Android Applications Classification with Deep Neural Networks

Mustapha Adamu Mohammed (adamu.mohammed@ktu.edu.gh)
Kwame Nkrumah University of Science and Technology

Michael Asante
Kwame Nkrumah University of Science and Technology

Seth Alornyo
Koforidua Technical University

Bernard Obo Essah
University of Ghana

Research Article

Keywords: mobile opearting system, android security, deep learning, Deep Belief Neural Network, benignware, andoid malware

Posted Date: October 27th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-2196025/v1

Android Applications Classification with Deep Neural Networks

Mustapha Adamu Mohammed[1,2*], Michael Asante[2†], Seth Alornyo[1,3†] and Bernard Obo Essah[2,3†]

[3,2*]Department of Computer Science, Koforidua Technical University, Ghana.
[2]Department of Computer Science, Kwame Nkrumah University of Science and Technology, Ghana.
[3]Department of Mathematics, University of Ghana, Ghana.

*Corresponding author(s). E-mail(s): adamu.mohammed@ktu.edu.gh;
Contributing authors: masante.csi@knust.edu.gh; sabigseth@ktu.edu.gh; boessah@st.ug.edu.gh;
†These authors contributed equally to this work.

Abstract

In an effort to foster the growth of Android in the mobile operating system market and keep current consumers, Google has made millions of applications, some of which are free and others of which are paid apps, available in the Google Play store. Users have, however, regularly complained that the store is full of malicious apps and low quality apps, putting their devices and personal information at risk. Detection of mobile applications vulnerabilities remains a significant challenge due to the constant evolution of methods to obfusticate and circumvent current detection and security schemes.The ability to correctly classify and categorize mobile applications, especially those built for Android, is crucial for separating malignant applications from benign ones thereby protecting the many more devices of unsuspecting users.This paper presents a deep neural network technique to classify android applications into legitimate and malware applications. Specifically,we first proposed applications classification model based on deep belief neural network classifier.The neural network was built and trained on real dataset to classify android-based applications using TensorFlow library and implemented on python programming language. We further trained and tested

our neural network's classification performance against that of four traditional deep feed-forward neural networks and seven baseline models based on machine learning algorithms on the same data. According to experimental results, a deep belief neural network-based model could accurately categorize Android apps into benign and malicious categories with 98.7% of the time. Compared to all previous deep learning and machine learning methods, this represents a significant improvement. Also,the categorization accuracy of the DBN model is better than that of numerous other models examined by earlier researchers.

Keywords: mobile opearting system;android security;deep learning; Deep Belief Neural Network;benignware; andoid malware.

1 Introduction

Mobile apps allow users to execute tasks that were previously only available on personal computers (PCs), thereby enhancing the functionality of smartphones.There are currently more than 6 billion cellphones, and 72% of them run the Android operating system [1].Thus,making Android the most popular mobile operating system on the market currently. In fact Android recorded over 2.5 billion active devices on monthly basis in the year 2019 alone. In fact,Android has become quite popular not only as an operating system for smartphones but also for cars, tablets, smart appliances, and IoT gadgets [2]. Android has made millions of apps easily accessible and downloadable in an effort to promote further growth and keep current customers. However, these apps frequently fall short of meeting even the most basic security standards and can be used to indirectly attack other managed or linked devices (such as Internet of things nodes) that carry out delicate tasks like controlling a smart automobile, opening a smart lock, or performing health checks.

Attackers would naturally choose a system with a large client base as their target, hence malware authors are making malwares that target Android applications. In fact, recent research revealed that 50% of currently accessible apps are susceptible to one or more serious vulnerabilities[3],[4],[5],[6]. In the future, there will be an increase in both the quantity and variety of malware that targets Android-based applications [7].

It is crucial to note that Android now has a security system that relies heavily on a sandbox and permissions system. Each Dalvic virtual machine in which a program runs has a distinct user ID issued to it. Application code executes independently of every other application code. In other words, one application cannot allow access to the files of another application..

Additionally, Android operates under the Least Privilege Principle: access to additional sensitive resources is restricted by the granting of certain permissions, while each installed application is only given the minimal capabilities necessary to ensure its operations.

Although the Android platform has these security models, side-loaded apps

and Google Play's open structure make application vulnerabilities frequent. That is,Google play does not verify uploaded applications mannually [8]. Even though google discourages users from side loading applications and third-party applications due to security concerns,some installations from other third party vendors are still permitted [9], [10].

