National College of Ireland

# Configuration Manual

MSc Research Project
MSC. Cyber Security

Saifullah Sheikh
Student ID: X19216815

School of Computing
National College of Ireland

Supervisor: Imran Khan

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

| | |
|---|---|
| **Student Name:** | Saifullah Sheikh |
| **Student ID:** | X19216815 |
| **Programme:** | MSc. In Cyber Security **Year:** 2020-2021 |
| **Module:** | Internship |
| **Lecturer:** | Imran Khan |
| **Submission Due Date:** | 16-08-2021 |
| **Project Title:** | **Improve the detection accuracy and performance of intrusion detection system using deep Bi-Directional LSTM.** |
| **Word Count:** | 1920 **Page Count:** 8 Pages |

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:** ……Saifullah Sheikh……

**Date:** ……16-08-2021……

**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

Saifullah Sheikh
Student ID: X19216815

## 1. System Requirements

To process & execute the implementation model smoothly & minimize the time consumption while running the program, we need to follow some essential software and hardware requirements.

**1.1 Software: -**

There is various software that I used for implementation and execution of project such as,

- Google Colaboratory (google cloud based environment)
- Py charm
- Microsoft excel
- Anaconda Navigator
- Jupyter Notebook
- Python

1.2 **Hardware:** -

The project implementation has done in Asus TUF Gaming laptop & the specification of machine is,

- Ram – 12 GB DDR4
- Hard Disk – 1 TB
- SSD – 512 GB
- GPU – NVIDIA Geforce GTX
- O.S. – Windows 10 Enterprise 64bit.
- Processor - AMD Ryzen 7 3750H with Radeon Vega Mobile Gfx 2.30 GHz

In this research project we used publicly available dataset of UNSW-NB15 dataset was created by the IXIA PerfectStorm, it shown below in fig.1

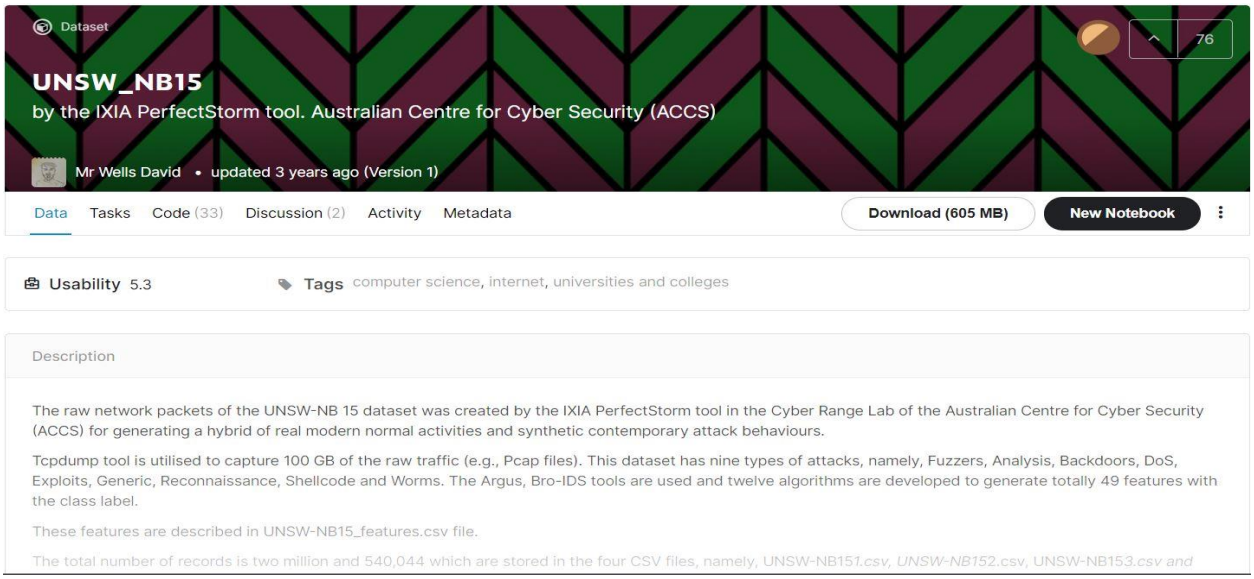


Fig.1 UNSW-NB15 Dataset

- Here, we used Google colaboratory that is cloud based platform to perform all implementation process, this steps helped to setup a Google colaboratory environment. as we can see in Fig.2

