National College of Ireland

A Lightweight 1-D CNN Model to Detect Android Malware On the Mobile Phone

MSc Internship

MSc Cybersecurity

Sangameshwaran P.L

Student ID: x18174965

School of Computing

National College of Ireland

Supervisor: Mr. Imran Khan

National College of Ireland

Project Submission Sheet – 2019/2020

Student Name:	Sangameshwaran P. L		
Student ID:	X18174965		
Programme:	MSc Cybersecurity	**Year:**	2019-2020
Module:	MSc Internship		
Lecturer:	Mr. Imran Khan		
Submission DueDate:	August 17		
Project Title:	A Lightweight 1-D CNN Model To Detect Android Malware On The Mobile Phone		
Word Count:	6871	**Page Count:**	20

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the references section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature:	Sangameshwaran P. L
Date:	17.08.2020

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. Projects should be submitted to your Programme Coordinator.
3. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
4. You must ensure that all projects are submitted to your Programme Coordinator on or before the required submission date. **Late submissions will incur penalties.**
5. All projects must be submitted and passed in order to successfully complete the year. **Any project/assignment not submitted will be marked as a fail.**

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

A Lightweight 1-D CNN Model to Detect Android Malware on the Mobile Phone

P.L. Sangameshwaran

MSc. Cybersecurity

X18174965

ABSTRACT

The mobile device has become an integrated part of everyone's life. The mobile users store their sensitive and private information in their easy to carry handsets or mobile device (bank information, critical business documents, etc). Android malware is a very big concern for internet security researchers and there have been many research works performed for the detection of android malware on the server-side. The detection of malware on the server-side is not efficient and detection on the mobile device is required to enhance the detection of the loosely controlled android application market. The malware uses obfuscation or repackaging techniques to escape the conservative signature-based analysis. In this paper, a lightweight model using deep neural networks is proposed. The model performs static analysis with help of manifest properties, API calls and application category features. The proposed model leverages the concept of the NLP(Natural Language Processing) to make use of the 1-dimensional convolution neural network (CNN) for malware detection with less training time and pre-processing computational overhead. The lightweight model proposed outperformed commonly used machine learning and deep learning models with an accuracy of 95.50% and this can serve as a great starting point to use the 1-D CNN for effective malware detection on the mobile phone or IoT devices.

Keywords: Android malware, malware detection, deep neural network, 1-dimensional Convolution Neural Network(1D-CNN), Static Analysis, Feature-based, Smartphone Security, API calls, Manifest Properties

1. Introduction

The usage of the mobile device and their applications has started to increase every day with more users storing their personal or sensitive information like bank information, passwords, etc in the mobile device. The android operating system conquers the mobile operating system platforms with around 74.6% sales to the end-user [21],[24].

In today's digital world, the weakest link to internet security is due to the mobile device so most of the malware (Malicious Software) developers target the mobile devices to create a capital loss or information security issue to the mobile device users [7]. The Android market is specifically targeted by the malware developers as it does not have a central management setup like the apple store or I-store. The user tends to install the application from Google store, third party markets or APK (Android Package) file available on the internet or in the open trusting it to be safe for use [3]. Android malware will not target the mobile devices alone but also the Internet of Things (IoT) devices using the Android OS such as “Android Of Things” also known as “Brillo” [21] or try to

pivot to the devices connected to the mobile device operating on Android OS creating huge chaos to the users and the security engineers to protect against the modern malwares.

Android malware detection has become a huge problem due to the steady increase in the android application and android mobile devices installing them. The malwares developed with modern techniques are difficult to detect with the conservative signature-based detection. Machine learning is a promising solution and a good alternate to the signature-based technique for the detection of a huge amount of applications which is more than 3 million just in the Google Store [21]. Malwares have started to use a technique known as "Metamorphism" or obfuscation or repacking to run unidentified to the signature-based analysis method and many third-party markets still use a signature-based approach to detect malicious applications which are very easy to bypass [3]. The Malware detected using the machine learning on the server-side in the Elite market like Google store and many other good third party markets like "Anzhi(Chinese third party market)" cannot react in a reasonable time in detecting a new family of malware. For this reason, a last line of defense in the mobile is required to keep the user safe from malicious applications. The existing idea of uploading the malware to the server-side for detection before installing will create a lot of network overhead and attackers might obfuscate the application to act benignly while uploading to the server-side for detection, so it is not an effective mechanism to detect malware on the mobile phone [3].

The featured based technique has proved to be a promising solution over commonly used dynamic or signature-based technique due to its less computation cost than dynamic technique and accurate fast detection than signature-based technique [13]. In this paper, we present a novel approach that leverages the concept of NLP (Natural Language Processing) and makes use of the 1D-CNN of deep neural networks to give a lightweight model for the mobile device, that does effective malware detection with less training time [5]. The 1D-CNN model performs detection using Manifest properties and API calls obtained from the Drebin dataset along with the Application categories features obtained from Appbrain Statistics ethically. The model gives very good results when compared to other machine learning or deep learning models and also we addresses the **research question Can *the accuracy be improved with less training time by using a lightweight 1D-CNN model?*** as discussed in the later sections.

The paper is structured as follow: Section 2 gives a brief overview of various malware detection techniques, section 3 discusses various deep learning and machine learning techniques implemented with feature-based malware detection, section 4 gives detail description on the methodology opted to implement the proposed model, section 5 gives the design overview of the proposed model, section 6 has the detail evaluation of various implemented models and a brief discussion of the proposed model contribution and performance and finally section 7 gives a detail conclusion of the paper with limitations and future works possible.

2. An Overview Of Various Malware Detection Techniques

Malware is usually detected by Surface analysis or Signature-based technique, Dynamic analysis or Static analysis. The surface analysis investigates the character string included in the target file, The dynamic analysis runs the file in a controlled environment to analyze its behavior and the

static analysis will decompile and disassemble the file to get in-depth information on the file to classify as malware or benign[5]. A detailed review of each technique with the comparison between them is given below.

A. Signature-based technique

The most widely used technique for malware detection is the signature-based technique. In this technique, the tool will check for matching code patterns by "Code Hoisting" to give a "Confidence centric" of the file to decide its malicious potency and classify it as malware or benign [2],[17]. The cloud-based or external server-based technique to check for the malware on device or mobile phone by matching with malware signatures stored in them was time-consuming and produced a heavy network overhead [3]. The work by Deepak et al [11] on detection of malware on mobile phones using signature matching was a really good start to the detection of malware on the mobile phone but these detection techniques were circumvented by malware obfuscation technique or polymorphism of malware to act as a benign application to evade the detectors.