Additionally,there are vulnerabilities associated with the adroid platfrom itself. Remote access tools (RAT), which are used to transmit messages, enable cameras for spying, and track GPS data, can access call history, online browser history, installed applications, and more on mobile targets.

However, due to the quick development of evasive techniques used by malware developers to undermine existing security measures, the discovery of vulnerabilities in Android applications remains a significant challenge. Social engineering tactics that are frequently exploited by attackers would be used to lure people. These include drive-by downloads, attachments, games, phishing, spam campaigns, and fake advertisements.

The classification algorithms for Android applications currently in use have some drawbacks. Even the more effective machine learning-based methods for classifying applications have variable recognition rates and high levels of computational complexity, which have an impact on software control and the monitoring of applications.

Our work proposes an Android application categorization model based on supervised deep neural network classifier to address the shortcomings of current mobile application classification technologies, fully learn the characteristics of mobile operating systems applications, and enhance the accuracy of android application classification. To obtain the best classification accuracy,we would subject our model through multiple stages, including data collection, preprocessing, training, and testing, a procedure also used by Wijaya et al.[11],in their study.

Hinton and Salakhutdinov [12], first presented the original Deep Belief Network model. Deep Belief Networks,also called graphical generative models address the automatic learning of features and also learn the important features that characterize a given dataset and get over the training's challenges by using the layer by ayer initialization technique. The application of the Deep Belief Neural network algorithm for characteristic recognition and categorization offers obviuos benefits [13].

The rest of this study is organized as follows: Section 2 discusses the prior research on android based applications cateorization using machine learning and deep neural networks. In section 3, we provide an overview of DBN and discuss how we built our DBN model using the Tensorflow library. Also, dataset creation and model parameter selection is discussed. In section 4,we implement our DBN android application model by training it on the datasets. Also,the test datasets are used to examine the classification accuracy of our model, and the outcome is compared to that of the plain neural network based model under the same conditions. section 5 presents the summary of the paper.

2 Related Work

The open handset alliance (OHA), an organization working to improve the android environment day by day, has offered many security solutions for Android platforms, including operating system improvements and software updates. Again,suggestions have been made to the effect that users download files and apps from reputable sites and sources. However these measures can only be practiced by IT savvy users. Naive users,who unfortunately are in the majority will always fall victims by not strictly observing these guidelines.The biggest traction in recent years has been achieved by new android based malware detection approaches based on machine learning techniques[14], such as ensemble clustering methods.

Akbar et al. [15], recently presented a permissions-based malware detection method that assesses an app's malice based on the use of suspecious permissions. The authors employed a multilevel technique by extracting the important features like permissions from over 10000 android apps.To classify the applications into their malicious or legitimate categories, the researchers used a variety of machine learning algorithms,including Support Vector Machine,Random Forest, Rotation Forest, and Naïve Bayes classifiers. Zhang et al.[16],subsequently proposed android malware detection by analyzing system call traces for legitimate and malicious android based apps. The authors trained six machine learning algorithms for the malware detection system.

Shatnawi et al,[17], Syrris et al.[18], Herron et al.[19], Islam et al.[20] and Raymond et al. [21],recently trained machine learning based models based on static features of android applications datasets to determine the presence of malignant applications in android systems. Much as machine learning based approaches have proven successful at detecting android malware experimentally as suggested by the above studies and many others. However,for the purpose of obtaining useful feature sets, machine learning models significantly rely on handcrafted feature engineering. Additionally, the majority of current datasets based on Android are unlabeled, which limits the ability of machine learning-based models to accurately categorize morden day android apps. Deep neural network approaches have recently been the subject of numerous studies on extending machine learning techniques for unlabelled data. For instance,Lakshmanarao et al.[22] recently targeted specific opcode sequences extracted from android applications dataset to train a recurrent neural network for malware classification. Fallah et al.[23] modeled an instance of traffic data as a series of flows by using long Short Term memory(LSTM) model for malware detection and classification into legitimate and malicious samples. Despite the fact that these most recent research (2022) seem to offer useful strategies for fending off new threats to Android platforms. Deep neural network techniques like LSTMs and RNNs, however, require a large quantities of training data in order to learn the intricate and nonlinear functions necessary to make accurate predictions. Therefore, these systems frequently underperform in predictions when the data is not large enough, rendering their methods unsuitable for generalization. To address these shortfalls,we train a deep belief

neural network (DBN) model on real dataset (CICInvesAndMal2019) collected from University of New brunswick(UNB) dataset repository to classify android applications that have never been seen before into malicious and legitimate categories. DBNs can be trained with a less amount of labeled data by stacking numerous Restricted Boltzmann Machines(RBMs)[24]. Since feature extraction is carried out unsupervised by multiple stacks of RBMs, a small collection is adequate to train a network suitable to classify android dataset into benign and malicious apps.

