Android Malware Detection Approach Using Stacked AutoEncoder and Convolutional Neural Networks

Brahami Menaouer, LABAB Laboratory, National Polytechnic School of Oran - M. Audin, Algeria*
https://orcid.org/0000-0003-0045-9797
Abdallah El Hadj Mohamed Islem, National Polytechnic School of Oran, Algeria
Matta Nada, TechCICO Laboratory, University of Technology of Troyes, Troyes, France

ABSTRACT
In the past decade, Android has become a standard smartphone operating system. The mobile devices running on the Android operating system are particularly interesting to malware developers, as the users often keep personal information on their mobile devices. This paper proposes a deep learning model for mobile malware detection and classification. It is based on SAE for reducing the data dimensionality. Then, a CNN is utilized to detect and classify malware apps in Android devices through binary visualization. Tests were carried out with an original Android application (Drebin-215) dataset consisting of 15,036 applications. The conducted experiments prove that the classification performance achieves high accuracy of about 98.50%. Other performance measures used in the study are precision, recall, and F1-score. Finally, the accuracy and results of these techniques are analyzed by comparing the effectiveness with previous works.

KEYWORDS
Android Security, Classification, CNN, Dimensionality Reduction, Features Extraction, Knowledge Management, Malware Detection, Mobile Malware, Smartphone Security, Stacked AutoEncoder

1. INTRODUCTION
In the world that we are living today, mobile device technologies have grown rapidly due to the daily increase in the number of users and facilities. According to (Senanayake et al. 2021), smartphone usage and its associated applications are rapidly increasing due to the growing improvement in the hardware and software on smart devices. According to statistics, it is predicted that there will be 4.3 billion smartphone users by 2023. Likewise, mobile devices are an integral part of our everyday life (Brahami et al. 2022). They are no longer used merely for facilitating wireless communication and are also widely used in education, social media, shopping, industry, and banking (Mohamad Arif et al. 2021). Their widespread use cause large quantities of data containing highly-sensitive information to be provided. Android mobile is very challenging because it is an open-source operating system
that is also vulnerable to attacks. Similarly, android has recorded a higher percentage of malware among smartphone devices. Android smartphones accounted for the highest percentage, with 50%. Currently, cybercriminals have proven significantly efficient in uncovering new vulnerabilities in popular mobile operating systems and installed applications (Kouliaris et al. 2020). According to (Penning et al. 2014), the cybercriminal motivations behind mobile malware may vary from collecting sensitive data to financial gain and accessing private networks. Likewise, malware writers continue to come up with new ways of hiding their attacks, making it increasingly difficult to identify and neutralize (Mercaldo et al. 2016). Currently, attackers who make malware applications have come up with new methods of targeting victims of Android users. To overcome these issues and to keep mobiles safe from a number of malicious applications, new mobile malware detection techniques are also evolving to counter these threats.

Mobile malware detection has traditionally been based on manually examining the behavior and/or de-compiled code of known malware programs in order to design malware signatures by hand (Bagui & Benson, 2021). According to (Aldriwish, 2021; Pektaş & Acarman, 2020), the mobile malware detection service provides organizations with critical information on malware infections and malware-based attacks that could impact their end users. At the same time, various mobile malware detection systems are designed to address malware issues like firewalls, antivirus software, and intrusion detection systems. For (Alazab et al. 2020), there are three methods of extracting features from mobile apps that are frequently used by security vendors and researchers: static, dynamic, and hybrid (a combination of static and dynamic). These techniques used by malware researchers encourage the need for new research on detection techniques, in which deep learning based methods are included and used to be more useful with high accuracy.

In a general sense, deep learning techniques automate the analysis of mobile malware detection by recognizing the malware pattern (Gohari et al. 2021; Li et al. 2018). These techniques are a specific field of artificial intelligence that predicts future decisions and outputs based on datasets (Zhang et al. 2021). They refer to the process of characterizing malware behavior and applying classifiers to evaluate the dataset. More recently, classifiers of deep learning have been involved in the development of intelligent systems. Deep learning methods develop a neural network that simulates the brain of humans for analytical learning. According to (Alzaylaee et al. 2019; Hou et al. 2017; Nix & Zhang, 2017), deep learning classifiers have inspired a great number of effective approaches in image segmentation, protein structure, machine translation, speech recognition, and natural language processing. The deep learning techniques acquire a labeled dataset and generate an output model which is able to process new data. Classifiers learn the input and labeled output from an abundance to construct a model.

To solve the previous problems, the proposal of automated classifiers to assist Android application detection can greatly reduce time and cost, which is of critical importance to increase detection accuracy and to improve mobile security. Overall, there are three major contributions made by this research: (1) A comprehensive review of ideas and research efforts in the background is conducted and organized through a three-step process catered for Deep Learning (SAE & CNNs) based Android malware detection and classification; (2) Using the Android application “Drebin-215” dataset containing vectors of 215 characteristics from 15,036 apps samples, we compared our performance model to state-of-the-art and recently presented models in the literature. As well, we described a scenario for analyzing the Drebin-215 dataset: malware binary that distinguishes benign from malicious applications; (3) an overview and summary of current state-of-the-art solutions are elaborated with in-depth analysis and future work in order to assist other researchers in broadening their perspective on DL-based Android malware.