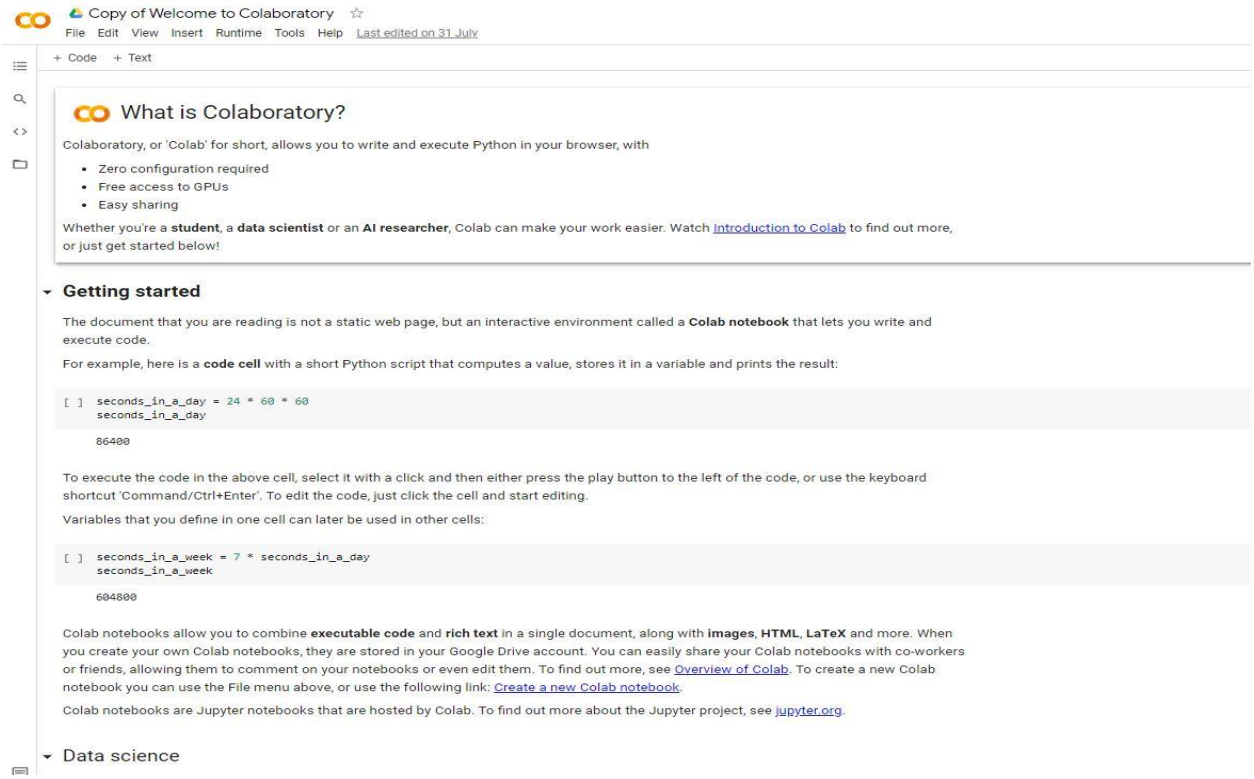


Fig.2 Google Colaboratory home page

After downloading the UNSW-NB15 dataset I uploaded it to google drive then directly mounted it on google colaboratory notebook. Shown in figures 3.

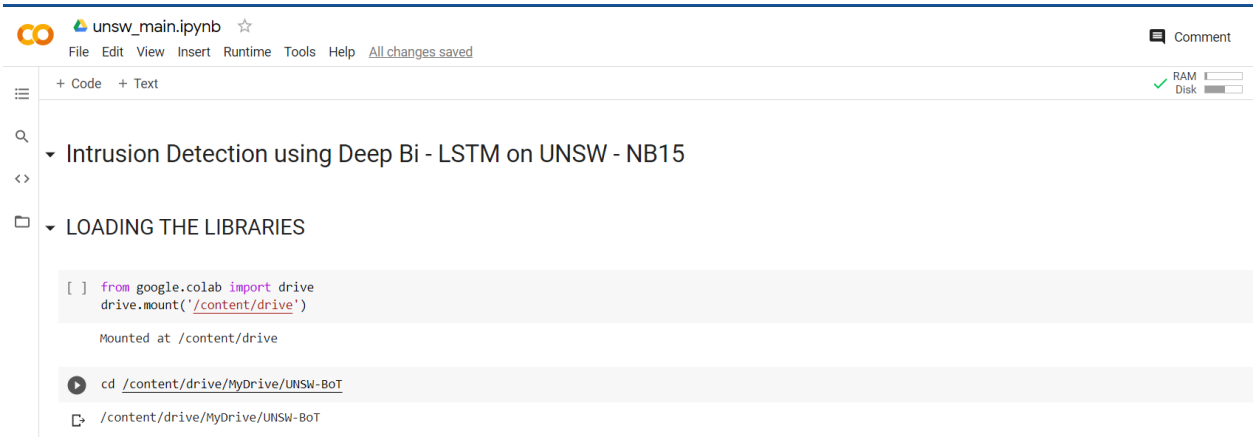


Fig. 3 one drive data imported into Google colaboratory.