3 Methodology

The goal of this study is to categorize whether an app is a malware or benignware, hence the problem is defined as a classification one. Therefore, we define the Android application categorization problem, give a brief introduction to deep belief networks, and demonstrate how it models our data in this section. We then give a general overview of the pre-training process. Finally, we applied a model to our dataset using the TensorFlow framework.

Definition of Problem: Given a set of Android Application Package (APK) expressed as an input vector $V_L = \{v_1, v_2, v_3 \cdots v_n\}$,where each v_i is represented by a vector containing the values of n dimensional features.

Let $v_i = \{feature_1, feature_2, feature_3 \cdots feature_n, Output\_Label\}$

output_label $\in$ { normal APK, malicious APK } is the class label associated with the application. V would be utilized to train a deep neural network classifier to learn the characteristics of both normal APK and malicious APK. The goal of a trained deep neural network is to categorize a given unlabeled APK_unseen $= \{feature_1, feature_2, feature_3, feature_n, ???\}$, that has never been seen before by assigning a label, Output_Label where output_label $\in$ { normal APK, malicious APK }.

Deep Belief Neural Networks:

Deep Belief Networks (DBN), is a deep neural network that forms the origin of unsupervised layer wise pretraining procedure for deep feed forward neural networks. We build an entirely unsupervised generative model[25] that mixes undirected and directed interactions between the variables that constitute either the visible layer or the hidden layers. Figure 1 is a Deep belief

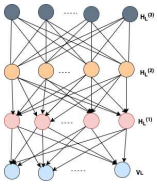


Fig. 1: A three hidden layer Deep Belief Network model

Network model with three hidden layers. As can be seen,we have undirectional interactions at the top layers and directed connections at the lower layers. In

a DBN,the top two layers will always form an RBM. Thus,the distribution, $p\left(H_L^{(1)}, H_L^{(3)}\right)$ of layer $H_L^{(2)}$ and layer $H_L^{(3)}$,is an RBM with undirected interactions. The other layers are going to form a Bayesian network with directed interactions. Specically the conditional distribution of the units given the layers above them takes the form :

$$p\left(H_{Lj}^{(1)} = 1|H_L^{(2)} = \sigma\left(b^{(1)} + W^{(2)^\top} + H_L^{(2)}\right)\right) \tag{1}$$

$$p\left(V_{(i)} = 1|H_L^{(2)} = \sigma\left(b^{(1)} + W^{(2)^\top} + H_L^{(2)}\right)\right) \tag{2}$$

This coresponds to a probabilistic model associated with the Logistic model[26]. That is : for the probability of a visible unit in a visible layer to be equal to 1, it is going to be the sigmoid (σ) applied on a linear transformation of the layer above it. For $H_L^{(1)}$ is going to be a linear transformation of $H_L^{(2)}$,and for the visible layer V_L, is going to be a linear transformation of $H_L^{(1)}$ as indicated in (1) and (2) respectively. When we have units that interact this way,we call such a model Sigmoid Belief Network(SBN)[27].

Generative process associated with our DBN model:

To generate (model) our android application data from our DBN,we applied Gibbs sampling procedure[28], between the top two layers over a number of iterations through the following algorithms:

Algorithm 1 Training algorithm based on Contrastive Divergence Method by Gibbs Sampling[29]

Initialize the training dataset
Set the states of the visible units to the training dataset
Positive_Phase $\leftarrow$ Update the Weights of all hidden units,using; $\sigma\left(B_j + \frac{m}{\sum_{i=1}^{n}} W_{ij}V_i\right)$
Negative_Phase $\leftarrow$ Update the Weights of all visible units,using; $\sigma\left(A_i + \sum_{j=1}^{n} W_{ij}H_j\right)$
while $W_{ji} \neq W_{ij}$ **do**
  if $W = n$, update the weights of all associated weights **then**
    $W_{ij} + L * \sigma\left(B_j + \sum_{i=1}^{m} W_{ij}V_i\right) - \sigma\left(A_i + \sum_{j=1}^{n} W_j H_i\right)$
  else if $W = 0$, Repeat over all training examples in one epoch, **then**
    then repeat until convergence and utill they have different iterations and epochs
    Repeat steps with all training samples till optimal threshold is obtained.
  end if
end while