B. Feature-based technique

The feature-based technique works on the concept of extracting distinguish features from the testing file by decompiling and disassembling the file to get in-depth information on the file so the malware cannot use obfuscation or polymorphism techniques to escape from the detectors[5]. The extracted features are usually fed into a machine learning or deep learning model to classify as malware or benign. This technique has proved to be very promising and has also reduced the intervention of human action hence giving a fast and error-free result [4],[10].

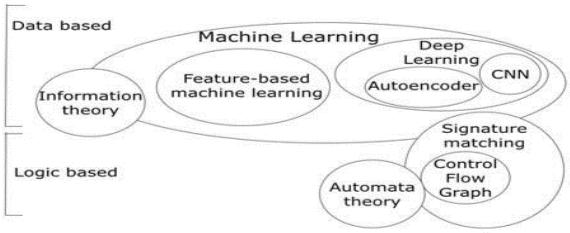


Figure 1.Overview techniques of malware detection[2]

C. Other techniques

Few approaches work with control flow graph or graph generation to measure distance which is used to check for the difference in functional similarity to classify as malware or not. The work by Monga et al [17] and Marion et al [19] generated a control flow graph of the testing file and compared the graph with known malware's generated control graph to classify as malware or not. This technique did find the malware trying to evade with obfuscation or polymorphic technique but the computation of graph for the large file was difficult and expensive making it an unpractical option for malware detection [10].

Another kind of technique used by kinder et al [18] used a concept known as model checking which was used to check the correctness of a system against a specification but the paper used it

for checking malicious chunks of code based on the behavior of the file to classify as malware or not [10],[18]. This technique failed to detect a few new kinds of malware or circumventing malware using modern techniques to evade the detector as they use stringent signature-based detection techniques.

The dynamic analysis is a technique that detects the malware better than the static techniques as discussed above but they are not much preferred for practical use as the malware needs to be executed in a controlled environment, unlike static technique which can classify as malware or benign without implementation [7]. The dynamic detection method requires more computational resources to be executed which makes it a bad choice of mobile device that works on low computational resources [20].

So, from the above discussions, we can see that the feature-based approach would be the right choice to design a lightweight malware detection model. The review of various Feature-based techniques supported by machine learning or deep learning along with their results is discussed in detail below.

3. Related Works

A detailed review of the related works was carried out based on the proposed concepts and methodologies. Many papers have done the static detection of android malware using Machine Learning and Deep Learning on the server-side and many have not analyzed on the mobile device, especially not much paper has researched the use of 1-D Convolution Neural Network(CNN) in the detection of malware. An overview of the various malware detection technique with a high-level comparison between each other is done in the below section.

The main idea between the use of deep learning for malware analysis is its ability to select the most relevant feature from the structure of the file and its robustness to detect unseen new data which facilitates it to be used for a large amount of data for malware detection. The repackaged or obfuscated malware can be completed understood and detected with dynamic analysis, but a major drawback of the technique is high computational resource is required. To overcome this issue Dong-Zie et al suggested a tool known as “DroidMat” that can detect the malware with extracted permission and API calls from the Apk file. The tool leveraged the concept of machine learning ie., used the Knn model to detect if malware or not[14]. The “DroidMat” achieved considerable accuracy, precision, recall and F1 score results when compared to other research works similar to it [9],[8],[1] but only varied in the choice of machine learning techniques used like SVM, Naïve Bayesian, Decision Tree(J48) and Random forest. The work by T.Ban et al[21] to detect malware using SVM considered additional features to permission and API calls and the work by D.Arp et al[22] was almost similar to wok by[9],[8],[1] and [14] but it used Drebin dataset to classify malware for a mobile device.[21] and [22] achieved an accuracy almost equal to “DroidMat” ie., on an average 95% with the use of the SVM classifier. Work by the above researches did remove the need for dynamic analysis to detect repackaged or obfuscated malware with considerable accuracy and a system that is driven in a fully automatic way. Even though leveraging the technique of machine learning contributed to the detection of malware with a less false positive rate the model was not preferred to work with a very large dataset as it left the large preprocessing

computational overhead when compared to the Deep learning models which gave better accuracy too.

Deep learning is a subset of machine learning that works on layers to learn features of data incrementally and removes the need for domain expertise and hardcore feature extraction. The preprocessing computational overhead is reduced, and higher accuracy is gained. The CNN (Convolution Neural Network) is a successful and promising deep learning model for malware detection. Cnn is a model that is mostly used for image classification or computer vision, where the work by M.Ganesh et al[4] and P.He et al [6] extracted permissions feature from the Apk file and converted them to images to be classified by 2D-CNN model, achieved an accuracy of 99.25% and 93.4% respectively.

The work by N.Xie et al[15] and Z. Wang et al[13] extracted various static features from the Apk file such as permissions, API calls, intent-filters, etc.. and converted the extracted features to feature matrix by embedding the features to make it fit input of 2-D CNN model. The work by W.Yuan et al[16] was similar to the work by [15] and [13] but it worked on the Drebin dataset to extract the features and generate feature matrix. The [16] achieved an accuracy of 93.39 % and gave a lightweight model that can be used for mobile or IoT devices. Another kind of hybrid approach using modern techniques like autoencoders or VAE [12] with CNN made it practically challenging as it required more data during the training phase [10]. "MobiDroid" by R.Feng et al[3] was another lightweight model that did malware detection on the mobile device using Manifest properties, API Calls and opcode sequence with help of 2D CNN. From the above discussion, we observed that the 2-D CNN gave a less computational overhead for preprocessing and also better accuracy when compared to machine learning but they had few drawbacks

- The conversion of the feature matrix to the image was a bad idea as it made the structure of the file to be lost and converting them into 2D space gave a mixture of vertical and horizontal proximity that had one which was meaningful and other which was meaningless.
- Developing an embedding space for the 2D CNN was sophisticated and had a lot of semantic problems.

