Configuration Manual

Academic Internship
MSc Cybersecurity

Shalini Srinivasan
Student ID: x21208000

School of Computing
National College of Ireland

Supervisor: Mr Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name**:**	Shalini Srinivasan
Student ID:	x21208000
Programme:	MSc in Cybersecurity — Year**:** 2023
Module**:**	Academic Internship
Supervisor:	Mr Vikas Sahni
Submission Due Date**:**	15th August 2023
Project Title:	Botnet detection using Multi D Convolutional Neural Network
Word Count:	654

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: Shalini Srinivasan

Date: 10/08/2023

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

Shalini Srinivasan
Student ID: x21208000

1 Introduction

The configuration manual presents information about the hardware requirements, the packages that must be installed, and the process necessary for effectively completing this project. Deep learning techniques are used in this research to detect botnets. The primary IDE required is anaconda navigator.

2 System Specification

The main specification of the system is:

- Apple M1 GPU
- CPU with 8 cores; 4 for performance and 4 efficiency cores.
- Storage: 256 GB SSD
- 8Gb memory with LPDDR4X
- Operating System: MacOS
- Python version: 3.11.3

(***Apple Macbook Air (M1, 2020): Full Specs, Tests and User Reviews*, n.d**.)

3 Software tools

The tools used for this research project are:

- Anaconda Navigator v23.5.0
- Python 3.11.4
- PyArrow
- Keras
- Sklearn
- Imageio
- Pandas
- Numpy
- Matplotlib
- Seaborn
- Label encoding
- Cv

3.1 Installation of the tools

The tools were installed using anaconda navigator. The steps below show how the project can be run on your local device.

Step 1: Install anaconda through the link https://docs.anaconda.com/free/anaconda/install/mac-os/.
Step 2: Once anaconda has been set up, Open jupyter notebook.

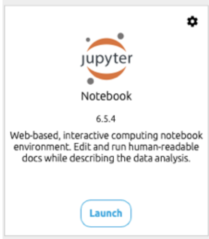

Figure 1: Jupyter Notebook

Step 3: Select and open the folder containing the code. When you open it, the file appears in green.

Figure 2: Folder on Jupyter

Step 4: First, we install python and check the version. The version used for the project was 3.11.4. And then, Install keras through the command line.

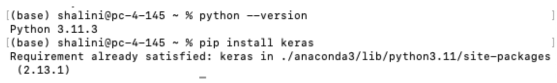

Figure 3: Installing Keras

If the installation does not take place, check for a upgrade, and upgrade pip using the command,

```
pip3 install --upgrade pip
```

(*How to Install Keras on MacOS? - GeeksforGeeks*, n.d.)

Step 5: The data processing notebook is created, and all the packages are imported. As we are using a parquet file to read the dataset faster, Pyarrow must first be installed. Because I've already installed it, it indicates that the requirements has been met.

Figure 4: Installing PyArrow

Step 6: Import the remaining packages for the execution of the code.

```
In [3]: import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import seaborn as sn
import matplotlib.pyplot as pl
import seaborn as sn
import matplotlib.pyplot as py
import cv2
import os
import imageio as im
from PIL import Image
import tensorflow as tf
from os import listdir
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, concatenate
from tensorflow.keras.layers import Conv2D, MaxPooling2D,BatchNormalization
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.models import load_model
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
```

Figure 5: Import Packages

Step 7: As we use a specific method to categorise the botnet traffic, two functions should be executed which categorises the traffic based on the protocol. The detect_traffic() and detect_http_traffic() is used.

```
def detect_traffic(df2):
    detected = []
    for index, row in df2.iterrows():
        # UDP Communication detection
        if row['proto'] == 'udp':
            if 36 <= row['src_bytes'] <= 67:
                detected.append('Suspicious traffic')
            elif row['src_bytes'] > 120 and row['dst_bytes'] > 400:
                detected.append('Malicious traffic')
            else:
                detected.append('Legitimate P2P')
        # TCP Communication detection
        elif row['proto'] == 'tcp':
            # Data Transmission Stage
            if row['src_bytes'] < 1000 and row['dst_bytes'] < 1000:
                detected.append('Malicious traffic')
            else:
                detected.append('Legitimate traffic')

        else:
            detected.append('Unknown')
    df2['Detection'] = detected
    return df2

updated_df = detect_traffic(df2)
display(updated_df)
```