Here,B_j is the bias associated with hidden units H_j,W_{ij} and V_i are the connections and visual layer products directly associated with hidden units. Also,A_i is the bias associated with visible layer V_i, and L is the learning rate. Once $H_L^{(2)}$ is sampled from the actual prior $p\left(H_L^{(2)}, H_L^{(3)}\right)$, then from $H_L^{(2)}$ we will directly generate $H_L^{(1)}$ using the stochastic processes Eq(1) and Eq(2). From $H_L^{(1)}$,we will generate directly, the input V_L,which gives us the sample or observation of input layer V_L from the DBN model. More specifically,the joint distribution of the input layer V_L and our three hidden layers is going to be a prior

$$P\left(V_L, H_L^{(1)}, H_L^{(3)}\right) = P\left(H_L^{(2)}, H_L^{(3)}\right) * P\left(H_L^{(1)} \mid H_L^{(2)}\right)$$

which is going to take the form

$$p\left(H_L^{(2)}, H_L^{(3)}\right) = \exp\left(H_L^{(2)^\top} \times W^{(3)} H_L^{(3)} + b^{(2)^\top} H_L^{(2)} + b^{(3)^\top} H_L^{(3)}\right) \mid \ Z. \quad (3)$$

We recognise here the probability distribution of Restricted Boltzmann machine,where we have our weight W. Between $H_L^{(2)}$ and $H_L^{(3)}$ and the biases $b^{(2)}$ and $b^{(3)}$ for the two layers ,and a normalization constant Z. So we have the prior $p\left(H_L^{(2)}, H_L^{(3)}\right)$ multiplied by $H_L^{(2)} \mid H_L^{(2)}$, that is part of the sigmoid belief network corresponds to the conditional $H_L^{(1)} \mid H_L^{(2)}$. So we know here for instance that $H_L^{(1)}$ is independent of $H_L^{(3)}$ given $H_L^{(2)}$ in our model and we multiply $P\left(V_L \mid H_L^{(1)}\right)$,so V_L is sampled from $H_L^{(1)}$ using the sigmoid belief network conditional distribution. Again,V_L is only dependent on $H_L^{(1)}$. And also,in the sigmoid belief network, the full distribution of $H_L^{(1)}$ given $H_L^{(2)}$ factorises into individual conditionals.

$$p\left(H_L^{(1)} \mid H_L^{(2)}\right) = \bar{\pi}_j p\left(H_{L_j}^{(1)} \mid H_L^{(2)}\right) \quad (4)$$

in otherwords,given $H_L^{(2)}$,each of the hidden units in the first hidden layer are independent and similarly for the visible layer x given $H_L^{(1)}$,thus

$$p\left(V_L \mid H_L^{(1)} = \pi_i p\left(v_i \mid H_L^{(1)}\right)\right. \quad (5)$$

Procedure of stacking RBMs for Pre-training the neural network:
We train a three layer DBN model Fig.2 in this study, by first going from a one hidden layer DBN (a), (which in essence is an RBM). We train that for a while and use its parameters to initialize a two hidden layer DBN (b). Then we would be left with the top part of the DBN (i.e. the RBM part of the two layer DBN) by keeping the weights fixed to initialize the parameters of the top layer.
To move to the three layer DBN(c), we again use the weights of the two layer (b) to initialize the lower part of (c) i.e. we use the weights of the top layer of (b) to initialize the upper part of (c) to get a good initialization for the top

layer of (c).

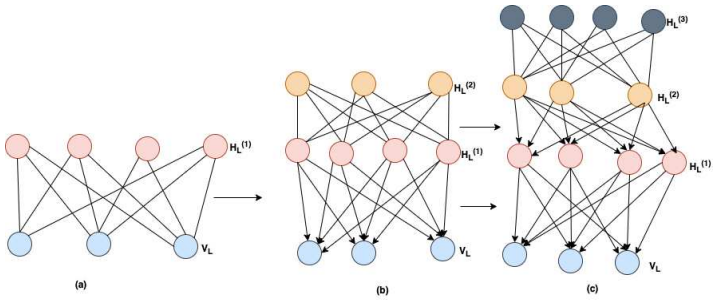


Fig. 2: A three layer DBN model. (a) is a on-layer DBN representing the starting point of the pre-training process (basically an RBM) (b) is a two-layer DBN initialized from (a). (c) shows a three-layer DBN forming a stack of Restricted Boltzmann machine. We used the weights of the top layer of (b) to initialize the upper part of (c) to get a good initialization to pre-train our entire network. This RBM the automatically extract meaningful features from the input vector (datasets)