To overcome the issue of the use of 2D CNN Hasegawa et al[5] and A.Sharma et al[10] leveraged the concept of the NLP(Natural Language Processing) community which was used for the sentence classification. This concept was used for the classification of malware using the one-dimensional filter or 1D CNN to classify the Apk file as malware or not with the help of the code extracted from the testing file. The work gave good accuracy as same as 2D CNN and even though it considered the embedding of space it has fewer semantic complications or less computational overhead while pre-processing when compared to 2D CNN. The only drawback of the above works was disassembling a repackaged or obfuscated Apk file was complicated and time-consuming to extract the code for classification. N-gram technique [10],[5] for feature extraction did not make sure if all the important features were considered for the detection of malware. To best of my knowledge, there is not much paper that has done malware detection using 1D CNN and none have analyzed malware using Manifest properties and API calls with 1D CNN like the proposed model. The proposed model was inspired by the work of "Mobidroid" [3] which was a lightweight 2D CNN model to detect malware on the mobile phone. The Mobidroid accuracy, f1-score, precision

and recall were improved by considering an extra feature application category that helps to classify the malware even better by understanding the reason for the permission request by the application to dangerous permissions[21] and the 2D CNN was replaced by 1D CNN to overcome the drawbacks discussed above. The model was also compared with other machine learning and deep learning like MLP, SVM and RF [1],[8],[9],[14]. A detailed explanation of the proposed model is done in the below sections.

4. Methodology

The research methodology has followed the various steps of the Cross-industry process for data mining (CRISP-DM) to implement and evaluate the working of the proposed model. The CRISP-DM steps are shown in figure 2. below. The various stages in CRISP-DM are explained in detail below.

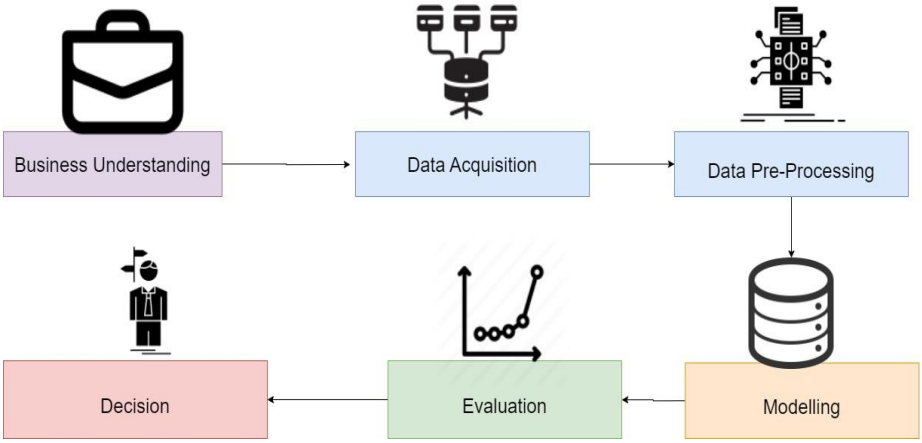


Figure 2. Overview of the CRISP-DM methodology

4.1 Business Requirement

Android malware is growing day by day due to the loosely coupled and popularity of the Adroid market. So the detection of the malware on the server-side alone won't be enough and the second level of detection on the mobile phone is required with a lightweight model to detect the malware. There is a need to have a very less false positive rate while detecting malware so many works have been done on detecting malware with the help of machine learning or deep learning.

Many have detected malware based on their Opcode, API calls and Manifest properties. To best of my knowledge, none have considered the Application category, API calls, Manifest properties and 1D CNN for the malware detection for the mobile device. This might be a good start to consider 1D CNN and Application category into account for better android malware detection with less pre-processing computational overhead and less training time. Prevent its widespread on ever-developing mobile or IoT devices.

4.2 Data Acquisition

Various malware datasets are available in open source for research purposes. After complete review over various malware datasets like Drebin dataset, Androzoo, Genome project, Contagio, etc.., and found that the Drebin dataset would be the right one for the proposed model. The Drebin dataset has 5560 android malware Apk samples from over 179 malware families and 123,453 benign samples. The proposed model is a starter to a new flavor of malware detection on the mobile device using 1D CNN, so a smaller dataset like Drebin is considered over Androzoo which is a bigger dataset similar to the Drebin dataset.

The dataset was downloaded from the Drebin dataset official portal[1][25], the downloaded feature vector had both the malware and malicious file together in. file format with 129,013 files in it saved in name of their hash values. To split the malicious file from benign files we had to download another file from the portal that had the hash values of the malware with their malware family. After splitting the dataset, a balanced dataset with 5560 samples of benign and 5560 samples of malware was created. We need only a few important Manifest properties, API call features and also need to mock with various application categories of Android play store obtained from Appbrain statistics[2].So the dataset requires a few preprocessing steps and labeling to be performed, before using it for the model which are explained in detail in the below section.

4.3 Data Pre-Processing

This is the most important step in the data preparation to be applied for the 1D CNN and other machine learning models that will be used for comparison with the proposed model. The Data preprocessing computation overhead was reduced, and the feature engineering of the dataset was simple to do as we were using 1D CNN instead of 2D CNN. The steps involved in data preprocessing is explained in detail below.

4.3.1 Data Extraction and Conversion

The final dataset with 11,120 samples of 5560 benign and 5560 malware Apk files was all in .file format. All the files were converted to .txt format and contents of all the .txt files where transferred to another file to generate the file .csv file. The contents of the .txt file were in key-value pairs with values of manifest properties and API Calls. From manifest properties uses-permission (real_permission and permission), intent-filters and uses-feature were extracted. The uses-permissions have the android system permissions that need to be requested by the application before accessing critical or sensitive information. The application component can access the device to perform various activities using intent-filter, where each intent filter has an intent object that carries the request of the applications(this may be used to do malicious activities like sending messages, read pictures, etc.). The uses-features have the information of the hardware feature requested by the application like audio, Bluetooth, WiFi, etc. Along with the above features, API calls are also extracted [3].

1 https://www.sec.cs.tu-bs.de/~danarp/drebin/

2 https://www.appbrain.com/stats/android-market-app-categories

All the above-discussed features are filtered using the excel and transposed to columns with rows value being the 11,120 samples filename which was got with the help of Kutools plugin for excel. To make the labeling easy we append '_bw' to the name of benign sample files.

4.3.2 Feature Engineering

The final dataset obtained after pre-processing needs to transformed feature matrix so it can be applied to 1D CNN and other machine learning algorithms with ease. Feature selection is not applied to the dataset so that we can select all the features of the dataset for better malware detection and not be doubtful if any core features are missed out if feature selection was applied. Finally, the dataset was labeled and mocked with the application category to be ready to be inputted to the 1D CNN model.