The remainder of this paper is organized as follows. Section 2 introduces the literature review on deep learning methods and mobile malware detection and classification techniques proposed by various researchers. Section 3 presents the proposed methodology based on Stacked AutoEncoder (SAE) and Convolutional Neural Networks (CNNs) used for the detection and classification of Android
malware. The experimental results and discussion are covered in Sections 4 and 5. Finally, Section 6 concludes and summarizes the key findings and future works.

2. LITERATURE REVIEW

As mentioned earlier, we can be usually divided the methods of malware detection into two categories: static analysis and dynamic analysis. In this field, detecting Android malware with static analysis is examined comprehensively by disassembling the malware binary files without. The advantages of static analysis are low cost and require a less time of analysis. In contrast, the dynamic analysis can analyze the behavior of the malware during executing it in a debugger. For this reason, several works address the mobile malware detection problem by classifying features got from real world application and using well-known deep learning techniques.

2.1. Android Malware Detection With Static Analysis

In most literature, many researchers have been developed several solutions using static analysis by using features such as permissions, API calls, Dalvik byte code, and intents. In the right context, many researchers have verified the effectiveness of the tools or suggested new tools that may be more effective in the process of accurately detecting malicious applications. Furthermore, conventional static detection methods are limited by signature databases, complex reflective calls, and high reliance upon users and resources. For example, Hou et al. (2016) proposed a deep belief network-based Android malware detection system DroidDelver using a real sample collection from Comodo Cloud Security Center. They perform feature engineering on API calls by categorizing them into blocks. (Fereidooni et al. 2016) proposed a system to detect Android malware, named ANASTASIA, using features such as intents, permissions, system commands, and API calls. The system uses many classifiers including the deep neural network. In other work, (Mchaughlin et al. 2017) proposed a novel Android malware detection system that uses a deep convolutional neural network (CNN). In the system, malware classification is performed based on static analysis of the raw opcode sequence from a disassembled program. In the work done by (Zegzhda et al. 2018), it explores the use of deep learning for malware identification in the Android operating system. For this, a self-designed approach is proposed for representing an Android application for a convolutional neural network, which consists in constructing an RGB image, the pixels of which are formed from a sequence of pairs of API calls and protection levels. In the study done by (Vinayakumar et al. 2018), to use the Recurrent Neural Network (RNN) technique of deep learning approach particularly Long Short-Term Memory (LSTM) for analysis and classification of malicious apps in Android devices of time-varying sequences of benign and malware apps. (Wang et al. 2019) proposed a hybrid model for Android malware detection with DAE and CNN to improve detection accuracy and reduce the training time. The CNN-S and CNN-P structures are employed in the training process, during which the “dropout” technique is used to prevent overfitting. Likewise, (Feng et al. 2020) proposed an effective Android malware detection system named MobiTive, leveraging customized deep neural networks to provide a real-time and responsive detection environment on mobile devices. MobiTive is a pre-installed solution rather than an app scanning and monitoring engine using after installation, which is more practical and secure. Another system by (Pektaş & Acarman, 2020), it is to propose a new mechanism for detecting malicious Android applications by using pseudo-dynamic analysis and constructing an API call graph for each execution path, based on a deep learning model built and trained on a data set consisting of approximately 30 thousand malicious apps and 25 thousand benign apps. In the study done by (Pektaş & Acarman, 2020), to use the API call graph as a graph representation of all possible execution paths that malware can track during its runtime. The embedding of API call graphs transformed into a low dimension numeric vector feature set is introduced to the deep neural network. Yet, (Feng et al. 2020) proposed an effective Android malware detection system named MobiTive leveraging customized deep neural networks to provide a real-time and responsive detection
environment on mobile devices. Moreover, deep learning-based approach can be maintained on server side efficiently for malware detection. Moreover, (Iadarola et al. 2021) proposed a method relying on application representation in terms on images used to input an explainable deep learning model designed by authors for Android malware detection and family identification. Similarly, (Amato et al. 2021) proposed a method based on deep learning (Neural Networks and MultiLayer Perceptrons) aiming at discovering attacks towards the CAN-bus by analyzing a real-world dataset with the injection of messages from different types of attacks: denial of service, fuzzy pattern attacks, and attacks against specific components. (Hoang Khoa et al. 2021) proposed a system named uitMultiClandroid to detect Android malware by using deep learning. For this, the authors applied image classification models to detect Android malware based on the converted images from APK files. Furthermore, (Gao et al. 2021) proposed a novel approach for Android malware detection and familial classification based on the Graph Convolutional Network (GCN). The general idea is to map apps and Android APIs into a large heterogeneous graph, converting the original problem into a node classification task. The heterogeneous graph is then fed into the GCN model, iteratively generating node embeddings that incorporate topological structure and node features. (Xing et al. 2022) proposed a novel malware detection model which combines a grey-scale image representation of malware with an autoencoder network in a deep learning model, analyses the feasibility of the grey-scale image approach of malware based on the reconstruction error of the autoencoder, and uses the dimensionality reduction features of the autoencoder to achieve the classification of malware from benign software. In the right context, (Dhabal & Gupta, 2023) proposed a novel Android malware detection framework using a hybrid of bidirectional long short-term memory (BiLSTM) and merged sparse auto-encoder (MSAE) with softmax deep learning mode. In the study done by (Alomari et al. 2023), introduced deep learning and feature selection methodologies to arrive at a high-performance malware detection system. Two different malware datasets are used to detect malware and differentiate it from benign activities. In conclusion, static analysis approaches can efficiently identify malware but they may become invalid when malicious applications use reflection or dynamic loading to perform malicious behaviors.