Finally, we do a fine-tuning procedure by using the up-down method. DBN neural nets can be divided into two major parts.The first part are multiple layers of RBMs to pre-train our network. Secondly,we used a feed forward backpropagation network to further refine the results from the RBM stack.
RBMs use unsupervised learning method to learn static analysis of features in our data by reconstructing the input into permissions used between a set of malware and benign samples. Pre-training is done using the stack of RBMs such that the RBMs,model learns information about permissions of applications,which represent features of the visible layer,and maps each Android Application Package (APK) against a list of permissions in the hidden layer. We employ greedy layer-by-layer algorithm [30] and iterative Gibbs Sampling [31], detailed ealier, for pre-training. The sigmoid belief network is then trained through fine tuning, where we employed the back-propagation technique to obtain the best weights; since we need to have labels for the supervised learning problem.

3.1 Experiments

In this section, we run experiments to show how well our suggested model performed. The purpose of the experiments is to contrast the DBN model with other deep learning-based models that are already in use. In the end, we evaluate the results of our work and contrast them with previously published works that classified Android applications.

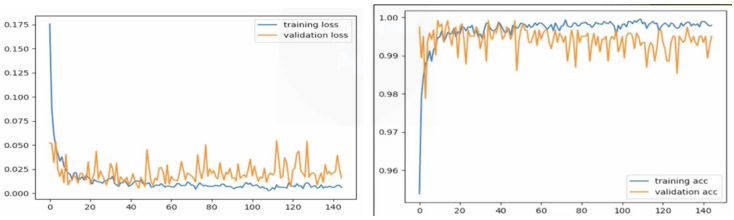
A. Experimental Setup: The simulation experiments were performed on a physical machine running macOS Monterey version 12.6 on a 1.4 GHz quad-core Intel Core i5 processor with 8 GB of 2133 MHz LPDDR3 memory. An NVIDIA GTX 1060 GPU with 6 GB memory sourced from Google colaboratory was used as an accelerator.
Our proposed DBN model and the baseline models trained for comparison were implemented on the TensorFlow version 2.3.4. and Keras version with python 3.
Internal hyperparameters were selected as follows: learning rate(L) was set at 0.01 at the initial stage and at every 10 iterations(epochs), a decay of 10% used.The batch size of 64 was chosen with adam as an optimizer for the trained models. We run the datset through 5000 epochs to arrive at the convergence when it was realised that the error in the visible layer became less than the treshold.

B. Dataset: The CICInvesAndMal2019 datasets, which is accessible at the University of New Brunswick (UNB), was utilized in this study. This dataset contains API calls, all created log files, permissions, and intents as static features, as well as samples with dynamic features,and were collected from real android devices. The datasets are used in prior studies, such as Taheri et al.[32]. Since our model uses supervised learning, we needed both benign and malicious applications to train it. As a result, we collected 5065 benign applications and 426 malicious applications, giving us a total of 5491 examples.

C. Metrics used in the Evaluation: The four widely used evaluation metrics in machine learning: Accuracy, Recall, Precision, and F1 Score of the top categorized apps, are used in this study. Accuracy essentially assesses the consistency of our classifications in comparison to the overall predictions observed. F1 Score is the weighted average of Precision and Recall, which provides us a good overall picture of how well our model performed. Recall indicates how effectively our model could identify the right apps, Precision is how well the model can distinguish between benign and harmful apps.

D. Neural network Configuration and model implementation: We built our DBN network in TensorFlow in two stages: first, we built a Python class for RBM to create and use the RBM, then we built our DBN. once the we create the RBM and load our datasets,we then create the DBN network. Three RBMs were employed in this study; the first had 4,000 hidden units, the second had 2000 hidden units, and the third had 50 hidden units. As a result, we generated a deep hierachy of representation of the training dataset. Each RBM was trained separately by calling the train function, which returns the current RBM output and uses it as the input for the next RBM. The learned representation of the input data is then transformed into a supervised prediction, which is a binary classifier. Finally,we use the output of the last hidden layer of the DBN network to classify applications.

