and classifying Android malware using static analysis along with creator information," *Int. J. Distrib. Sens. N.*, vol. 11, no. 6, p. 479174, 2015.

[5] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for Android malware detection using various features," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 3, pp. 773–788, 2019.

[6] S. Hou, A. Saas, Y. Ye, and L. Chen, "DroidDelver: An Android malware detection system using Deep Belief Network based on API call blocks," in *Int. Conf. Web-Age Inf. Manage.* Springer, 2016, pp. 54–66.

[7] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, "CopperDroid: Automatic reconstruction of Android malware behaviors," in *NDSS*, 2015.

[8] D.-H. Lee, "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks," in *Workshop on challenges in representation learning, ICML*, vol. 3, 2013, p. 2.

[9] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-Sec: Deep learning in Android malware detection," in *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4. ACM, 2014, pp. 371–372.

[10] R. Nix and J. Zhang, "Classification of Android apps and malware using deep neural networks," in *2017 IEEE Int. Joint Conf. Neural Netw.* IEEE, 2017, pp. 1871–1878.

[11] T. Hsien-De Huang and H.-Y. Kao, "R2-d2: color-inspired Convolutional Neural Network (CNN)-based Android malware detections," in *IEEE Int. Conf on Big Data*. IEEE, 2018, pp. 2633–2642.

[12] W. Wang, M. Zhao, and J. Wang, "Effective Android malware detection with a hybrid model based on deep autoencoder and convolutional neural network," *J. Amb. Intel. Hum. Comp.*, pp. 1–9, 2018.

[13] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "MalDozer: Automatic framework for Android malware detection using deep learning," *Digit. Invest.*, vol. 24, pp. S48–S59, 2018.

[14] X. Xiao, S. Zhang, F. Mercaldo, G. Hu, and A. K. Sangaiah, "Android malware detection based on system call sequences and LSTM," *Multimed. Tools Appl.*, vol. 78, no. 4, pp. 3979–3999, 2019.

[15] Y.-S. Yen and H.-M. Sun, "An Android mutation malware detection based on deep learning using visualization of importance from codes," *Microelectron. Reliab.*, vol. 93, pp. 109–114, 2019.

[16] S. Ma, S. Wang, D. Lo, R. H. Deng, and C. Sun, "Active semi-supervised approach for checking app behavior against its description," in *2015 IEEE 39th Annu. Comput. Softw. and Appl. Conf.*, vol. 2. IEEE, 2015, pp. 179–184.

[17] L. Chen, M. Zhang, C.-Y. Yang, and R. Sahita, "Semi-supervised classification for dynamic Android malware detection," *arXiv preprint arXiv:1704.05948*, 2017.

[18] H. Cai, N. Meng, B. Ryder, and D. Yao, "DroidCat: Effective Android malware detection and categorization via app-level profiling," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 6, pp. 1455–1470, 2018.

[19] S. Mahdavifar and A. A. Ghorbani, "Application of deep learning to cybersecurity: A survey," *Neurocomputing*, vol. 347, pp. 149–176, 2019.

[20] "Contagio Mobile Malware Mini Dump," http://contagiominidump.blogspot.ca/, online; accessed 6 May 2019.

[21] A. F. A. Kadir, N. Stakhanova, and A. A. Ghorbani, "An empirical analysis of Android banking malware," *Protecting Mobile Networks and Devices: Challenges and Solutions*, p. 209, 2016.

[22] A. F. Abdul Kadir, N. Stakhanova, and A. Ghorbani, "Android botnets: What URLs are telling us," in *Network and System Security*, M. Qiu, S. Xu, M. Yung, and H. Zhang, Eds. Cham: Springer International Publishing, 2015, pp. 78–91.

[23] Nanazhu, "Pseudo label for deep neural networks," https://github.com/nanazhu/Pseudo-Label-for-Deep-Neural-Networks/blob/master/src/Pseudolabel.py, online; accessed 6 May 2019.

[24] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," in *2012 IEEE Symp. Secur. and Prv.* IEEE, 2012, pp. 95–109.

[25] J.-w. Jang, J. Yun, A. Mohaisen, J. Woo, and H. K. Kim, "Detecting and classifying method based on similarity matching of Android malware behavior with profile," *SpringerPlus*, vol. 5, no. 1, p. 273, 2016.

[26] L. Taheri, A. F. A. Kadir, and A. H. Lashkari, "Extensible Android malware detection and family classification using network-flows and api-calls," in *2019 Int. Carnahan Conf. Secur Technol.* IEEE, 2019, pp. 1–8.