Documentation

1. **Executive Summary:**

   In this part, we have tried to address the problem of security evaluation of android devices using Machine Learning techniques. Machine learning is the scientific study of algorithms and statistical models that computer/android systems use in order to perform a specific task effectively without using explicit instructions, relying on patterns and inference instead. Such algorithms operate by building a model based on inputs and using that to make predictions or decisions, rather than following only explicitly programmed instructions. To be specific, we have used supervised Machine Learning technique named Artificial Neural Network.

   Here, we have used 11 metrics to evaluate security system of an android system. These metrics are some of the characteristics of a mobile device like its settings, locks applied, API level etc. The data is generated by taking all the possible combinations of inputs from the system which we have already built using JESS Rule Engine. This data is then provided as inputs the Artificial Neural Network (ANN) which comprises of multiple layers. The Artificial Neural Network trains on this data and predicts the evaluation scores of an android system. In this project, the Android System is developed using a tool named Android Studio with a Java, an object oriented language. A sophisticated and easy to understand Graphical User Interface (GUI) is also provided since GUI makes it easy for a layman to evaluate the security system of his/her android device. Based on the security of the provided Android device, after clicking on the evaluate button, program approxes the evaluation score.

2. **Requirements**

   Hardware requirements:

   - A laptop or computer
   - 4 GB RAM
   - A hard disk
   - Internet connection is required

   Software requirements:

   - Java minimum version 7
   - Java SDK
   - Minimum Sdk Version 16
   - Android Studio
   - Python 3
   - Tensorflow
   - Keras
   - Pandas
   - Numpy
   - Seaborn

- Matplotlib
- io
- __future__
- Pathlib

3. **Implementation**

We have built an Artificial Neural Network using Tensorflow and Keras Deep Learning framework. The model generated was then converted to a tflite file which makes it easy to run on an android device. The model was trained on a dataset which was obtained by firing JESS rule engine over a set of inputs. This dataset had total 28,672 data points. It was split into train and test set with ratio 80:20. The rows in dataset were randomly shuffled to avoid overfitting. We also performed one-hot encoding to split the parameter 'API level'. The range of inputs for API level was <16 and [16, 28]. Since all the other parameters took binary values only, it was important to bring this parameter in same range. This ensured better training of the neural network. Hence we performed on-hot encoding over this parameter. In this technique, we split the values of API Level into a vector of columns. Whichever number the API level received as input, was assigned a value 1 in the resultant vector. All the other vector columns were given value 0. The entire data preprocessing part was done using pandas framework.

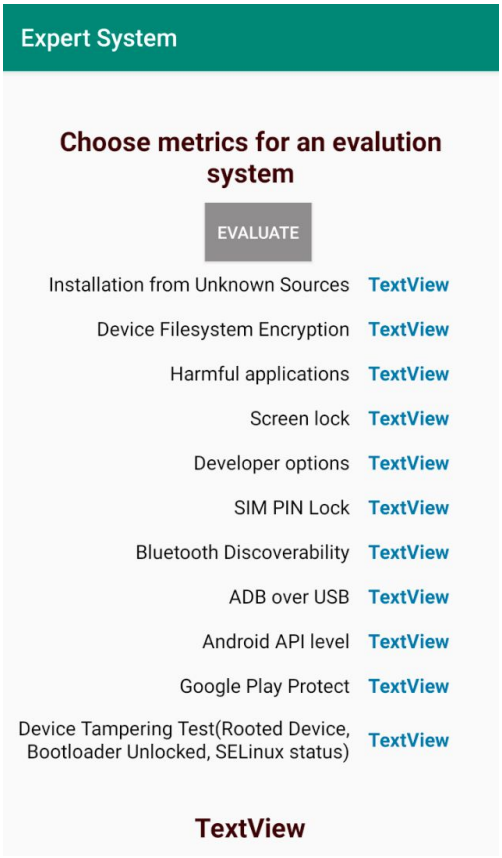
While training, we experimented with hyperparameter tuning of Neural Network. We first started with a concept named, base modeling. It is a concept in which model is trained on data with minimal valued hyperparameters. So, initially we trained the data with 12 nodes and only 1 layer. Then we observed the output. We kept on tuning the hyperparameters in such a way that we ended up adding 3 layers and with 64 nodes in each layer. After that, we have also used a concept called, EarlyStopping. In Keras, it is possible to implement early stopping as a callback function. Callbacks are functions that can be applied at certain stages of the training process, at the end of each epoch. Specifically, in our solution, we included `EarlyStopping(monitor='val_loss', patience=2)` to define that we wanted to monitor the test (validation) loss at each epoch. whenever the test loss does not improve in a range of two epochs, training gets interrupted. However, since we set patience=2, we will get the best model after two epochs. The entire program of generating

The neural network developed using `tflite` was then integrated with the android app. Tflite is a tensorflow based library that is used to deploy neural networks on android smartphones.

4. **Explanation and User's Guide**

We have designed an android application with a basic template which is Empty Template. Initially, we used ConstraintLayout as a based layout. All the widget have resided on this Layout only. After that, we have used several TextView widgets for heading, metrics and to show the evaluation score for an android system. To

systematically show 11 metrics and their corresponding values, we have used TableLayout which comprises of 11 rows with 2 columns. One column for metrics and other for their values (Initial value is TextView which will be replaced by appropriate values before finally appearing on the User Interface).



All the Graphical User Interface (GUI) as well as Logic is developed in Java.

5. Experiments

We performed a set of experiments with neural networks for this project. Different combinations of number of layers, nodes and activation function were tried in the process. We realised that the model overfitted with more than 1 layer. Therefore, we majorly performed some hacks with just one hidden layer.

Additionally, the android app developed was tested on different mobile devices. It helped us to check the compatibility of this app. We ensured that the mobile devices had different parameters in order to perform tests. The output values obtained were coherent to the values obtained in part 2 of this project.

6. Results

As mentioned earlier, we performed some experiments with the neural networks developed. Their results are as follows,

Number of hidden layers	Activation Function	Total number of nodes	Mean absolute Error
1	Relu	12	0.01
1	Relu	8	0.01
1	softmax	8	0
1	softmax	12	0
1	tanh	8	0
1	tanh	12	0

As evident from the table above, the test loss in our experiments is close to or equal to zero in most cases. This was mostly due to less availability of data. A good neural network has balance between number of input parameters and the number of datapoints in the dataset. We had 23 input parameters and just 28672 data points in total. This clearly is not a good balance for developing a robust neural network.