E. Model Training and Validation: We trained our network using a dataset that includes both legitimate and malicious applications because our study used the supervised learning technique as stated ealier. 80% of the entire dataset used for training, and the remaining 20% for testing. Then, we flip it 10 fold to simulate an unseen application by using a portion of the entire dataset as test set. This would help us measure how effective our model would be performing. The current training method involves training with various epochs. Every epoch is a period in which a whole model is developed and tested. Testing is carried out using the training and validation set, where training loss and training accuracy were measured are tracked. The goal was to choose the model that results in the least amount of training loss. When we realized that the validation loss did not improve after every 100 epochs, we created a mechanism for stopping the training. Therefore, in our study, 500 epochs of training were completed, which means that epoch 500 was the point at which we found our optimum model.
As depicted in the figure 3,we have training loss decreasing with time at various epochs before flatulating.The figure illustrates that there is close relationship between training and validation loss as well as between training accuracy and validation accuracy. This is signigficant because if they are too wide apart, we begin to worry that our model might be overfitting.



(a) Training Loss (b) Validation Loss

Fig. 3: Training and Validation Losses

Table 1 shows different machine learning classifiers. Random Forest and decision Tree classifiers outperformed all the other algorithms considered in the study. With classification accuracy of 87.4% and 87.2% as well as 94.3% and 92.3% precision recorded for these two learners,with can see these are the closest to our DBN model. Logistic Regression classifier,(accuracy 63.2%) was the worst performer on our dataset. However,it produced the highest F1 score of 48.7% of all the machine learning models. This is due to the fact that our dataset is highly imbalanced classes of malicious apps 426 against benign sample 5065. When there is imbalanced data, F1 score has been found to be a more accurate measure of performance by Yerima et al[33]. We may therefore state with confidence that logistic regression also accurately described our data. Overall, the DBN model outperformed all seven well-known machine

learning classifiers on which we trained our data.

4 Results and Discussion

4.1 Comparison with baseline Machine Learning models trained on the CICInvesAndMal2019 dataset

.

Table 1: Comparison with baseline Machine Learning models trained on the CICInvesAndMal2019 dataset.

Machine Learning Classification Algorithm	Accuracy	Precision	Recall	F1 _ Scor
Logistic Regression Classifier	0.632	0.864	0.454	0.487
Linear Suppor Vector Machines	0.783	0.912	0.168	0.328
XGBoost Classifier	0.785	0.917	0.136	0.349
Gradient Boosting	0.762	0.950	0.153	0.416
Decision Tress Classifier	0.874	0.943	0.143	0.255
Random Forest Classifier	0.872	0.923	0.147	0.257
Bagging Classifier	0.867	0.963	0.191	0.247
KNeighbors Classifier	0.833	0.964	0.131	0.228
DBN Model	**0.987**	**0.989**	**0.974**	**0.979**

4.2 Comparison with baseline Deep Learning models trained on the CICInvesAndMal2019 dataset

In the same experimental setting, we trained four back propgation neural networks [34] on our data, and the results are shown in table 2. In general, the results were excellent, with accuracy, precision, recall, and F1 score all exceeding 98%. With these values, we have a model that is showing a lot of promise.

Table 2: Comparison with baseline Deep Learning models trained on the CICInvesAndMal2019 dataset.

Deep Learning Model	Accuracy	Precision.	Recall.	F1 _ Score
Deep Neural Network	0.872	0.728	0.874	0.795
Back-Propagation based Neural Network model	0.985	0.982	0.965	0.973
Multilayer Perception Neural Network	0.867	0.736	0.832	0.781
Rule based neural network	0.989	0.983	0.978	0.981
DBN Model	**0.987**	**0.989**	**0.974**	**0.979**

4.3 Comparison with current studies on android applications classification

.

Table 3 compares our model with other existing techniques that also trained models on cutting-edge Android datasets similar to our own, using deep learning techniques. The preliminary processing of the datasets in these studies is the same as in this paper. The different entropy features of APK data are extracted and those that can best represent Android applications are selected and classified using deep learning algorithms.

