Configuration Manual

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:	1167	**Page Count: 9**	

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):	

Configuration Manual

Sangameshwaran P.L

x18174965

1. Introduction

The configuration manual gives the step by step procedure required to be followed to pre-process the dataset downloaded from the Drebin portal and Appbrain Statistics so that it can be applied by the proposed lightweight 1-dimensional convolution neural network (1D-CNN) and the other machine learning and deep learning models considered in our experiments.

2. Prerequisite

2.1 System Requirements

The minimum requirement required to replicate the proposed work is explained below:

- Operating System (os): Windows 10, Linux, Ubuntu or MAC (preferably windows 10 system).
- Minimum storage: 25 gigabytes (GB).
- Minimum RAM: 8 GB

2.2 Software Requirements

The implementation was done in Anaconda's Jupyter notebook 6.0.3 (Python 3.7.6) where the following packages are installed in a Jupyter note to support machine learning or deep learning, various file operation and visualize the results of the model run on the generated datasets:

- Pandas
- Numpy
- Keras
- Matplotlib
- Math
- Sklearn.metrics
- Scikitplot
- Os
- Glob
- Scipy
- Kutools for excel
- shutil

3. Data processing

3.1: Feature extraction and merging

- The zip file downloaded from the Drebin dataset portal had a file named feature_vector when extracted and the file contained all the malware and benign application stored in it in .file format. The file name was their hash values name.
- To separate the malware from benign files so we can create the final balance dataset with 5560 malware and 5560 benign application we downloaded another file from the portal that had the malware hash values in .csv format file with their family.
- With the help of python, we filter the malware files and benign files in the feature_vector file and store them in 2 separate files.

```
import os
path="C:/Users/plsan/Downloads/drebin/feature_vectors"
list2=['File name comes here in .txt format']
list1=os.listdir(path)
os.chdir(path)
for i in list1:
    for j in list2:
        if i == j:
            count = 0
            count += 1
```

Figure 1. The separating of malware files

```
for file_name in list4:
    original : 'C:/Users/plsan/Downloads/drebin/feature_vectors/'+file_name
    target : 'C:/Users/plsan/Desktop/target/'+file_name
    shutil.copyfile(original, target)
```

Figure 2. Copying files to a different location

- We copy 5560 files from the benign samples renamed with hash value and '_bw' and combine it with malware 5560 samples in a new folder and convert all the files to .txt format using the python.

```
import os
def rename(path):
    list5=os.listdir(path)
    os.chdir(path)
    for i in list5:
        filename, file_extension = os.path.splitext(i)
        os.rename(filename+'_bw'+file_extension)
path='C:/Users/plsan/Desktop/trial/'
rename(path)
```

Figure 3. Renaming of the benign files

- After the files are converted the contents in the file are in key-value pair separated by '::' symbol and all the contents 11,120 malware and benign files contents are copied to a single .csv file.
- With the help of CSV inbuilt functions, we remove the duplicates and separate the key and value with the help of data separator to extract the manifest properties and API calls.
- The manifest properties (permissions, real_permission, intent, features) and Api_calls value are generated with filter option and stored in 2 separate files.

3.2: Dataset conversion to feature matrix

- The filenames of malicious and benign applications are got using the kutools[1] downloaded for excel with their "import and export" option for the filename list.
- The filenames are set as rows and extracted values above are set as a column by transposing it and 2 new columns application category and target is added for mocking and labeling purpose.
- If the column name is present in the particular sample a 1 is inserted to a particular column or a 0 is inserted and if the name of the file is with a '_bw ' a 1 is inserted to the target column denoting it has benign or else a 0 is inserted denoting it as a malware. The 49 application category is given integer value from 1 to 49 and is randomly inserted in the application category column with the rand function of python to mock the dataset.

```
count = 0
for j in df['Samples']:
    for i in data:
        with open('C:/Users/plsan/Desktop/work/'+j) as file:
            contents = file.read()
            if i in contents:
                df[''+i+''][count] = 1
    count += 1
        df.to_csv('C:/Users/plsan/Desktop/api_calls1.csv')
```

Figure 4. Generating of the feature matrix

```
import pandas as pd
import numpy as np
df1 = pd.read_csv("C:/Users/plsan/Desktop/manifest_proproperties2.csv")
df1['Application_Category'] = np.random.randint(1,49,size=(len(df1),1))
print(df1)
```

Figure 5. Rand function implementation

- Finally, the manifest properties and API calls dataset converted to feature matrix is combined to give the third dataset which contains the application category and target column too.