4.3.2.1 Feature Matrix

The feature matrix with 1s and 0s is generated with help of one-hot encoding, where if the permission or API call is present in the sample file, which is the row name 1 is added to the corresponding column or a 0 is inserted and if the file name (row name)is without '_bw' a 'target' column is added in the end and given a value 0 denoting it as a malware and if '_bw' is present a 1 is inserted in the target column denoting it as benign to complete the labeling process for training for the deep learning or machine learning model. We did not get any malware dataset considering application category to the best of my knowledge, so the various application categories of android applications in the play store are got from Asppbrain statistics. They were around 49 application category and each application category is give an integer value from 1 to 49. With the help of python panda's data frame and NumPy rand function, we put random values from 1 to 49 to the 'Application _category' column for 11,120-row names for the mocking purpose. Following the same process above, 2 more .csv files were created including the application category in them for manifest properties and API calls separately. So, at the end of the data pre-processing step, three .csv files are generated ie., one containing manifest properties, one containing API calls and the other with both manifest properties and API calls. A detailed description of the generated dataset is given below.

4.3.3 Data Description

The final datasets generated had the following number of rows and columns as explained below:

- The dataset with Manifest properties alone had 11,120 rows with 934 columns in them. (including the application_category and target column).
- The dataset with API calls alone had 11,120 rows with 197 columns in them. (including the application_category and target column).
- The combined dataset of Manifest properties and API calls had 11,120 rows with 1129 columns in them. (including the application_category and target column).

The above dataset described will be applied to the various machine learning and deep learning models whose implementation is explained in the below sections.

5. Design Specification

Figure 3 shows the three-layer design flow that has been followed for the detection of malware. Each layer functions is explained in detail below:

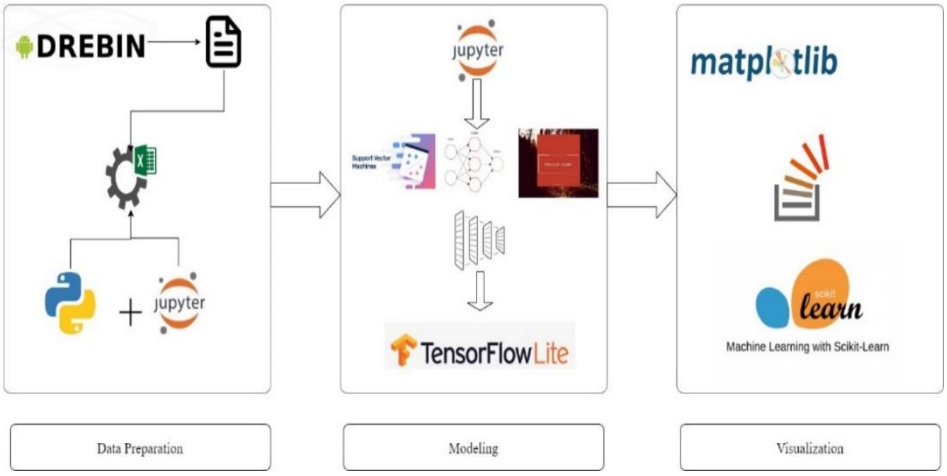


Figure 3. Android Malware detection-Design Flow

- Data Preparation: In this layer, the data set for the proposed model is acquired from the Drebin portal and the necessary data conversion is performed on it so that the required features can be extracted. Preprocessing of the dataset and mocking with the application category is performed to generate three .csv files with manifest properties, API calls alone and manifest properties, API calls combined to be given to various deep learning and machine learning models.
- Modeling: In this layer the various machine learning model ie., SVM and RF and Deep learning model ie., 1D CNN and MLP were executed. The model was evaluated with few metrics such as F1score, Accuracy, Precision and Recall. All the models were run on the Jupyter notebook in the local machine. After training the models the CNN model was converted to a TensorFlow Lite model with. tflite extension.
- Visualization: The model was not only evaluated with the above-mentioned metrics, but its results were also visually represented in the form of the confusion matrix and graph for a better understanding of the reader.

The various steps involved in the Implementation of the model along with its metrics calculation and its results plot is explained in detail in the below sections.

6. Implementation

Each model was applied to three .csv files, whose description is given in the next section and the models were subjected to hyperparameter tuning to get the best results out of it. Before applying the model to any of the deep learning or machine learning models, the dataset was split into 90% for training and 10% for testing and 70% training and 30% for testing for deep learning and machine learning models respectively. The splitting was achieved through “train_test_split”

function of the “sklearn.model_selection” package. A detailed description of the implementation of the machine learning model SVM and RF and deep learning model 1D CNN and MLP is given in the below sections.

6.1 Deployed Machine Learning

Supervised machine learning model like Support Vector Machine (SVM) and Random Forest (RF) is used on the three datasets generated. The model works in the concept that the target variable or dependent variable is predicted with the help of independent variables or the static features extracted from the Apk file as explained below.

6.1.1 Support Vector Machine(SVM)

This classification method plots the value of each predictor in an N-dimensional space where N is the number of features or predictors considered and the value of each feature is the value of a particular coordinate. After splitting the datasets generated as explained above, we apply the SVM model with default parameter values. The SVM model is executed three times for the three datasets generated and the snippet of the code is given below in figure 4.

```
#Import svm model
from sklearn import svm

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(x_train, y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

Figure 4. Code Snippet of SVM Model

6.1.2 Random Forest(RF)

The Random Forest (RF) has a group of Decision trees (Forest) where each decision tree will give a classification result based on the attributes and the model will choose the tree which has the maximum number of votes. The model was executed with few hyperparameters tuning value ie., n_estimators=100 will decide the number of trees that needs to be built for the detection, max_depth=10 will give the depth of each decision tree, Criterion=’Gini’ will give the incorrect classification done by the classifier for a random variable considered. Like above, the RF model was run thrice for the three datasets generated and the code snippet of the RF model is shown in Figure 5 below.