Table 3: Comparison with current studies on android applications classification

Study Reference	Malware App/Benign App	Model	Dataset
Kimet al.[35]	170/4659	1D CNN	Drebin
Cai et al.[36]	135/136	RF Algorithm	AndroZoo
Masum et al.[37]	1260/2539	Deep NN	MalGenome
Aktas et al.[38]	21/2479	kNN	UpDroid
Turkeret al.[39]	1635/1635	MLP	AMD
Our Study	**426 / 5065**	**DBN**	**CICInvesAndMal20**

5 Conclusion

Applications classification is an essential step to seperating malicious apps from legitimate ones to reduce the security threats on Android powered smart devices. In this study, we have built a DBN neural network using the TensorFlow framework, to model real Android application data (CICInvesAndMal2019 dataset) and categorize Android applications into benign and malicious classifications. The greedy layer-wise pre-training procedure was adopted as a way of initializing better parameters of the DBN. After that we used the same initialization to train other traditional neural networks with varying layer configurations as well as seven machine learning classifiers,for the purposes of fair comparison to assess the performance of our approach. Our model provides an intuitive and yet promising, approach to classifying Android applications.
The practicality of our approach is confirmed by comparative studies in tables 1,2,and 3. Additionally, it demonstrates that deep belief neural networks have good classification performance and offers promising research opportunities for addressing Android security problem.

References

[1] Caputo, D.: On the security and privacy challenges in android-based environments. PhD thesis, University of Genoa, Italy (2022)

[2] Alani, M.M., Awad, A.I.: Paired: An explainable lightweight android malware detection system. IEEE Access **10**, 73214–73228 (2022)

[3] El-Zawawy, M.A., Faruki, P., Conti, M.: Formal model for inter-component communication and its security in android. Computing, 1–27 (2022)

[4] Mazuera-Rozo, A., Escobar-Velásquez, C., Espitia-Acero, J., Vega-Guzmán, D., Trubiani, C., Linares-Vásquez, M., Bavota, G.: Taxonomy of security weaknesses in java and kotlin android apps. Journal of Systems and Software **187**, 111233 (2022)

[5] Garg, S., Baliyan, N.: Android security assessment: A review, taxonomy and research gap study. Computers & Security **100**, 102087 (2021)

[6] Garg, S., Baliyan, N.: Comparative analysis of android and ios from security viewpoint. Computer Science Review **40**, 100372 (2021)

[7] Meng, L., Zhang, Y.: Machine Learning and Intelligent Communications: Third International Conference, MLICOM 2018, Hangzhou, China, July 6-8, 2018, Proceedings vol. 251. Springer, ??? (2018)

[8] Alzaylaee, M.K., Yerima, S.Y., Sezer, S.: Dynalog: An automated dynamic

analysis framework for characterizing android applications. In: 2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security), pp. 1–8 (2016). IEEE

[9] Sun, Y., Fang, J., Chen, Y., Liu, Y., Chen, Z., Guo, S., Chen, X., Tan, Z.: Energy inefficiency diagnosis for android applications: a literature review. Frontiers of Computer Science **17**(1), 1–16 (2023)

[10] Polese, A., Hassan, S., Tian, Y.: Adoption of third-party libraries in mobile apps: A case study on open-source android applications (2022)

[11] Wijaya, N., Mulyani, S.H., Anggraini, Y.W.: Deepfruits: efficient citrus type classification using the cnn. Iran Journal of Computer Science, 1–7 (2022)

[12] Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. science **313**(5786), 504–507 (2006)

[13] Shao, H., Tang, L., Dong, L., Chen, L., Jiang, X., Wang, W.: A research of network applications classification based on deep learning. In: International Conference on Machine Learning and Intelligent Communications, pp. 13–21 (2018). Springer

[14] Senanayake, J., Kalutarage, H., Al-Kadri, M.O.: Android mobile malware detection using machine learning: A systematic review. Electronics **10**(13), 1606 (2021)

[15] Akbar, F., Hussain, M., Mumtaz, R., Riaz, Q., Wahab, A.W.A., Jung, K.-H.: Permissions-based detection of android malware using machine learning. Symmetry **14**(4), 718 (2022)

[16] Zhang, X., Mathur, A., Zhao, L., Rahmat, S., Niyaz, Q., Javaid, A., Yang, X.: An early detection of android malware using system calls based machine learning model. In: Proceedings of the 17th International Conference on Availability, Reliability and Security, pp. 1–9 (2022)

[17] Shatnawi, A.S., Yassen, Q., Yateem, A.: An android malware detection approach based on static feature analysis using machine learning algorithms. Procedia Computer Science **201**, 653–658 (2022)

[18] Syrris, V., Geneiatakis, D.: On machine learning effectiveness for malware detection in android os using static analysis data. Journal of Information Security and Applications **59**, 102794 (2021)