# **Intrusion Detection using Deep Bi - LSTM on UNSW – NB15**

## 1. Loading Libraries: -

**1.1)** Here, as we can see below in fig. 4. I imported pandas and numpy that is predefined package library for the use of pre-processing of data & MinMaxScaler is imported to scale the data for deep learning. seaborn (sns) is used for plotting and data visualization and it built on the top of matplotlib to plot the data.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
import matplotlib.pyplot as plt
```

Fig. 4. Loading Library

**1.2)** Here, we imported model from keras library and this are the thing for neural network as all this are the major packages for deep learning which we imported from keras library.

```
from keras.models import Model
from keras.layers import Dense, LSTM, Bidirectional, Dropout, Input
from keras.preprocessing.sequence import TimeseriesGenerator
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from keras.optimizers import Adam
```

Fig.5. Keras Library Imported

**1.3)** Now here, we loaded our dataset for training and testing two data should be available so for the total dataset will go through on training and remaining will pass through to testing. The dataset is used as input the pd (pandas) library and function “pd.read_csv” for both train and test. After loading the dataset, we executed it and got message ‘Data Loaded Successfully’.

```
[ ]  print("\n\tDATA LOADED SUCESSFULLY\n\t*****************************\n")
     train_set = 'Training.csv'
     test_set = 'Testing.csv'
     train = pd.read_csv(train_set, index_col='id')
     test = pd.read_csv(test_set, index_col='id')
```

Fig.6. Data Loading

**1.4)** Now here, we used fig to give a figure size in count plot, and in next line we used seaborn (sns) themes ‘darkgrid’ for better visualization as there are various seaborn themes such as whitegrid, dark, white, tick and darkgrid so here we chose a random theme darkgrid and we set a title attack types for plotting. We plotted the data to know the attack types & we passed the train data, this plot shown like normal, backdoor, analysis. It shown the number of occurrences for each attack category & it mainly show how many attack which are repeatedly occurs on total datasets like we have plotted, Figure 7.

```
[ ]  # Count Plot for Sequences
     print("\n\tPLOTTED COUNT PLOT\n\t*****************************\n")
     fig = plt.figure(figsize=(16,8))
     sns.set_theme(style="darkgrid")
     ax = sns.countplot(x="attack_cat", data=train)
     ax.set_title("Attack Types")
     plt.show()
```

Fig.7. Attack types & count plotting

## 2. Data Pre-Processing: -

**2.1.** Here, in pre-processing step we used train.isna().sum() to return the number of missing values in each column & count total missing values in data frames. In order to pass the data, zero (0) means these features is having null missing values. After checking that there are no missing values then we move on to features selection by using correlation.

PRE PROCESSING

```
[ ]  print("\n\tCHECKING MISSING VALUES\n\t*****************************\n")
     print(train.isna().sum())
```

```
        CHECKING MISSING VALUES
        *******************************

dur                  0
proto                0
service              0
state                0
spkts                0
dpkts                0
sbytes               0
dbytes               0
rate                 0
sttl                 0
dttl                 0
sload                0
dload                0
sloss                0
dloss                0
sinpkt               0
dinpkt               0
sjit                 0
djit                 0
swin                 0
stcpb                0
dtcpb                0
dwin                 0
tcprtt               0
synack               0
ackdat               0
smean                0
dmean                0
trans_depth          0
response_body_len    0
ct_srv_src           0
ct_state_ttl         0
ct_dst_ltm           0
ct_src_dport_ltm     0
ct_dst_sport_ltm     0
ct_dst_src_ltm       0
is_ftp_login         0
ct_ftp_cmd           0
ct_flw_http_mthd     0
ct_src_ltm           0
ct_srv_dst           0
is_sm_ips_ports      0
attack_cat           0
label                0
dtype: int64
```

Fig.8.Data pre-processing

# 3. Feature Selection: -

**3.1.** It will show based upon the label each and every features are correlated so likewise it will remove the less correlated values like we can see the values in (-). and greater than 0 means highly correlated values and we have to remove the negative correlated values and only pass the positive correlated values. we dropped all this features ‘dload’ , ‘swin’ , ‘attack_cat’ from train & test because all these contains unwanted data. Then train and test label will be saved on training_label & testing_label & then we copied trainig_label into temp_train and testing_label into temp_test.

FEATURE SELECTION

