Malware Detection Using Conventional Neural Network and Regression on smartwatches

Configuration Manual

Research Project
M.Sc. Cybersecurity

Raakesh babu Venkateswara
Student ID: X21105227

School of Computing
National College of Ireland

Supervisor: Mr. Niall Heffernan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Raakesh babu Venkateswara		
Student ID:	21105227		
Programme:	M. Sc. cybersecurity	**Year:**	2021-2022
Module:	Research project configuration manual		
Lecturer:	Mr. Niall Heffernan		
Submission Due Date:	15/08/2022		
Project Title:	Malware detection using Conventional Neural Network and Regression on smartwatches		
Word Count:	1103	**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 bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Raakesh babu venkateswara

Date: 15/08/2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

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

Configuration Manual

Raakesh babu Venkateswara
X21105227

1 Introduction

In this proposal, "Malware Detection Using Conventional Neural Network and Regression on Smartwatches" using CNN-R is implemented using various applications and hardware requirements. The requirements to do the implementation are given below. The main goal of this research is to detect malware using a conventional neural network with regression (CNN-R), a deep learning method. From wireless sensors, the network traffic data under usual conditions and when the attack is recorded in the WSN-DS dataset . This dataset is fed in to one-dimensional CNN-R to classify the normal and abnormal or anomaly data traffic. In this implementation, the data preprocessing, training, testing, and evaluation matrix are implemented.

2 System Requirements

For the smooth processing of the model and to reduce the processing time, the following hardware and software are required:

2.1 Hardware Requirement's

The implementation was performed on an MSI laptop, the configuration of the device is as follows

1. Processor – Intel(R) Core i5-8750H CPU @ 2.20GHz
2. RAM - 32.0 GB DDR4
3. Hard Disk – 1 TB PM981 NVMe SSD
4. OS – Windows 10 Pro 64 – bit

2.2 Software Requirements

The software requirements are listed below.

Software	Version
Python	3.8.3
Anaconda Navigator	1.9.12
Jupyter Notebook	6.0.3
Tensorflow	2.4.0
Numpy	1.18.5
OpenCV	4.4.0
Scikit-learn	0.23.1
IBM SPSS	27.0.0

Table 1: software requirements

3 Data Preprocessing and Splitting

This part represents all steps required for preparing data for the machine learning model.

3.1 Importing Libraries

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv1D, Dropout, BatchNormalization, MaxPooling1D, LeakyReLU
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from keras import backend as K
from sklearn.metrics import confusion_matrix, classification_report
from keras.models import load_model
```

In order to begin the process of data preparation and visualization, the first step is to load all of the essential libraries. These libraries include pandas, Matplotlib, seaborn, and NumPy. Sklearn offers a variety of programs for dividing and converting data. Modeling and layering using Keras for use in model training.

3.2 Data Loading