```
from sklearn.ensemble import RandomForestClassifier

#Create a Randomforest Classifier
clf=RandomForestClassifier(n_estimators=100,max_depth=10,criterion='gini')

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(x_train,y_train)
y_pred=clf.predict(x_test)
y_pred1=clf.predict(x_train)
```

Figure 5. Code Snippet of RF Model

6.2 Deployed Deep Learning

The deep learning will use the concept of Artificial neural network(ANN) that will mirror the way the human brain thinks. They are a self-learning algorithm that will generate a data pattern by extracting features from unknown elements in the input distribution. They self train at multiple levels to build a model for various tasks. Various Deep neural models are available but the one best suited for our experiment was MLP(Multilayer Perceptron) and Convolution Neural Network(CNN) whose implementation is explained in detail below.

6.2.1 Multilayer Perceptron(MLP)

The MLP is a feed-forward supervised learning algorithm with at least two hidden layers to generate a set of outputs from a given set of unknown inputs. The MLP can handle well large features of the input particularly the non-linear ones. The MLP classifier is run with the following parameters: hidden_layer_sizes=(150,100,50) will decide the number of layers and nodes to be set for the neural network model, max_iter=20 will decide the number epochs the model has to run, activation=relu is the default activation function set by the classifier to its hidden layers to help the model understand complex patterns in input distribution, random_state=1 will initialize the internal random number generator that will decide the weight of the nodes. The MLP model will following parameters is run thrice for the three datasets generated whose snippet is shown in figure 6 below.

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(150, 100, 50), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=20,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=1, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
```

Figure 6. The snippet of MLP Model

6.2.2 Convolution Neural Network(CNN)

CNN is similar to the MLP which is a feed-forward supervised multi-layer neural network that uses perceptron to analyze and learn about the data. Usually, CNN is used for image classification using 2D CNN or 3D CNN but in this experiment, we have leveraged the concept of Natural language processing to make use of the 1D CNN. 1D CNN works similarly to the 2D CNN or 3D CNN but only vary in the structure of input data and the way the filter or kernel moves across to data to analyze and learn about it. In our experiment we have made few hyperparameter tuning to get the best results from the 1D-CNN will less training time. The proposed customized 1D CNN model summary is shown in figure 7 below. The model has two single dimension convolution layer ie., this the first layer of the model which will extract the features of the input data with the kernel size specified as 5 for the first convolution layer for broad feature extraction and 3 for the second convolution layer to narrow down the extracted feature. The output filter for both the convolution layer is specified as 128. The activation function Relu which is the most preferred successful non-linearity function is applied to the parameters after convolution to induce non-

linearity to the parameters. Valid padding is done to make sure the output of the convolution has a smaller dimension than input data before passing to the fully connected layer. The convolution layer is followed by a max-pooling layer that almost performs the same function as convolution but in a specific way by taking the maximum value of the filter region or pool size=3 to reduce the dimensionality of network and computational cost. The same layers are repeated with the same parametric values and are flattened to a 1-dimensional array before feeding into the dropout layer with value 0.8 to prevent overfitting. The parameters are then fed into the dense layers or hidden layers or fully connected with output space as 64,32,16 and the final layer has as output space or units value as 2 for malware or benign with activation function as "softmax" to represent the probability distribution of possible outcomes of the model. After execution of all the layers if the weight of malware node ie., 0 is greater than benign node ie., 1 in the final layer it will classify the input as malware or vice-versa. The model is run for a batch size of 256 for each epoch and the model is run for 20 epochs with "adam" optimizer to get the most accurate result possible by reducing loss, and since our dataset is mutually exclusive ie., each sample belongs to a particular class which is a malware or benign we use the "sparse categorical cross-entropy" loss function. The accuracy metrics for training and the testing dataset (validation accuracy) is calculated while running the model for each epoch. Code snippet of 1D CNN is shown in figure 8 below which was applied thrice for 3 datasets generated.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 191, 128)	768
conv1d_1 (Conv1D)	(None, 189, 128)	49280
max_pooling1d (MaxPooling1D)	(None, 63, 128)	0
conv1d_2 (Conv1D)	(None, 59, 128)	82048
conv1d_3 (Conv1D)	(None, 57, 128)	49280
max_pooling1d_1 (MaxPooling1	(None, 19, 128)	0
flatten (Flatten)	(None, 2432)	0
dropout (Dropout)	(None, 2432)	0
dense (Dense)	(None, 64)	155712
dense_1 (Dense)	(None, 32)	2080
dense_2 (Dense)	(None, 16)	528
dense_3 (Dense)	(None, 2)	34

Figure 7. Model summary of CNN model

```
model = Sequential()
intput_shape=(x_train.shape[1], 1)
model.add(Conv1D(128, kernel_size=5,padding = 'valid',activation='relu', input_shape=input_shape))
model.add(Conv1D(128,kernel_size=3,padding = 'valid', activation='relu'))
model.add(MaxPooling1D(pool_size=(3)))
model.add(Conv1D(128,kernel_size=5,padding = 'valid', activation='relu'))
model.add(Conv1D(128,kernel_size=3,padding = 'valid', activation='relu'))
model.add(MaxPooling1D(pool_size=(3)))
model.add(Flatten())
model.add(Dropout(0.8))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.summary()

model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

history = model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size, validation_data=(x_test,y_test),validation_split=
                    verbose=2, shuffle=False)
```

Figure 8. Code snippet of CNN Model

The above generated trained Keras models of CNN were converted to the TensorFlow Lite model(.tflite extension) by flowing the TensorFlow Lite guide[3] to give a lightweight model that will be suitable for the mobile device or IoT devices. The results, evaluation and comparison of the above-executed model along with their graphs and confusion matrix are explained in detail in the below section.

7. Evaluation

The deep learning and machine learning models are evaluated with a few metrics calculated with the help of a confusion matrix and "sklearn.metrics" package. Graph and Confusion matrix is also plotted for the CNN model when applied to the 3 datasets generated above.

A confusion matrix is generated for each of the models when executed. The confusion matrix is a table that is used to describe the performance of the model. Metrics like Accuracy, F1-Score, Precision, Recall and especially AUC-ROC curve can be calculated using the confusion matrix. The confusion matrix has the following parameters [23]:

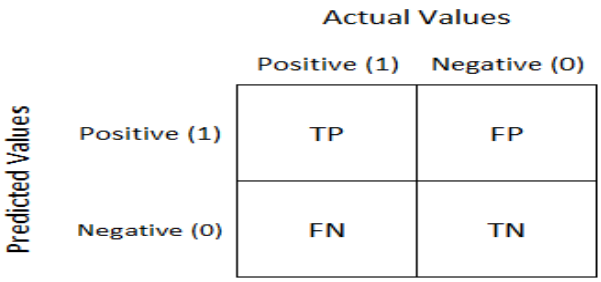


Figure 9. Overview of Confusion Matrix[4]

- TP (True Positive): When the predicted value is true, and the actual value is also true. In our case, if we predict it as malware, it is a malware or vice-versa.
- TN (True Negative): When the predicted value is false, but the actual value is true. In our case, if we predict it as not a malware but it is a malware or vice-versa.
- FN (False Negative): When the predicted value is false, and the actual value is also false. In our case, if we predict it as not malware, it is not a malware or vice-versa.
- FP (False Positive): When the predicted value is true, but the actual value is false. In our case, if we predict it as malware but it is a benign application or vice-versa.