2.2. Android Malware Detection With Dynamic Analysis

In most studies, dynamic analysis involves running an application in a sandbox environment or on a real device. In dynamic analysis, the detection phases and training happen during the execution of the applications and analyze a series of data information (log, network traffic). According to (Arshad et al. 2016), the current antimalware strategies are divided into two categories, Static and Dynamic Approaches. A study was conducted by (Bulut & Yavuz, 2017) in order to present a novel model based on deep learning for the prediction of mobile malware without requiring execution in a sandbox environment. After optimizing their weights with automatic encoder and they were classified with a multilayer perceptron with an accuracy of 93.67%. Additionally, (Martín et al. 2017) proposed a new genetic algorithm designed to evolve the parameters, and the architecture, of a DNN with the goal of maximizing the malware classification accuracy, and minimizing the complexity of the model. The model described is tested on a specific problem within the security field, which consists of determining the malware family of different malicious samples. A study was conducted by (Kim et al. 2018) in order to present a malware detection framework based on multiple neural networks. Every network has a single feature input and output score. The final detection result is a combination of all the models. (Xu et al. 2018) proposed an effective approach named CDGDroid for Android malware detection based on deep learning. For this, the authors used the semantics graph representations, that is, control flow graph, data flow graph, and their possible combinations, as the features to characterize Android application in order to train the classification model via Convolutional Neural Networks (CNNs). Additionally, (Jin et al. 2020) proposed a malware detection method based on deep learning, which uses malware images and a set of autoencoders to detect malware. The method is to design an autoencoder to learn the functional characteristics of malware, and then to observe the reconstruction error of the autoencoder to realize the classification and detection of malware and benign software.
A study in dynamic analysis domain was conducted by (Ali-Gombe et al. 2019) in order to develop the DroidScraper system to recover important runtime data structures of application software by enumerating and reconstructing the objects in memory for mobile device forensics and postmortem analysis. In an earlier work, Lu et al. (2020) proposed an Android malware detection algorithm based on a hybrid deep learning model which combines deep belief network (DBN) and gate recurrent unit (GRU). First of all, analyze the Android malware; in addition to extracting static features, dynamic behavioral features with strong antiobfuscation ability are also extracted. Then, build a hybrid deep learning model for Android malware detection. In the study done by (Mbaziira, 2020), to use a deep neural network (DNN) implementation of deep learning called DeepLearning4J (DL4J) to generate our models for mobile malware detection. The models detect mobile malware with accuracy rates ranging between 97% and 99% when applied to two types of malicious datasets. In other words, (Arslan et al. 2021) proposed a novel method encompassing feature selection and feature weighting for malware detection in mobile applications using deep learning (multi-layer structure). To this end, firstly, the manifest file was read from the Android application package. Different features such as activities, services, permissions were extracted from the file, and for classification, a selection was made among these features. Besides that, (Sihag et al. 2021) proposed De-LADY (Deep Learning based Android malware detection using Dynamic features) an obfuscation resilient approach. It utilizes behavioral characteristics from dynamic analysis of an application executed in emulated environment. In the study of (Alzubaidi, 2021) which has implemented a detection scheme using effective deep learning algorithms (LSTM and MLP). Also, the author has tested their ability to detect malware by employing private and public datasets, with an accuracy of over 99%. Besides that, (Ahmad et al. 2021) have implemented a detection system using efficient Deep Learning algorithms (i.e., LSTM and BLSTM) by employing the latest publicly available “CICAndMal2017” dataset in order to protect the android systems against numerous attacks. Further, it uses standard evaluation metrics for the measurement of the system’s performance. (Lukas & Kolaczek, 2021) described the basic concepts of the Android and deep learning algorithms in order to test several features of the application and checking several deep learning algorithms. In addition, the authors have proposed a solution based on the use of binary file representation and self-organizing maps. Another research by (Sumit & Adamuthe, 2022), it is to propose a deep learning model capable of building a feature set from the EMBER dataset and classifying the malicious codes’ static PE files efficiently. In earlier work, Atacak et al. (2022) proposed a hybrid detection system that combines the feature extraction and dimension reduction power of the convolution layers in the CNN architecture, and the decision-making capability of fuzzy logic is proposed. The proposed system reduces the number of inputs for classification by applying feature extraction with only two convolution layers, two pooling layers, and five connected layer neurons to all permission information obtained through static analysis. In the study done by (Akhtar & Feng, 2022), used the stacking CNN-LSTM techniques to overcome major malware detection deficiencies, including the inefficiency of human feature building and the limitations of existing learning algorithms. The proposed CNN-LSTM method is used for the detection of advanced malware without any feature engineering.

Therefore, to overcome the issues associated with the existing models in detecting previously unseen malware. Likewise, all the techniques described above use a training process in which input samples are labeled as benign or malicious. In this paper, we present a deep learning model applying Stacked AutoEncoder (SAE) for dimensionality reduction and Convolutional Neural Network (CNN) for detection and classification of malware apps in Android devices of time-varying sequences of benign and malware apps.

3. APPROACH PROPOSED

As per our objective and motivations, this study is associated with some background ideas and research efforts as shown in Figure. 1. Briefly, especially using Autoencoder Neural Network and
CNNs models for dimensionality reduction and classification (mobile malware). In general, widely followed automatic dimensionality reduction and binary classification approaches performed with Deep Learning models such as SAE and CNNs have been directed to the malware apps in Android (See Figure 1).