4. Model implementation

The steps to implement the machine learning model (SVM, RF) and deep learning (1D-CNN, MLP) are discussed in detail in this section. Before applying the model we need to read the file that has the desired input for the model or dataset generated, (the steps for one dataset are shown in detail below and the same as to be applied for the other two datasets). After which we need to split the dataset for the test and training. In these experiments, we split the data for test and train as 70% and 30 % for machine learning and 90% and 10 % for deep learning.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3,random_state = 120)
```

Figure 6. Machine learning split

```
x = np.array(x[:])
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.1,random_state = 120)

x_train = x_train.reshape(x_train.shape[0],x_train.shape[1], 1)
x_test = x_test.reshape(x_test.shape[0],x_test.shape[1], 1)
```

Figure 7. Deep learning split

[1] https://www.extendoffice.com/download/kutools-for-excel.html

```
import pandas as pd
measurements = pd.read_csv('C:/Users/plsan/Desktop/api_calls1.csv')

import numpy
import matplotlib.pyplot as plt
import pandas
import math
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from tensorflow.keras.callbacks import EarlyStopping
from keras.layers import Dropout
from keras.layers import *
```

```
x = measurements.iloc[:,1:-1]
y = measurements.iloc[:,-1]
```

Figure 8. Reading the file to be inputted to model

4.1 Support Vector Machine (SVM)

The Code for the SVM model to be applied to the read file and split dataset as above is given below. The code does the training of the model and does the prediction with the test dataset (30%) with which can give the performance evaluation metrics for the model.

```
#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)
```

```
y_pred = clf.predict(x_test)
```

Figure 9. SVM model code for the training and testing

4.2 Random Forest (RF)

The code of the RF model with various hyper tuning parameters is given below with training and prediction done for the evaluation of the model according to the considered metrics.

```
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)
```

Figure 10. RF model code for the training and testing

4.3 Multilayer Perceptron (MLP)

The MLP is ran for 20 epochs with train and test datasets split as 90% and 10% respectively. The model is subjected to a few hyperparameter tuning as shown below.

```
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier(hidden_layer_sizes=(150,100,50), max_iter=20,activation = 'relu',solver='adam',random_state=1)

classifier.fit(x_train, y_train)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571: ConvergenceWarning: Stochas
tic Optimizer: Maximum iterations (20) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

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)

y_pred = classifier.predict(x_test)
```

Figure 11. MLP model code for the training and testing

4.4 1-dimensional Convolutional Neural Network (1D-CNN)

Finally, the 1D-CNN model is run for train and test dataset for 20 epochs. Hyperparameter tuning is applied to the model and is evaluated by plotting confusion matrix, accuracy plot and precision and recall curve for better visualization of the model performance as discussed in the next section in detail.

```
batch_size = 256
num_classes = 2
epochs = 20
input_shape=(x_train.shape[1], 1)
import keras
from keras.models import Model
from keras.models import Sequential
from keras.layers import Convolution1D, ZeroPadding1D, MaxPooling1D, BatchNormalization, Activation, Dropout, Flatten, Dense

from keras.layers.convolutional import Conv1D


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)
y_pred = model.predict(x_test, batch_size=10,
                       verbose=1)
```

Figure 12. CNN model code for the training and testing

5. Metrics Calculation

The F1-Score, Accuracy, Precision and Recall is calculated for all the models discussed above and we have plotted the confusion matrix, test vs train accuracy plot and precision-recall graph for the 1D-CNN model as discussed below.

The metrics are calculated with the help of the “sklearn. metrics” package as follow for SVM, RF and MLP and 1-D CNN (precision, recall and F1-Score) where the accuracy alone is calculated using keras’s packages evaluate the function for 1D-CNN. The metrics snapshot of one dataset (Api calls) with its output is :

1. Accuracy:

```
from sklearn import metrics

# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8690047961630696
```

Figure 13. Accuracy with sklearn

```
score = model.evaluate(x_test, y_test,verbose=1)

print(score)
```

```
112/112 [==============================] - 1s 8ms/step
35/35 [==============================] - 1s 15ms/step - loss: 0.2693 - accuracy: 0.8921
[0.2693162858486175, 0.8920863270759583]
```

Figure 14. Accuracy with Evalute

2. F1-Score

```
from sklearn.metrics import f1_score
f1_score(y_test, y_pred, average='binary')
```

```
0.8765885343123412
```

Figure 15. F1-Score with sklearn

3. Precision and Recall

```
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Precision: 0.8421052631578947
Recall: 0.9140164899882215
```

Figure 16. Precision and Recall with sklearn

The confusion marix, test and train accuracy plot and precision and recall was plotted for 1-D CNN as follow for the api_calls dataset and can be replicated for the rest two dataset.