With the above parameters we can calculate the metrics considered as explained below:

a) Accuracy: Accuracy is the ratio of the sum of the number of all correct predictions and by the total number of the data ie.,

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

b) Precision: Precision is the ratio of the number of correct positive predictions by total positive prediction ie.,

$$\text{Precision} = \frac{TP}{TP+FP}$$

3 https://www.tensorflow.org/lite/guide

4 https://towardsdatascience.com/understanding-confusion-matrix

c) Recall: Recall or Sensitivity(SN) or True positive rate is the ratio of the number of correct positive predictions by the total number of positives ie.,

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}}$$

d) F1-Score: The F1-Score is the weighted average or harmonic mean of the precision and the recall that considers the False-positive and False-negative score ie.,

$$\text{F1} - \text{Score} = \frac{2*(\text{Recall}*\text{Precision})}{\text{Recall+Precision}}$$

7.1 SVM, RF and MLP

The model after applied with SVM, MLP and RF on the generated three datasets is evaluated with the help of the confusion matrix to get the following metrics such as Accuracy, F1-Score, Precision, Recall or use "sklearn.metrics" package for the calculation of the considered metrics and its results is given in table 1 below for the performed three experiments on three datasets generated for each model.

Table 1. Evaluation results of SVM, MLP and RF

Model Name	*RF*			*SVM*			*MLP*		
	API	*MANIFEST*	*MANIFEST AND API*	*API*	*MANIFEST*	*MANIFEST AND API*	*API*	*MANIFEST*	*MANIFEST AND API*
Accuracy	*86.18*	*91.54*	*92.35*	*86.90*	*93.28*	*94.06*	*89.32*	*93.76*	*93.91*
F1-Score	*87.16*	*91.90*	*92.67*	*87.65*	*93.49*	*94.23*	*89.79*	*93.75*	*93.93*
Precision	*82.63*	*89.68*	*90.42*	*84.21*	*92.25*	*93.25*	*87.48*	*95.64*	*95.38*
Recall	*92.22*	*94.22*	*95.05*	*91.40*	*94.75*	*95.22*	*92.22*	*91.93*	*92.52*

The above results in table 1 show that all the models give better accuracy results when applied to the combined features of API calls and Manifest properties. The SVM model provides the best accuracy with 94.06% followed by the MLP which gave a good accuracy almost equal to SVM along with the highest precision of 95.38%. Even though these models gave good results or considerable accuracy for malware detection they were outperformed by the proposed customized 1D CNN model as explained in detail below.

7.2 1-Dimensional-CNN

The execution of the CNN model for each of the three datasets is plotted with a confusion matrix, validation accuracy or accuracy of test data vs prediction accuracy or train data accuracy graph is plotted and precision vs recall curve is plotted as in the below sections.

7.2.1.1 API Dataset, Manifest Dataset and Manifest and API Dataset

The CNN model when ran on the API dataset with 11,120 rows and 197 columns, Manifest dataset with 11,120 rows and 934 columns and the combined Manifest and API call dataset with 11,120 rows and 1129 columns for 20 epochs gave the following results for metrics considered whose values are in table 3 can be calculated using the confusion matrices plotted in table 2 or with the "sklearn.metrics" package which is used in this experiment.

Table 2. Confusion Matrices of three datasets

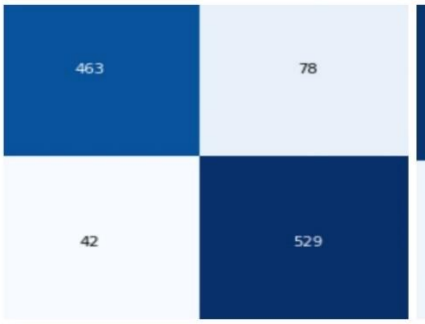
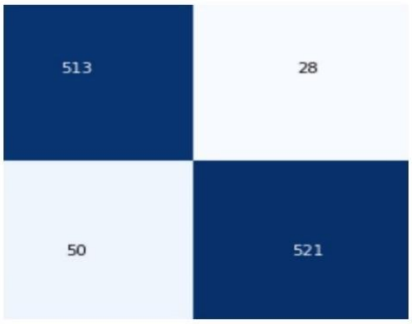
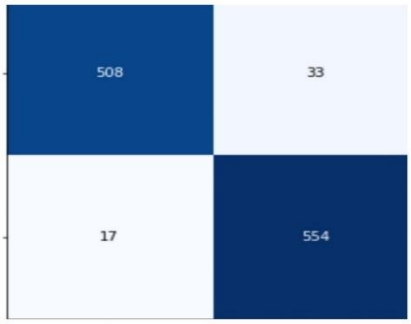
API	*Manifest*	*Manifest and API*
		

Table 3. Evaluation results of the three Datasets

API Dataset(CNN)	Accuracy	F1-Score	Precision	Recall
	89.12	89.73	84.38	92.66
Manifest Dataset(CNN)	Accuracy	F1-Score	Precision	Recall
	92.99	93.05	91.08	91.24
Manifest and API Dataset(CNN)	Accuracy	F1-Score	Precision	Recall
	95.50	95.68	93.09	97.02

The CNN model is performed the best for the combined dataset API and Manifest when compared to manifest and API dataset alone. We can see that TP, TN rates are high and FP, FN rates are low as excepted in the confusion matrix. Figure 10,11,12 gives the plot of test vs train data accuracy for API, manifest and API and Manifest model respectively. We see that the test and train plot are linear close to each other ie validation accuracy is almost equal to training showing that overfitting was avoided.