In the context mentioned above, this study followed an easy-to-design Dimensionality Reduction using SAE and deep CNN approach for automatic detection and classify Android malware, by considering the Android application as an input dataset (Drebin-215). In this respect, Figure. 2 represents the stages within the flow of the introduced deep learning approach. After the image pre-processing based enhancement, the classification was made by using SAE and CNN models. In the next stages, we evaluate our introduced deep learning approach by using an original dataset (Drebin-215) consisting of 15,036 application samples. The whole flow is a deep learning approach applied to target Android malware data, which is essential for detection from applications samples inputs in the form of visual elements (see Figure. 2).

3.1. Dataset Description

This dataset has been used for analyzing the performance of deep learning algorithms used for the automated detection and classification of mobile malware. For this, some researchers provided Android malware datasets with features in order to facilitate the analysis and utilization of features. To evaluate our approach performance and compare it with existing popular approaches, we compiled an Android application dataset (Drebin-215) comprising of vectors of 215 features from 15,036 app samples. Formally, the Drebin samples are publicly available and widely used in the research community. For (Arp et al. 2014; Yerima & Sezer, 2018), 9,476 were benign samples while the remaining 5,560 were malware samples from the Drebin project. The supporting file contains a further description of the feature vectors/attributes obtained via static code analysis of the Android apps. Drebin dataset comes
from different malware families which they crawled in the period of August 2010 to October 2012 (Wu et al. 2021; Zhang et al. 2018), providing eight types of features namely:

- transact, API call signature
- nonservice-connected, API call signature
- bindService, API call signature
- READ_PHONE_STATE, Manifest Permission
- GET_ACCOUNTS, Manifest Permission
- BROADCAST_SMS, Manifest Permission
- android.intent.action.PACKAGE_REPLACED, Intent
- android.intent.action.BOOT_COMPLETED, Intent
- android.intent.action.BATTERY_LOW, Intent
- etc.

Furthermore, the dataset has been divided into two parts: training and testing. The training data consist of 80% of the dataset. The checking and testing data, on the other hand, consist of 20% of the dataset. Finally, the dataset was stored in a Microsoft Excel spreadsheet and then save as Comma Separated Values (CSV).

4. AUTOENCODER NEURAL NETWORK

In the past few years, feature learning with Artificial Neural Network (ANN) architectures has attracted increasing attention, which can be used to extract optimal features from high-dimension data. Likewise, the AutoEncoder (AE) is a kind of ANN used in semi-supervised and unsupervised learning which is proposed by Hinton & Salacukhudinov in 2006. The autoencoder is an unsupervised deep learning (DL) algorithm for dimensionality reduction and heterogeneous data integration based on feed-forward neuronal networks. As well, autoencoder is a type of neural network that can be used to learn a compressed representation of raw data (Wang et al. 2017). More specifically, the autoencoder develops a better feature representation for the high dimensional input data. It finds the correlation between the input data (Aslam et al. 2021). According to (Mahmud et al. 2020), AE is a neural network comprising an encoder denoted by $E$, followed by a decoder denoted by $D$, and it aims to reconstruct the input at the output under certain constraints. The autoencoder is a three-layer feed-forward neural network that comprises an input layer, hidden layer, and output layer.
As illustrated in Figure 3 (see Figure 3), the input layer and the hidden layer construct an encoder. The hidden layer and the output layer construct a decoder.

The encoder encodes the high-dimensional input data \( x = \{ x_1, x_2, \ldots, x_n \} \) into a low dimensional hidden representation \( h = \{ h_1, h_2, \ldots, h_m \} \) by a function \( f \):

\[
h = f(x) = S_f(W_x + b) \tag{1}
\]

where \( S_f \) is an activation function. The encoder is parameterized by a \( m \times n \) weight matrix \( W \) and a bias vector \( b \in \mathbb{R}^m \).

The decoder in Figure 1 maps hidden representation \( h \) back to a reconstruction \( x' = \{ x'_1, x'_2, \ldots, x'_n \} \) by a function \( g \):

\[
x' = g(h) = S_g(W'h + b') \tag{2}
\]

where \( S_g \) represents the decoder’s activation function. The decoder’s parameters are comprised of a bias vector \( b' \in \mathbb{R}^n \) and a \( n \times m \) weight matrix \( W' \). The function \( S_f \) and \( S_g \) are usually non-linear activation function, e.g., the hyperbolic tangent function and the sigmoid function (Zhang et al. 2020). Furthermore, the AE is trained to minimize the reconstruction error between \( x \) and \( x' \) using two ways of formulating the reconstruction error: Square Error and Cross-entropy.

The first part of the network is what we refer to as the Encoder. It receives the input and encodes it in a latent space of a lower dimension. The second part (the Decoder) takes that vector and decode it in order to produce the original input. The cross-section between encoder and decoder named code the layer is the core of Autoencoder that can reflect the essential characteristics of high dimensional data set with nested structure, and set intrinsic dimensions of high dimensional data sets. The key

*Figure 3. Schematic representation of an Autoencoder with three fully connected hidden layers*
characteristic of autoencoders’ architecture is a bottleneck which limits the amount of information that can flow through the network, forcing a learned compression of the input data.