```
data = pd.read_csv('WSN-DS.csv')
data.columns
```

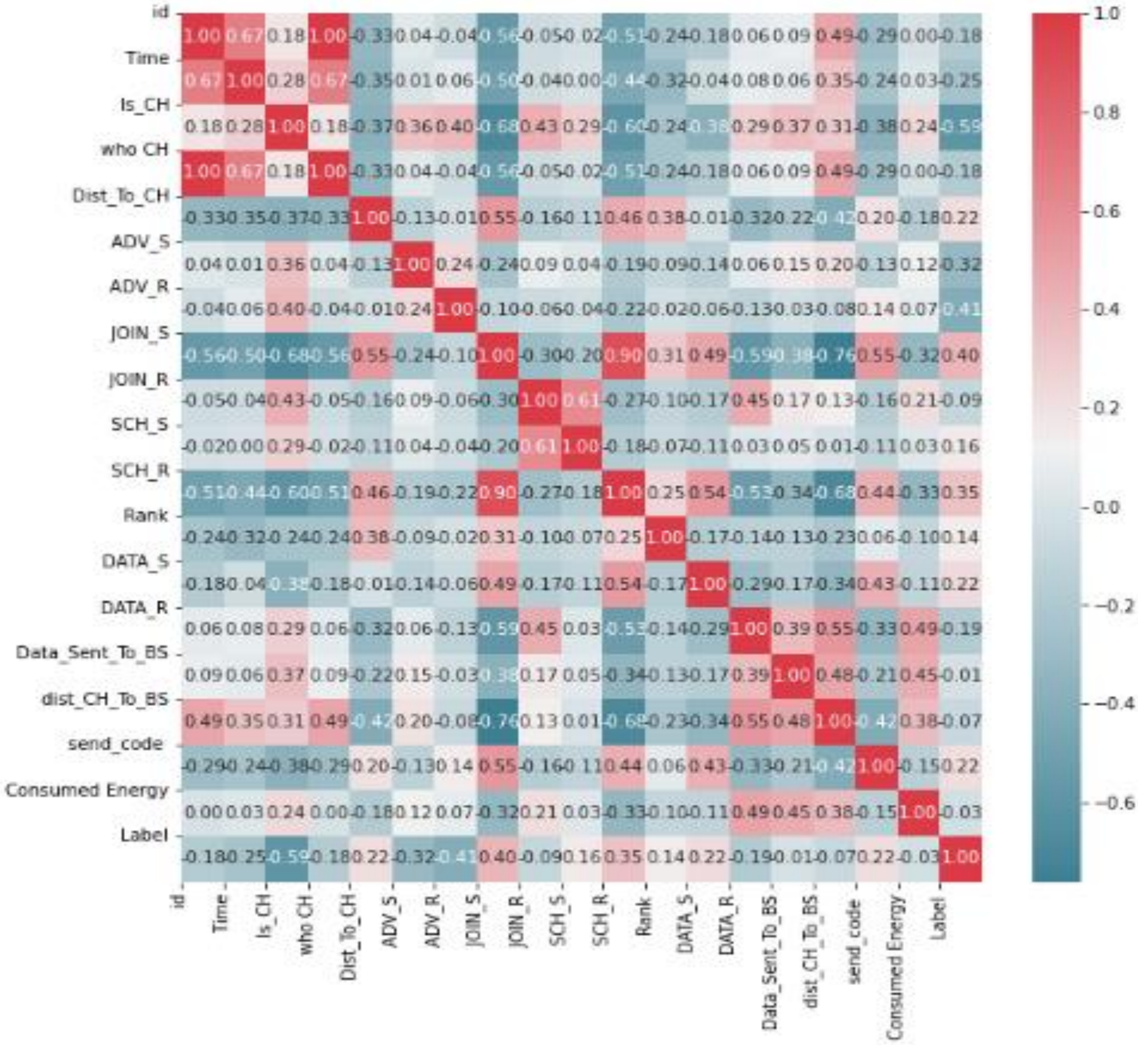
```
Index([' id', ' Time', ' Is_CH', ' who CH', ' Dist_To_CH', ' ADV_S', ' ADV_R',
       ' JOIN_S', ' JOIN_R', ' SCH_S', ' SCH_R', 'Rank', ' DATA_S', ' DATA_R',
       ' Data_Sent_To_BS', ' dist_CH_To_BS', ' send_code ', 'Consumed Energy',
       'Label'],
      dtype='object')
```

```
#all columns present in data
data.head()
```

	id	Time	Is_CH	who CH	Dist_To_CH	ADV_S	ADV_R	JOIN_S	JOIN_R	SCH_S	SCH_R	Rank	DATA_S	DATA_R	Data_Sent_To_BS	dist_CH_To_BS
0	101000	50	1	101000	0.00000	1	0	0	25	1	0	0	0	1200	48	130.08535
1	101001	50	0	101044	75.32345	0	4	1	0	0	1	2	38	0	0	0.00000
2	101002	50	0	101010	46.95453	0	4	1	0	0	1	19	41	0	0	0.00000
3	101003	50	0	101044	64.85231	0	4	1	0	0	1	16	38	0	0	0.00000
4	101004	50	0	101010	4.83341	0	4	1	0	0	1	25	41	0	0	0.00000

The 'WSN-DS' dataset has to be loaded into a pandas dataframe before the data.head() function can be used to show all of the columns.

3.3 Data Visualization



The correlation between each column of the dataset is shown in the graphic that can be seen above.Since there are no null or garbage values in the data, we can go ahead and split it up and get it ready for the CNN model.

3.4 Data Splitting

```
x_train, x_test, y_train, y_test = train_test_split(np.asarray(x), np.asarray(y), test_size = 0.3, random_state = 42)
```

The data is divided into training and testing sets using the SKELEARN train test split technique with a test size of 30% and a random state of 42.

3.5 One Hot Encoding

```
# Convert class vectors to binary class matrices. This uses 1 hot encoding.
y_train_binary = keras.utils.to_categorical(y_train, num_classes)
y_test_binary = keras.utils.to_categorical(y_test, num_classes)
```