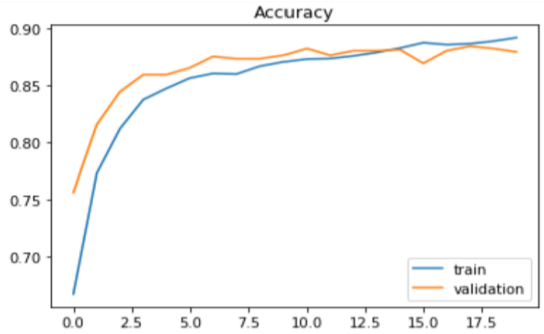


Figure 10. Validation vs Training Accuracy(API-CNN)

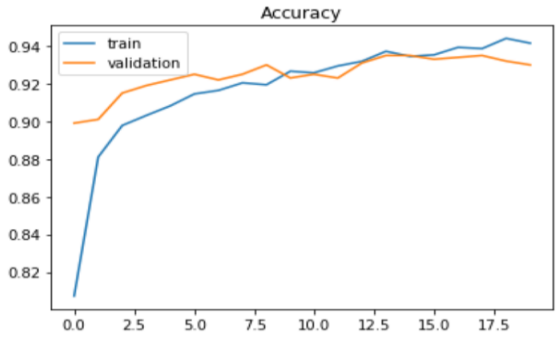


Figure 11. Validation vs Training Accuracy(Manifest-CNN)

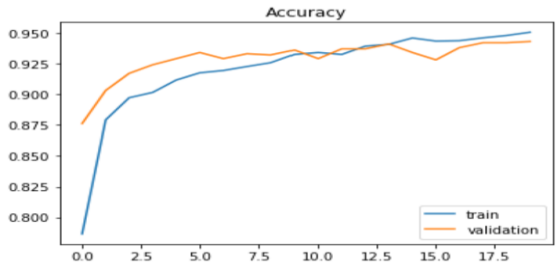


Figure 12. Validation vs Training Accuracy (Manifest-API-CNN)

Finally, the Precision vs Recall curve is plotted (figure 13,14,15) which gives the AUC curve for the models respectively. The curve value was 94.6%, 97.8% and 98.0% for API, Manifest and API and Manifest dataset respectively denoting high precision value ie., less false positive rate and high recall value ie., less false negative showing that it is a good model.

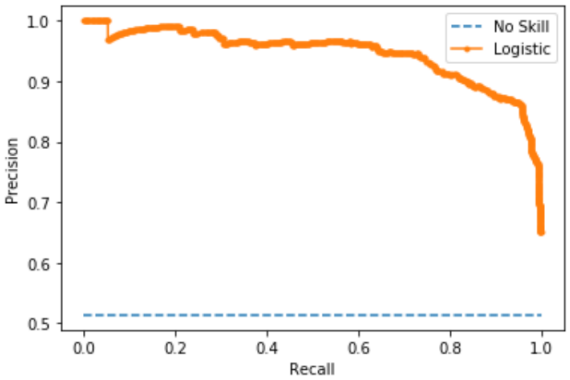


Figure 13. Precision-Recall Curve(API-CNN)

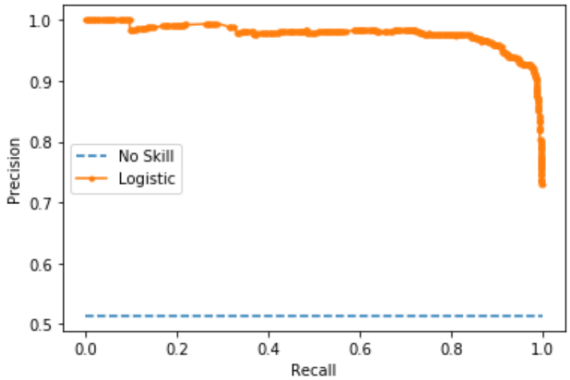


Figure 14. Precision-Recall Curve(Manifest-CNN)

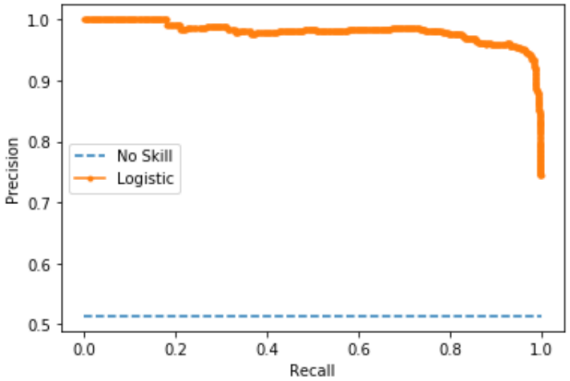


Figure 15. Precision-Recall Curve (Manifest-API-CNN)

7.3 DISCUSSION

This paper was implemented in an idea to improve the detection of Android malware for the mobile phone by developing a lightweight model. Unlike any other researches that utilized 2D CNN for the detection of malware while using the Manifest properties and API calls, this paper uses 1D CNN and also included an additional feature known as application category to improve the detection capability of the model. The adoption of 1D CNN reduced the pre-processing computational overhead and also the training time with better results when compared with the inspired work "Mobidroid" [3] by R.Feng et al. The model was also compared with other machine learning and deep learning models, where the proposed 1D-CNN model gave the best results.

The model was run on three kinds of datasets and was found that when the number of feature considerations increases better results are achieved, so the combined dataset of application category, API calls and Manifest properties gave the best result for SVM, MLP, RF and 1D-CNN. The 1D-CNN model outperformed the RF model in all the metrics and when compared to SVM it outperformed it all the metrics except for precision with almost the same value. When compared to MLP which also a deep learning model it gave better accuracy, very high recall and F1-Score with values 95.50%, 97.02%,95.68% respectively, but it failed to attain a better precision than MLP model which had a value of 93.09% when compared to MLP's 95.38%. When the proposed model is compared with the inspired work of "Mobidroid" whose value is shown in Table 4 below. The proposed model did not perform very well for the API call feature but it drastically outperformed for the manifest_properties and combined manifest and API dataset.

Table 4. Evaluation results of the Mobidroid model

API Dataset(CNN)	Accuracy	Precision	Recall
	92.00	92.00	92.00
Manifest Dataset(CNN)	Accuracy	Precision	Recall
	77.65	77.47	77.47
Manifest and API Dataset(CNN)	Accuracy	Precision	Recall
	90.37	90.37	90.37

When compared to Mobidroid, the proposed model has considered an additional feature application category for better detection and additional F1-Score metric for better evaluation of the model. As stated the model had very less training time of 20 epochs and less preprocessing computational overhead when compared to Mobidroid which had a long training time and preprocessing computational overhead. Finally, the model was converted to a lightweight model with only 1.27 megabytes (MB) suitable for deployment in mobile or IoT devices with the help of the TensorFlow Lite guide.

8. Conclusion

The research was performed for the detection of the Android malware with a lightweight model for mobile device or IoT device using 1D-CNN. The proposed model outperformed the other machine learning and deep learning models considered ie., MLP, SVM and RF. The results show

that when more features are considered ie., Manifest and API combined dataset (11,120 x 1129) better accuracy is obtained which is very important for malware detection. In malware detection, false positive rate (FP) should be very less because a wrong detection will be held to high capital cost to remedy it. The confusion matrix shows that the model has a very few false-positive rates, hence proving to be a suitable model for malware detection. The model also had less training time and less preprocessing computational overhead when compared to the 2D CNN model.