In the early development of deep learning, autoencoder has been viewed as a solution to solve the problem of unsupervised learning. According to (D’Angelo et al. 2019), the main difference with a classical neural network is that for an Autoencoder, the desired output coincides with the given input. In the area of data dimension reduction, AE has been applied to dimensionality reduction (Martin et al. 2019; Wang et al. 2016), abnormal value detection, Image denoising, Image compression, and Image generation (Zhao et al. 2021). Briefly, dimensionality reduction is the process of reducing the number of dimensions in the data either by excluding less useful features (Feature Selection) or transforming the data into lower dimensions (Feature Extraction). As well, dimensional reduction is the problem of learning a transformation from a higher-dimensional input space, to a latent space while preserving as much as possible of the variation of the input space (Yuan et al. 2021; Seyfioğlu et al. 2017). Autoencoder reduces the dimensionality of linear and nonlinear data hence it is more powerful than PCA (Principal Component Analysis) because PCA can only learn the linear transformation of the features. Besides that, Autoencoders are deep neural networks used to reproduce the input at the output layer i.e. the number of neurons in the output layer is exactly the same as the number of neurons in the input layer (Pang et al. 2021). In this sense, the deep autoencoder utilizes the excellent feature reconstruction capability of the autoencoder to learn features. In recent years, many variants of deep Autoencoder have been proposed, such as Stacked autoencoders (or Deep Autoencoders [DAE]), Sparse Autoencoder (SAE), Under Complete Autoencoder (UCAE), Variational Autoencoder (VAE), and LSTM Autoencoder.

In addition to this, Sparse AutoEncoders (SAE) is an extension of the autoencoder framework as they contain several layers between the input and the output (Chen et al. 2021). It’s useful when using autoencoders as inputs to downstream supervised models as it helps to highlight the unique signals across the features. Sparse autoencoders are similar to the under-complete autoencoders in that they use the same image as input and ground truth. As well, sparse autoencoders are used to pull out the most influential feature representations. Moreover, Stacked AutoEncoder (SAE) is a neural network consisting of a few layers of sparse autoencoders, with the yield of each hidden layer compared with the output of the successive hidden layer. As well, the stacked autoencoder (SAE) is used to learn and deliver strong and high-level features. According to (Sahaai et al. 2021; Guo et al. 2021), the SEA consists of three stages:

- To train the autoencoder with input data and to obtain scholarly data.
- The information gathered from the previous layer is used as a contribution for the next layer, and this process is repeated until the planning is completed.
- After all of the hidden layers have been prepared, use the backpropagation calculation to restrict the cost potential, and loads are refreshed with the preparation set to achieve tweaking.

For instance, Zabalza et al. (2016) have proposed to use of a stacked autoencoder for dimensionality reduction and feature extraction in hyperspectral imaging. Besides that, Sahay et al. (2019) suggests the use of a cascaded autoencoder that can perform both tasks of denoising and dimensionality reduction. Thus, autoencoders prove to be a useful tool for dimensionality reduction as this method has added benefits over traditional methods such as PCA. In the work done by Li et al. (2015), is to presents an ML-NIDS that uses an autoencoder for dimension reduction and a DBN for classification. The DBN consists of multiple layers of RBMs and an additional layer of BP neural network as a classifier that distinguishes malicious samples from normal samples. Another study by Shone et al. (2018), is to produce promising results by implementing a deep learning algorithm to convert high-dimensional data to low-dimensional data by utilizing a deep auto-encoder.
4.1. Dimensionality Reduction Using Stacked AutoEncoders (Phase I)

As per our objective, we use the SAE as an unsupervised dimensionality reduction model. In the literature, an unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data. This could convert complicated high dimensional data into low dimensional codes with nonlinear mapping thereby reducing the dimensionality of data, extracting the main features of the data then using the CNN learning method to detect and classify mobile malware code. In this respect, we carried out several tests in order to find the good optimal value of the hidden layers number of neural networks. Indeed, the retained value was 2 hidden layers between the input layer ($X$), a vector of size 215, and the code layer ($Z$) whose size is 100. At the same time and after several evaluation tests, the learning of our stacked autoencoder model was set at 100 epochs and a batch of 10. we summarize the characteristics of our SAE model as shown in Table 1 (See Table 1).

4.1.1 The Autoencoder Evaluation Parameters

In addition, we conducted to study of the evaluation parameters of our stacked autoencoder model. Therefore, the best evaluation parameter of the model is to minimize the loss function which is equal to the Mean Square Error (MSE) function using the Adaptive Delta (Adadelta) Optimizer algorithm. The optimization objective is to minimize the Mean Square Error (MSE) loss function between the Input Layer ($X$ input vector) and the Output Layer ($X$ output vector) of our stacked autoencoder model (see Figure 1). It was concluded that the Adadelta optimizer algorithm and Mean Square Error (MSE) loss function consistently result in the best evaluation parameter.

In the context mentioned above, Table 2 provides the summary test of some stacked autoencoder architectures where we played on parameters such as the hidden layers number, loss function, and Adadelta optimizer.