```
prediction_labels=[]
labels = [0, 1]

for p in y_pred:
    prediction_labels.append(labels[np.argmax(p)])
sum(y_test==prediction_labels)/len(prediction_labels)

import scikitplot as skplt
skplt.metrics.plot_confusion_matrix(y_test, prediction_labels,
                    title='Confusion matrix', figsize = (10,5), text_fontsize='medium', cmap='Blues')
```

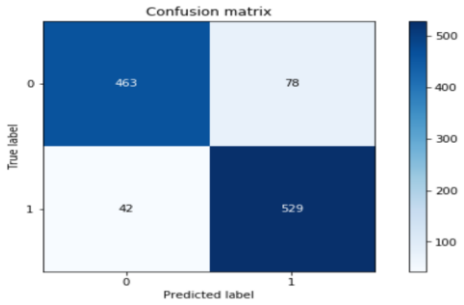

Figure 17. Confusion matrix code with output

```
from matplotlib import pyplot
# plot history accuracy
plt.title('Accuracy')
pyplot.plot(history.history['accuracy'], label='train')
pyplot.plot(history.history['val_accuracy'], label='validation')
pyplot.legend()
pyplot.show()
```

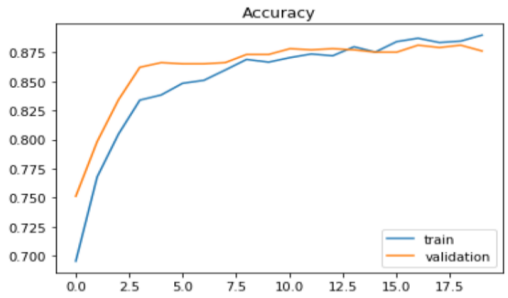

Figure 18. Test vs Train accuracy code with output

```
# plot the precision-recall curves
no_skill = len(arr[arr==1]) / len(arr)
pyplot.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')
pyplot.plot(lr_recall, lr_precision, marker='.', label='Logistic')
# axis labels
pyplot.xlabel('Recall')
pyplot.ylabel('Precision')
# show the legend
pyplot.legend()
# show the plot
pyplot.show()
```

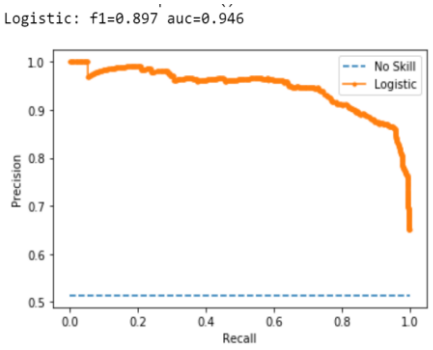

Figure 19. Precision vs Recall curve code with output

After implementing all the experiments we convert the trained 1D-CNN model to tensorflow lite using the tensorflow lite to get a file of 1.27 megabytes(MB) size suitable for deployment in a mobile or IoT device.

6. Reference

[1]R. Ahmed Sayyad, "How to Use Convolutional Neural Networks for Time Series Classification", *Medium*, 2020. [Online]. Available: https://medium.com/analytics-vidhya/how-to-use-convolutional-neural-networks-for-time-series-classification-80575131a474. [Accessed: 13- Aug- 2020].

[2]A. Navlani, "(Tutorial) Support Vector Machines (SVM) in Scikit-learn", *DataCamp Community*, 2020. [Online]. Available: https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python. [Accessed: 13- Aug- 2020].

[3]A. Navlani, "Random Forests Classifiers in Python", *DataCamp Community*, 2020. [Online]. Available: https://www.datacamp.com/community/tutorials/random-forests-classifier-python. [Accessed: 13- Aug- 2020].

[4]A. Nair, A. Nair and A. Nair, "A Beginner’s Guide To Scikit-Learn’s MLPClassifier", *Analytics India Magazine*, 2020. [Online]. Available: https://analyticsindiamag.com/a-beginners-guide-to-scikit-learns-mlpclassifier/. [Accessed: 13- Aug- 2020].

[5]"TensorFlow Lite guide", *TensorFlow*, 2020. [Online]. Available: https://www.tensorflow.org/lite/guide. [Accessed: 13- Aug- 2020].