Limitations: Due to the time constraints it was not possible to extract the opcode sequence from the Apk file, so was not able to show my improvement over inspired Mobidroid model in the aspect of the opcode sequence feature and was not able to create an application to deploy the trained proposed model in the mobile device to see how it work in real for better representation of the model working.

The model has performed well for this small Drebin dataset with just 5,560 malware and 5,560 benign samples considered. In the future, the model can be tested on a large dataset like the Androzoo and its performance can be analyzed. More features like opcode, application cluster, etc., can be considered and the model can be implemented on the mobile device to demonstrate its real-time working.

Acknowledgment

I would like to thank my research supervisor Imran Khan for his relentless support and motivation throughout this research work. For 13 weeks, my supervisor assisted me with all the queries and guided me on the right path when I was lost to achieve this milestone in my career. I must sincerely thank my family for their exceptional love, constant understanding and support and the college for supporting me with abundant resources for the thesis purpose. Lastly, thanks to my data analytics friends who helped me at tough times during implementation and especially thanks to the School of Computing for this wonderful opportunity.

Reference

[1] P. P. K. Chan and W.-K. Song, “Static Detection of Android malware by using permissions and API calls,” 2014 International Conference on Machine Learning and Cybernetics, 2014.

[2] M. Christodorescu, S. Jha, S. Seshia, D. Song, and R. Bryant, “Semantics-aware malware detection,” 2005 IEEE Symposium on Security and Privacy (S&P05), 2005.

[3] R. Feng, S. Chen, X. Xie, L. Ma, G. Meng, Y. Liu, and S.-W. Lin, “MobiDroid: A Performance-Sensitive Malware Detection System on Mobile Platform,” 2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS), 2019.

[4] M. Ganesh, P. Pednekar, P. Prabhuswamy, D. S. Nair, Y. Park, and H. Jeon, "CNN-Based Android Malware Detection," 2017 International Conference on Software Security and Assurance (ICSSA), 2017.

[5] Hasegawa and H. Iyatomi, "One-dimensional convolutional neural networks for Android malware detection," 2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA), 2018.

[6] P. He and G. Gan, "Android Malicious App Detection Based on CNN Deep Learning Algorithm," IOP Conference Series: Earth and Environmental Science, vol. 428, p. 012061, 2020.

[7] "A Friend or a Foe? Detecting Malware using Memory and CPU ..." [Online]. Available: https://www.researchgate.net/publication/307879187_A_Friend_or_a_Foe_Detecting_Malware_using_Memory_and_CPU_Features. [Accessed: 04-Aug-2020].

[8] N. Peiravian and X. Zhu, "Machine Learning for Android Malware Detection Using Permission and API Calls," 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, 2013.

[9] S. K. Dash and A. Sharma, "Mining API Calls and Permissions for Android Malware Detection," Cryptology and Network Security Lecture Notes in Computer Science, pp. 191–205, 2014.

[10] P. Malacaria, A. Sharma, and M. Khouzani, "Malware Detection Using 1-Dimensional Convolutional Neural Networks," 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), 2019.

[11] Venugopal and G. Hu, "Efficient Signature Based Malware Detection on Mobile Devices," Mobile Information Systems, vol. 4, no. 1, pp. 33–49, 2008.

[12] W. Wang, M. Zhao, and J. Wang, “Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network,” Journal of Ambient Intelligence and Humanized Computing, vol. 10, no. 8, pp. 3035–3043, 2018.

[13] Z. Wang, G. Li, Y. Chi, J. Zhang, T. Yang, and Q. Liu, “Android Malware Detection Based on Convolutional Neural Networks,” Proceedings of the 3rd International Conference on Computer Science and Application Engineering - CSAE 2019, 2019.

[14] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, “DroidMat: Android Malware Detection through Manifest and API Calls Tracing,” 2012 Seventh Asia Joint Conference on Information Security, 2012.

[15] N. Xie, “Andro_MD: Android Malware Detection based on Convolutional Neural Networks,” International Journal of Performability Engineering, 2018.

[16] W. Yuan, Y. Jiang, H. Li, and M. Cai, “A Lightweight On-Device Detection Method for Android Malware,” IEEE Transactions on Systems, Man, and Cybernetics: Systems, pp. 1–12, 2020.

[17] D. Bruschi, L. Martignoni, and M. Monga, “Detecting Self-mutating Malware Using Control-Flow Graph Matching,” Detection of Intrusions and Malware & Vulnerability Assessment Lecture Notes in Computer Science, pp. 129–143, 2006.

[18] J. Kinder, S. Katzenbeisser, C. Schallhart, and H. Veith, “Detecting Malicious Code by Model Checking,” Detection of Intrusions and Malware, and Vulnerability Assessment Lecture Notes in Computer Science, pp. 174–187, 2005.

[19] G. Bonfante, M. Kaczmarek, and J.-Y. Marion, “Control Flow Graphs as Malware Signatures,” Inria, 03-Oct-2007. [Online]. Available: https://hal.inria.fr/inria-00176235. [Accessed: 04-Aug-2020].

[20] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, “DL-Droid: Deep learning based android malware detection using real devices,” Computers & Security, 12-Nov-2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404819300161. [Accessed: 04-Aug-2020].

[21] T. Takahashi and T. Ban, “Android Application Analysis Using Machine Learning Techniques,” SpringerLink, 01-Jan-1970. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-98842-9_7. [Accessed: 05-Aug-2020].

[22] “DREBIN: Effective and Explainable Detection of Android ...” [Online]. Available: https://www.researchgate.net/publication/264785935_DREBIN_Effective_and_Explainable_Detection_of_Android_Malware_in_Your_Pocket. [Accessed: 05-Aug-2020].

[23] Narkhede, Sarang. “Understanding Confusion Matrix.” Medium, Towards Data Science, 29 Aug. 2019, towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62#:~:text=It is extremely useful for measuring Recall, Precision, Specificity, Accuracy.

[24] “Mobile Operating System Market Share Worldwide.” StatCounter Global Stats, gs.statcounter.com/os-market-share/mobile/worldwide/2019.

[25] M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck, and J. Hoffmann, “Mobile-sandbox,” Proceedings of the 28th Annual ACM Symposium on Applied Computing - SAC 13, 2013.