<table>
<thead>
<tr>
<th>Table 1. The characteristics of stacked autoencoder model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>input_1</td>
</tr>
<tr>
<td>dense_1</td>
</tr>
<tr>
<td>dense_2</td>
</tr>
<tr>
<td>dense_3</td>
</tr>
<tr>
<td>dense_4</td>
</tr>
<tr>
<td>dense_5</td>
</tr>
<tr>
<td>dense_6</td>
</tr>
</tbody>
</table>

Total Params: 156.755
Trainable Params: 156.755
Non-trainable Params: 0

<table>
<thead>
<tr>
<th>Table 2. The tested architectures results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>
4.1.2 Results and Discussions of Dimensionality Reduction

In this study, the most widely used Drebin-215 dataset has been chosen to verify the proposed dimensionality reduction using Python programming language with the Keras framework. As a result, we noticed that our proposed Stacked autoencoder architecture achieved its goal with a minimal loss rate compared to the other tested architectures cited in the previous table (See Table 2). The graphical representation shown in Figure 4 gives the dimensionality reduction results graphs for 100 epochs with training loss vs testing loss.

5. CONVOLUTIONAL NEURAL NETWORK (CNN)

Deep Learning (DL) is part of an artificial neural network technique and a subclass of machine learning (Clottey et al. 2021). Moreover, DL is part of a broader family of machine learning methods based on learning data representations (Brahami et al. 2022; Brahami et al. 2022; Qiu et al. 2021). Deep learning is an advanced sub-field of machine learning, which advanced Machine Learning closer to Artificial Intelligence. In DL, multiple layers are used for a higher level of the feature from the input dataset (Sundareswaran & Lavanya, 2020). Furthermore, DCNN is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery (El-Alami et al. 2020). According to earlier works, deep learning algorithms can be classified into three categories based on the form of the learning process, namely supervised, semi-supervised, and unsupervised (Liu et al. 2021). The architecture of CNN models, in any CNN model there are three types of main layers convolutional layer, pooling layer and dense layer (Brahami et al. 2022). Generally, a CNN mainly consists of three key parts: convolution layers, pooling layers, and fully connected layers (Maduranga & Nandasena, 2022).

Most of the layers in CNN convert an input image to features. Only the last few layers are used for classification (Gadekallu et al. 2020). Finally, Figure 5 graphically presents the schematic diagram of a basic convolutional neural network (CNN) architecture, with its main elements (see Figure 5).

CNN utilizes the convolution operation to replace the affine transformation as in the case of DNN. According to (Pandey et al. 2019), CNN usually consists of a convolution layer that utilizes a set of convolutional filters to extract multiple local patterns at each local region in the input space and produce many feature maps. The research by (Vinayakumar et al. 2018) shows that CNN and its variant architecture are better than classic machine learning classifiers in Android malware detection. Experimental results show that one-dimensional convolution in CNN has high accuracy in Android malware detection. For (Liu et al. 2017), CNN–based approaches show their powerful feature extraction
ability and a high classification rate and accuracy because it learns through weight sharing, local perceptive field, sub-sampling, and other related processes.

In general, we remember that deep learning techniques have the characteristics of exploiting non-linear processing among logical units that are connected to each other in a cascade over different layers, where outputs of one layer serve as input of the next one.

5.1 System Requirements

To validate the proposed method, computer hardware was configured including Intel(R) Core(TM) i7-7500U, CPU (2.70 GHz to 2.90 GHz), GTX1050Ti GPU, and 8 GB RAM. The primary software configuration included Microsoft Windows 10 64-bit Operating System, Python compiler, Spyder 4.0.1 editor, deep learning framework PyTorch, and uses the neural network library Keras 2.2.4, Numpy, Pandas, SciPy, scikit-learn, and Matplotlib 3.1.3.

5.2 Evaluation Criteria

The performance of an Android malware detection and classification approach is evaluated by various performance metrics. The accuracy metric, which determines the correctness of the identified instances in both classes of binary classification (Malware (M) and Benign (B)), must be supplemented by other metrics such as precision, recall, $F_1$ score, and AUC. These popular parameters are defined as follows:

$$Sensitivity = \frac{TP}{TP + FN} \quad (3)$$

Sensitivity (true positive fraction) is the probability that a detection test is positive, given that the application has the malware app.

The recall (Specificity) (true negative fraction) is the probability that a detection test is negative, given that the application does not have the malware.

$$Recall = \frac{TN}{TN + FP} \quad (4)$$

In general, sensitivity and specificity evaluates the effectiveness of the algorithm on a single class, positive and negative respectively.

$$Accuracy = \frac{TP + TN}{(TP + TN + FP + FN)} \quad (5)$$
Precision = \( TP / (TP + FP) \) \hspace{1cm} (6)

The precision metric will show the ratio of true positives over the total number of detected entities. In other words, this metric will help us understand how well a model is in returning only the true positives and not unrelated entities.

\[
F - score = 2 \frac{Precision \times Recall}{Precision + Recall}
\] \hspace{1cm} (7)

Commonly, accuracy is the most used metric to evaluate classification performance. This metric calculates the percentage of samples that are correctly classified. As well, precision is how “precise” the model is out of those predicted positives and how many of them are actually positive. Thus, F1 Score (also known as F-measure) might be a better measure when a balance between Precision and Recall is needed with an uneven class distribution (a large number of Actual Negatives). This metric can be used to show the overall performance of a tool.

Likewise, a confusion matrix is commonly used to visualize the performance of a classification algorithm. Measurement of TP, FP, TN, and FN uses a confusion matrix of classification with two classes. where TP, TN, FP, and FN stand for true positives, true negatives, false positives, and false negatives, respectively.