[19] Herron, N., Glisson, W.B., McDonald, J.T., Benton, R.K.: Machine learning-based android malware detection using manifest permissions. (2021). Proceedings of the 54th Hawaii International Conference on

System Sciences

[20] Islam, F.Z., Jamil, A., Momen, S.: Evaluation of machine learning methods for android malware detection using static features. In: 2021 IEEE International Conference on Artificial Intelligence in Engineering and Technology (IICAIET), pp. 1–6 (2021). IEEE

[21] Raymond, V.J., Raj, R., Retna, J.: Investigation of android malware with machine learning classifiers using enhanced pca algorithm. COMPUTER SYSTEMS SCIENCE AND ENGINEERING **44**(3), 2147–2163 (2023)

[22] Lakshmanarao, A., Shashi, M.: Android malware detection with deep learning using rnn from opcode sequences. International Journal of Interactive Mobile Technologies **16**(1) (2022)

[23] Fallah, S., Bidgoly, A.J.: Android malware detection using network traffic based on sequential deep learning models. Software: Practice and Experience **52**(9), 1987–2004 (2022)

[24] Bulso, N., Roudi, Y.: Restricted boltzmann machines as models of interacting variables. Neural Computation **33**(10), 2646–2681 (2021)

[25] Wani, T.M., Gunawan, T.S., Qadri, S.A.A., Kartiwi, M., Ambikairajah, E.: A comprehensive review of speech emotion recognition systems. IEEE Access **9**, 47795–47814 (2021)

[26] Tao, L., Mughees, A.: Deep Learning for Hyperspectral Image Analysis and Classification vol. 5. Springer, ??? (2021)

[27] Zhang, Y., Henao, R., Li, C., Carin, L.: Bayesian dictionary learning with gaussian processes and sigmoid belief networks. In: IJCAI, pp. 2364–2370 (2016)

[28] Yamaguchi, K., Templin, J.: A gibbs sampling algorithm with monotonicity constraints for diagnostic classification models. Journal of Classification **39**(1), 24–54 (2022)

[29] Neo, Y., Teo, T.T., Woo, W.L., Logenthiran, T., Sharma, A.: Forecasting of photovoltaic power using deep belief network. In: Tencon 2017-2017 IEEE Region 10 Conference, pp. 1189–1194 (2017). IEEE

[30] Gao, Z., Birge, J.R., Chen, R.L.-Y., Cheung, M.: Greedy algorithms for the freight consolidation problem. In: 22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2022) (2022). Schloss Dagstuhl-Leibniz-Zentrum für Informatik

[31] Kang, Z., You, C., Zhang, R.: 3d placement for multi-uav relaying: An iterative gibbs-sampling and block coordinate descent optimization approach. IEEE Transactions on Communications **69**(3), 2047–2062 (2020)

[32] Taheri, L., Kadir, A.F.A., Lashkari, A.H.: Extensible android malware detection and family classification using network-flows and api-calls. In: 2019 International Carnahan Conference on Security Technology (ICCST), pp. 1–8 (2019). IEEE

[33] Yerima, S.Y., Sezer, S.: Droidfusion: A novel multilevel classifier fusion approach for android malware detection. IEEE transactions on cybernetics **49**(2), 453–466 (2018)

[34] Karmakar, S., Goswami, S.: A new procedure for optimization of hidden layer neurons during learning through gradient descent process of neural network and improvement of performance in the chaos forecasting. Iran Journal of Computer Science **4**(4), 293–303 (2021)

[35] Kim, H.-I., Kang, M., Cho, S.-J., Choi, S.-I.: Efficient deep learning network with multi-streams for android malware family classification. IEEE Access **10**, 5518–5532 (2021)

[36] Cai, H., Meng, N., Ryder, B., Yao, D.: Droidcat: Effective android malware detection and categorization via app-level profiling. IEEE Transactions on Information Forensics and Security **14**(6), 1455–1470 (2018)

[37] Masum, M., Shahriar, H.: Droid-nnet: Deep learning neural network for android malware detection. In: 2019 IEEE International Conference on Big Data (Big Data), pp. 5789–5793 (2019). IEEE

[38] Aktas, K., Sen, S.: Updroid: Updated android malware and its familial classification. In: Nordic Conference on Secure IT Systems, pp. 352–368 (2018). Springer

[39] Türker, S., Can, A.B.: Andmfc: Android malware family classification framework. In: 2019 IEEE 30th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC Workshops), pp. 1–6 (2019). IEEE