```
[ ]  print("\n\tBEST FEATURES USING CORRELATION\n\t*****************************\n")
     print(train.corr()["label"])

     train.drop(["dload","swin","attack_cat"],axis = 1,inplace = True)
     test.drop(["dload","swin","attack_cat"],axis = 1,inplace = True)

     training_label = train['label'].values
     testing_label = test['label'].values
     temp_train = training_label
     temp_test = testing_label
```

```
        BEST FEATURES USING CORRELATION
        *******************************
```

Fig. 9. feature selection

**3.2.** here, we created a new dummy column by using python function get_dummies() for data manipulation & it basically converts all categorical string data in the form of dummy or numerical indicators. Here we converted the categorically data such as 'proto' (protocol), 'service', 'state' into the numerical indicators values. If we type 'train' or 'test' in new line and run it then we will see protocol, services and states. we convert the categorical data into numerical order to pass the numeric data into the neural network

```
# Creates new dummy columns
unsw = pd.concat([train, test])
unsw = pd.get_dummies(data=unsw, columns=['proto', 'service', 'state'])
```

Fig. 10 String conversion to numeric

**3.3.** In the unsw value it contains all the labels like normal and abnormals, here we have taken a label means whether its normal and abnormal if these features comes it means the label will be normal. I dropped the label here (unsw.drop). now we are taking all the values in arrays format, as unsw.values will be copied to unsw_label so in the next step of minmax scaling we will scale the unsw_value.

```
[ ]  # Normalising all numerical features:
     unsw.drop(['label'], axis=1, inplace=True)
     unsw_value = unsw.values
```

Fig. 11 Normalizing Numeric features

# 4. MIN MAX SCALING: -

**4.1.** It scales based upon the minimum and maximum values so basically it goes through the dataset and then it will check the lowest minimum values and highest maximum values so based upon that it will divide all the dataset into maximum values and lowest values so likewise it divides all the things accordingly, why we are doing this? This will convert all the features into the float values like 0.001, 0.1 will be changed. For the reason is 1 consists of more memory as compared to 0.001 any float values, so if you pass 0.001 the training time will be reduced as compared to passing 1 so that's why we carried the total data. We have scaled total dataset that is in train_set & test_set .There are lots of float values in our dataset that's why we used min max scaling and it gives exact result for our scaling process, and if the dataset consists 1 to 100 means we can use standard scaler but here our dataset consists float values that's the main reason we are using minmax scaling rather than standardscaler.

▾ MIN MAX SCALING

```
[ ]  # MinMax Scaling
     print("\n\tSCALING COMPLETED SUCESSFULLY\n\t*****************************\n")
     scaler = MinMaxScaler(feature_range=(0, 1))
     unsw_value = scaler.fit_transform(unsw_value)
     train_set = unsw_value[:len(train), :]
     test_set = unsw_value[len(train):, :]
```

```
SCALING COMPLETED SUCESSFULLY
*****************************
```

Fig. 11. MinMax Scaling

# 5. DEEP BI-DIRECTIONAL LSTM: -

**5.1.** Bidirectional (this is a package, so we have to imported it), LSTM (this is a lstm architecture, this we have to declared between the bidirectional). Units=24 (this are the number of unit s means which passed into a feature. Units means number of neurons in it. So here 24 neurons can pass in a single time so like this which we have declared it). Activation function is 'tanh'. We used 'tanh' for implementing my process because as compared to other activation function it is better. Basically, it is very common to the sigmoid activation function. The 'tanh' function range is from (-1 to 1). return_sequence=True = At first iteration it will learn some feature, so feature means if this value is coming so it will be normal.In order to prevent that over fitting we reduced the dropout layer; it will take the features and after that will check the result again whether the result improved or not. likewise, it will check every layer. now it will check 0.1, 0.2, 0.3, 0.4 & 0.5 is enough and more than 0.5 means it will check the iteration like it sends more data and there will be too much loss while running the code so for that reason we must reduce the dropout like 0.1 to 0.5.To learn the features using TANH as it delivers better training performance for multilayer neural networks for the Bi-LSTM Layers and then that layers will be passes as to the Dense layer which consists of RELU and at last for the classification we have used SIGMOID Function.here we also used optimizer which is a adam as adam is the most popular optimizer which is used for classification and here (learning rate) is 'lr=0.001' so it will learn in a keen manner and learn accurately for 0.001.

DEEP BI DIRECTIONAL LSTM