5.3. Results and Discussions (Detection and Classification – Phase II)

In this section, experiments on a popular Android applications dataset are conducted. we first apply the proposed approach to the binary classification task for malware. To distinguish the malware samples and the normal, benign samples, we need a sufficient quantity of both malware samples and benign samples. Otherwise, the most widely used Android apps (Drebin-215) dataset has been chosen to verify the proposed approach using Python programming language with Keras framework. For this, we investigated the automatic binary classification of the cases of Android malware using deeper networks (SAE and Baseline CNN). Multiple layered models have been designed for performing convolution. The Rectified Linear Unit (ReLU) activation function is used to define the output of internal layers. Moreover, the learning of our CNN network was fixed at 100 epochs, a batch of 50 instances, 5 for the kernel size of the 1D convolution window, 64 for the number of convolution output filters, and the value of 2 for the max-pooling size of the aggregation layer. As per our approach, the activation function chosen for the convolution layers is ReLU (Rectified Linear Activation), which performs robust training of deep networks and leads to greater classification precision, and the last layer is Sigmoid (logistic function). For this, the class number of the last layer Nb_Classes equals 1 because we have just two classes (malware and non-malware “benign”). Indeed, the values mentioned above were chosen after several tests on our architecture. For this, the following table summarizes the characteristics of our proposed model for phase II (Detection & Classification) (see Table 3).

The model which we have chosen aims to classify the vector resulting from the dimensionality reduction step using stacked AutoEncoders. The best model evaluation parameter for this is to maximize accuracy. For this, we trained the network with multiple optimizers (Adadelta and Adam) to minimize the loss function using binary cross-entropy as a classification loss function, which is a technique that is optimized for binary classification. Furthermore, we tried to use the confusion matrix as model evaluation parameters. As a result, the table below provides a summary of the classification results obtained by our model with high accuracy (see Table 4).

As per our objective, the results were obtained from data divided into 80% training (12,028 samples) and 20% testing (3,008 samples) sets. An average of 1 min 53 s was required for this training. The training was provided over 100 epochs and a batch size of 50, since the loss rate in training and
testing has become 0.07%, and the level of accuracy has reached 98.50%. This shows that the increase in the number of applications assigned for the training set will cause more successful results in the classification of tested applications.

Figure 6 show the accuracy and loss in training and testing sets which remained very satisfactory (high) and reflect the good performance of the model with the different parameters using binary cross-entropy loss function and an Adadelta learning optimizer (see Figure 6).

As expected, our approach achieves the highest score of recall, precision, and F-measure 0.9818, 0.9874, and 0.9841 respectively, in which it was able to correctly detect 98.50% of the Android malware samples. Likewise, Figure 7 shows the confusion matrix of the test. High classification success was achieved in TP and TN values. However, the high values of the classification numbers

<table>
<thead>
<tr>
<th>Layer</th>
<th>Type</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv1d_1</td>
<td>(Conv1D)</td>
<td>(None, 96, 64)</td>
<td>384</td>
</tr>
<tr>
<td>Conv1d_2</td>
<td>(Conv1D)</td>
<td>(None, 92, 64)</td>
<td>20544</td>
</tr>
<tr>
<td>Max_Pooling 1d_1</td>
<td>(Max_Pooling 1D)</td>
<td>(None, 46, 64)</td>
<td>0</td>
</tr>
<tr>
<td>Flatten_1</td>
<td>(Flatten)</td>
<td>(None, 294)</td>
<td>0</td>
</tr>
<tr>
<td>Dense_1</td>
<td>(Dense)</td>
<td>(None, 100)</td>
<td>294500</td>
</tr>
<tr>
<td>Dense_2</td>
<td>(Dense)</td>
<td>(None, 1)</td>
<td>101</td>
</tr>
</tbody>
</table>

Table 3. The one-dimensional convolutional neural network (1D CNN) characteristics of our architecture

<table>
<thead>
<tr>
<th>Model</th>
<th>Optimizer</th>
<th>Loss function</th>
<th>Accuracy Training</th>
<th>Testing</th>
<th>Loss Training</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoEncoder +</td>
<td>Adadelta</td>
<td>MSE (Mean Squared Error)</td>
<td>0.9974</td>
<td>0.9840</td>
<td>0.0025</td>
<td>0.0130</td>
</tr>
<tr>
<td>CNN</td>
<td>Adadelta</td>
<td>Binary cross-entropy</td>
<td><strong>0.9991</strong></td>
<td><strong>0.9850</strong></td>
<td><strong>0.0044</strong></td>
<td><strong>0.0771</strong></td>
</tr>
<tr>
<td></td>
<td>Adam</td>
<td>Binary cross-entropy</td>
<td>0.9989</td>
<td>0.9847</td>
<td>0.0049</td>
<td>0.0796</td>
</tr>
</tbody>
</table>

Table 4. The results obtained by our model

Figure 6. The accuracy and loss in training and test datasets
for FP and FN indicate a very dangerous situation for Android application users. Similarly, FP and FN were obtained in only 45 of the total 3,008 tests performed in the present work, demonstrating the success of our proposed model (see Figure 7).

In this respect, the result will cause users not to use some useful and safe applications for no reason and, it will cause users to be at risk because some malicious applications are considered safe by them.

Besides that, we conducted a comparative study of our proposed approach with other existing malware detection and classification models on grounds of the approach used, a number of Android application (Drebin-215) datasets used in experimentation, methodology, features used for extraction and classification, and percentage accuracy achieved. In addition, our method extracts various types of features to reflect all aspects of applications as comprehensively as possible. The results in the table further verify that our proposed model achieves the best performance among all the methods (see Table 5).