```
y_train_binary
```

```
array([[0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0.],
       ...,
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0.]], dtype=float32)
```

Since there are no numbers in the target column, we will have to use one keras hot encoding to turn it into a categorical format.

3.6 Convert data into 3D arrays

```
x_train = np.repeat(x_train[:, :, None], repeats=3, axis=2)
x_test = np.repeat(x_test[:, :, None], repeats=3, axis=2)
```

Convolutional layers need data in a 3D format, while the first one just needs data in a 2D format. In this particular scenario, the data dimensions are altered with the help of numpy.

4 Model

Now the data is ready for training, we need to build a model that fits the data accurately and makes good results.

4.1 Defining Model

```
model = keras.models.Sequential()
model.add(Conv1D(filters=6, kernel_size=21, strides=1, padding='same', activation='relu',
                 input_shape= (x_train.shape[1],3),kernel_initializer=keras.initializers.he_normal()))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=2, strides=2, padding='same'))
model.add(Conv1D(filters=16, kernel_size=5, strides=1, padding='same',activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling1D(pool_size=2, strides=2, padding='same'))
model.add(Flatten())
model.add(Dense(120, activation='relu'))
model.add(Dense(84))
model.add(Dense(5, activation='softmax'))
model.summary()
```

The CNN model is initiated by the Sequential() function and the input layer. Then the model is created with two layers of conv1D, Maxpooling1D, and batch normalization, one flattened layer, and three densely connected layers as shown above. ReLU is the activation function in the initial layers and the last layer has a softmax function due to multi-class classification.

These layers are defined using filters of 6 and 26 in the conv1D layer with the stride of 1 and input size according to data shape which is 18 in this case. Maxpooling1D is used using a pool size of 2 and 2 strides after each iteration.

4.2 Compiling Model

```
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(),
              metrics=['accuracy'])
```

For compiling model, the categorical_crossentropy loss function is selected due to multi-class classification, Stochastic gradient descent optimizer is used and the accuracy metric is declared as shown in the above code.

4.3 Model Training

```
batch_size = 128
epochs = 10
history = model.fit(x_train, y_train_binary,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test_binary))
```

The model is now ready to fit on data using hyperparameters of batch size 128 and 80 epochs with validation data that is used to validate model performance on unseen data.

4.4 Model Evaluation

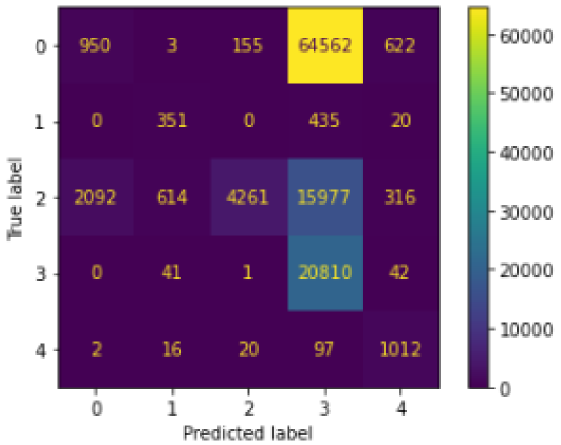
Finally, the model is evaluated using 'accuracy' and 'loss' at each epoch obtained from seen and unseen data which are further shown below.

```
In [270]:

model.evaluate(x_test,y_test_binary)

3513/3513 [==============================] - 8s 2ms/step
6
```

The above picture shows the evaluation of our data test data. Then performance accuracy we are achieving through this model is upto 96 % and testing loss we are achieving from this model is 0.25.

```
ConfusionMatrixDisplay(matrix,display_labels = ['0','1','2','3','4']).plot()
```

```
Out[267]:

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x22b2261e310>
```



After checking the model's performance through testing it is now necessary to display the confusion matrix for all the 5 labels. With help of confusion matrix we would be able to measure the accuracy, precision, recall through the parameters of the confusion matrix.

4.5 Saving model

```
model.save('cnn1dmodel.tflite')
```

Then the model is saved with the'model.save' function with the extension '.tflite', which represents the lightweight model for use further in the future.

5 Conclusion

In this model, implementation is explained in each phase in this configuration manual, and the accuracy is 97.96%. So, the model is very light-weight and the size is very small, so it can be implemented easily in IoT devices to detect the malware and stop them.