```
[ ] print("\n\tMODEL CREATED SUCESSFULLY\n\t*****************************\n")
    # Deep Bi Directional LSTM Model
    input_traffic = Input(shape=(time_steps, 194))

    lstm1 = Bidirectional(LSTM(units=24, activation='tanh',
                               return_sequences=True, recurrent_dropout=0.1))(input_traffic)
    lstm_drop1 = Dropout(0.5)(lstm1)
    lstm2 = Bidirectional(LSTM(units=12, activation='tanh', return_sequences=False,
                               recurrent_dropout=0.1))(lstm_drop1)
    lstm_drop2 = Dropout(0.5)(lstm2)
    mlp = Dense(units=6, activation='relu')(lstm_drop2)
    mlp2 = Dense(units=1, activation='sigmoid')(mlp)
    classifier = Model(input_traffic, mlp2)
    optimize = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-8)
```

Fig.12 Deep Bi-Directional LSTM

**5.2.** After the model declaration we have to Compile it. So here classifier is the model name. here, we used binary_crossentropy . Binary means at last we have to mention 0 or 1, likewise our output will be this attack or this attack) as there are many crossentropy such as categorical or binary.

```
print("\n\tMODEL COMPILED SUCESSFULLY\n\t*****************************\n")
classifier.compile(optimizer=optimize, loss='binary_crossentropy', metrics=['accuracy'])
```

Fig.13 Compilation process.

# 6. Model Fitting: -

**6.1.** File path= "model.hdf5" (This is the model name)
Model Checkpoint – whether to only keep the model that has achieved the "best performance" so far, or whether to save the model at the end of every epoch regardless of performance. (This is the call backs).

Monitor =Val_accuracy (It will check the validation accuracy each and every iteration, the accuracy does not improve Gradually means it will stop the iteration). Then, we just fitting the model like Classifier Is the model name

So in each and every iteration what's The accuracy and validation accuracy we got. So at the end we got 93.3 accuracy it means there is a no improvement on it. So validation accuracy did not improve from 0.81377 & It will automatically stop the overall iteration.

As you can see the epoch at 50, We assigned epoch 50 so it'll check 50 iterations.

MODEL FITTING

```
[ ] filepath="model.hdf5"
    checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')

    reduc_lr = ReduceLROnPlateau(monitor='val_accuracy', patience=10, mode='max', factor=0.2, min_delta=0.0001)


    history = classifier.fit_generator(train_generator, epochs=50, verbose=2, steps_per_epoch=168,
                                       callbacks = [checkpoint],
                                        validation_data=test_generator, shuffle=0, validation_steps=80)
```

Fig.14. Model Fitting

# 7. Performance Metrics: -

**7.1.** So this is the performance metrix our predicted variable will be stored in label_original (in model prediction) & our predicted variable is y_pred where we saved the things. The import function of testing, the accuracy score, confusion matrix, classification report were obtained from the library of 'sklearn.metrics. In the final step of the testing stage, i obtained a confusion metrix that was completed with the help of 'sklearn.metrics' library using function ' confusion_matrix' .

PERFORMANCE METRICS

```
[ ]  from sklearn.metrics import confusion_matrix, classification_report,accuracy_score

     print("\n\tACCURACY SCORE\n\t*****************************\n")
     print(f"\t{accuracy_score(label_original, y_pred)}")

     print("\n\tCONFUSION MATRIX\n\t*****************************\n")
     print(f"\t{confusion_matrix(label_original, y_pred)}")

     print("\n\tCLASSIFICATION REPORT\n\t*****************************\n")
     print(f"\t{classification_report(label_original, y_pred)}")
```

```
        ACCURACY SCORE
        *****************************

        0.9149518694196429

        CONFUSION MATRIX
        *****************************

        [[ 42961  13040]
 [  1591 114440]]

        CLASSIFICATION REPORT
        *****************************

                      precision    recall  f1-score   support

           0       0.96      0.77      0.85     56001
           1       0.90      0.99      0.94    116031

    accuracy                           0.91    172032
   macro avg       0.93      0.88      0.90    172032
weighted avg       0.92      0.91      0.91    172032
```

Fig.15. Result Testing