In general, the results of the present study were evaluated comparatively according to similar parameters, we have seen that the proposed approach produced successful results with respect to other studies. However, much better results were produced in earlier studies as aforementioned, better results were obtained with slight differences with the studies using similar formalization. Very good performance metrics results were obtained according to studies with similar malware applications dataset sizes. This shows that existing detection and classification performance metrics values were taken one step further with the proposed approach in the present study. It suggests that the proposed approach has improved specificity compared with the previous works.

6. CONCLUSION

In today’s technological advancements world, where Android malware threatens users, taking security measures, detecting, and preventing the damages that malware software can cause emerge as an important field of study. The work conducted within the framework of this study is aimed
to detect and classify malware and benign applications targeting the Android operating system. According to (Andrea et al. 2022), there are two publishers for the Android operating system (Google Mail Services (GMS) and Open Handset Distribution (OHD)). Likewise, deep learning was widely applied to several applications and has proven to be a powerful machine learning tool for many complex problems such as detecting and classifying malware applications. As well, the application of deep learning for Android malware detection and classification is a hot developing research topic with many challenges. The fast advancements of new deep learning techniques will keep driving the enhancement of Android malware detection, classification, and analysis. This paper proposes a deep learning model for mobile malware detection and classification. It is based on Stacked AutoEncoder (SAE) and it could be used for reducing the data dimensionality in the case of mobile malware detection. Then, a Convolutional Neural Network (CNN) is utilized to detect malware through malware binary visualization. In the right context, the experiments were conducted using the real-world dataset called Drebin-215 which contains 15,036 app samples with 9,476 benign samples and 5,560 malware samples. The findings prove that the proposed combined approach of SAE and CNN achieves a malware category classification accuracy of about 98.5%. In the experimental results, a 0.9874 precision value, 0.9818 recall value, 0.9850 accuracy and 0.9841 F-measure value were obtained. Due to the high performance achieved by this model, we believe that these results help users to make decisions pertinent to a very dangerous situation for Android applications use. In contrast, we compared our results with the previous studies that used main features extraction to achieve better results and some of them used main static features.

For future works, we aim to develop a full system for malware via deep learning detection and classification with telecommunications and network research labs. In addition, the results may be improved using more datasets namely CICAndroidBot, Kharon, CICAAAGM2017, CICAndMal2017, and CICInvesAndMal2019 that can be added to improve the model performance, more experiments on hyperparameters settings in order to improve the approach, and more sophisticated feature extraction techniques based on deep learning that was developed for the android malware.

<table>
<thead>
<tr>
<th>Authors</th>
<th>Dataset # Samples</th>
<th>Methodology</th>
<th>Features Extraction</th>
<th>Performance Metrics and Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hasegawa &amp; Iyatomi, 2018</td>
<td>- Drebin - CIC-AndMal17</td>
<td>CNN</td>
<td>- 215 features - 80 features</td>
<td>95.4% Accuracy</td>
</tr>
<tr>
<td>Mudzfirah et al. 2019</td>
<td>- Drebin</td>
<td>CNN &amp; LSTM</td>
<td>215 features</td>
<td>96.76% Accuracy</td>
</tr>
<tr>
<td>Lu et al. 2021</td>
<td>- Drebin</td>
<td>CNN &amp; LSTM</td>
<td>215 features</td>
<td>95.83% Accuracy 95.24% Precision 96.15% Recall 95.69% F1-score</td>
</tr>
<tr>
<td>Atacak et al. 2022</td>
<td>- Drebin - VirusTotal</td>
<td>Feature Extraction (ANFIS) + CNN</td>
<td>/</td>
<td>92.00% Accuracy 92.15% Precision 92.00% Recall 92.01% F1-score</td>
</tr>
<tr>
<td>Proposed Approach</td>
<td>- Drebin</td>
<td>SAE and CNN</td>
<td>215 features</td>
<td>98.50% Accuracy 98.18% Recall 98.74% Precision 98.41% F-score</td>
</tr>
</tbody>
</table>
REFERENCES


Brahami Menauer is a professor of computer science in the National Polytechnic School of Oran (Algeria). He received a HDR degree in computer science from the University of Science and Technology of Oran (Algeria), in 2018. He received a Ph.D. in computer science from the University of Oran (Algeria). His research interests include Knowledge Management, Machine Learning, Business Intelligence, Deep Learning and Boolean Modeling. He is currently serving on various editorial positions such as Editorial Board Member in Journals such as International Journal of Information Systems in the Service Sector, International Journal of Information Retrieval Research, and Artificial Intelligence for Engineering Design, Analysis and Manufacturing.

Abdallah El Hadj Mohamed Islam received her first Degree in Computing at National Polytechnic School of Oran, Computer Systems Engineering department, Algeria. He furthered her master study in the area of Knowledge Management and Artificial Intelligence in 2019. Her research interests are knowledge management, machine learning and healthcare studies to improve her work.

Matta Nada received a HDR degree in computer science from the University of Technology of Troyes (France), in 2004. She is a professor at University of Technology of Troyes (UTT) in the Computer Science Department. His research interests include Information Systems, Knowledge Engineering, Knowledge Management, Knowledge capitalization, Memory Project, and Knowledge Tracing. She work has appeared in International Journal of Knowledge Management Research and Practices, International Journal on Advances in Information Sciences and Service Sciences, Journal of Communication and Information Sciences, International Journal of e-Collaboration on “Creativity, Innovation and e-Collaboration”, etc.