Figure 6: Function Detect Traffic

```
def detect_http_traffic(updated_df):
    detected = []
    for index, row in updated_df.iterrows():
        if 'http' in row['label'].lower() and 'botnet' in row['label'].lower()
        :
            # Additional logic for HTTP traffic with 'botnet' label and
            'http' in the label
            range1 = (0, 500)
            range2 = (501, 1000)
            range3 = (1001, 1500)

            src_bytes = row['src_bytes']
            dest_bytes = row['dst_bytes']

            if range1[0] <= src_bytes <= range1[1] and range1[0] <=
            dest_bytes <= range1[1]:
                detected.append('Malicious traffic - Range 1')
            elif range1[0] <= src_bytes <= range1[1] and (range2[0] <=
            dest_bytes <= range2[1] or range3[0] <= dest_bytes <= range3[1]):
                detected.append('Legitimate traffic - Range 1')
            else:
                detected.append(row['Detection'])  # Preserve the existing
                value
        else:
            detected.append(row['Detection'])  # Preserve the existing value
    updated_df['Detection'] = detected
    return updated_df
```

Figure 7: Function detect http traffic

Step 8: After removing the null values and categorising the data, label encoding is used to ensure that the data is ready for processing.

```
In [28]: label_encoder = LabelEncoder()
         df2['dir'] = label_encoder.fit_transform(df2['dir'])
         df2['proto'] = label_encoder.fit_transform(df2['proto'])
         df2['state'] = label_encoder.fit_transform(df2['state'])
         df2['label'] = label_encoder.fit_transform(df2['label'])
         df2['Detection'] = label_encoder.fit_transform(df2['Detection'])

In [29]: df2

Out[29]:
```

	dur	proto	dir	state	tot_pkts	tot_bytes	src_bytes	label	dst_bytes	Detection
0	1.026539	0	0	381	4	276	156	3	120	1
1	1.009595	0	0	381	4	276	156	3	120	1
2	3.056586	0	0	363	3	182	122	4	60	1
3	3.111769	0	0	363	3	182	122	4	60	1
4	3.083411	0	0	363	3	182	122	4	60	1
...	...	...	...	...	...	...	...	...	...	...
10598766	0.025135	0	0	136	44	38456	1264	3	37192	0
10598767	0.000336	1	3	21	2	231	74	1362	157	0
10598768	0.000325	1	3	21	2	211	74	1362	137	0
10598769	0.000466	1	3	21	2	263	85	1362	178	0
10598770	0.077026	0	0	305	10	3641	431	5	3210	0

10428885 rows × 10 columns

Figure 8: Label Encoding

Step 9: Divide the dataframe evenly to ensure that the data has been correctly categorised for train, test, and further prediction.

```
In [32]: # Step 1: Grouping the DataFrame based on the 'Detection' column
         grouped = df2.groupby('Detection')

         # Step 2: Calculate the desired number of rows for each group to achieve equal split
         desired_rows_per_group = 150000  # Change this value as per your requirement

         # Step 3: Reduce the number of rows for each group to match the desired number of rows
         reduced_df = grouped.apply(lambda x: x.sample(n=min(desired_rows_per_group, len(x)), replace=False))

         # Reset the index of the reduced DataFrame to remove the grouped index
         reduced_df.reset_index(drop=True, inplace=True)

         # Displaying the reduced DataFrame
         display(reduced_df)
```

	dur	proto	dir	state	tot_pkts	tot_bytes	src_bytes	label	dst_bytes	Detection
0	0.175393	1	3	21	2	135	75	7	60	0
1	0.012563	1	3	21	2	180	90	8	90	0
2	0.135676	0	0	136	38	30631	913	5	29718	0
3	0.231470	0	0	136	38	29512	1029	5	28483	0
4	0.000997	1	3	21	2	570	78	7	492	0
...	...	...	...	...	...	...	...	...	...	...
449995	0.240429	1	3	21	3	2834	60	7	2774	2
449996	0.007343	1	3	21	4	3028	60	7	2968	2
449997	0.000366	1	3	21	2	313	64	7	249	2
449998	0.000460	1	3	21	2	207	66	1362	141	2
449999	0.344751	1	3	21	4	2996	60	7	2936	2

450000 rows × 10 columns

Figure 9: Divide dataframe equally

Step 10: Reduce the dataset to create images. A function is run, which generates the folder IMG1 and three sets of test and train photos.

```
In [44]: import imageio as im

         def data_to_img(x,y,type_of_data):
             len_of_rows = x.shape[0]
             for i in range(0,len_of_rows):
                 temp = np.array(x.iloc[i])
                 temp = temp.reshape(-1,9)
                 filename = "IMG1/"+type_of_data+'/'+str(y.iloc[i])+'/img_'+str(i)+'.jpg'
                 os.makedirs(os.path.dirname(filename), exist_ok=True)
                 im.imwrite(filename, temp)
         data_to_img(X_train, y_train, 'Train1')
         data_to_img(X_test, y_test, 'Test1')

Lossy conversion from float64 to uint8. Range [0, 1]. Convert image to uint8 prior to saving to suppress this warnin
g.
Lossy conversion from float64 to uint8. Range [0, 1]. Convert image to uint8 prior to saving to suppress this warnin
g.
Lossy conversion from float64 to uint8. Range [0, 1]. Convert image to uint8 prior to saving to suppress this warnin
g.
Lossy conversion from float64 to uint8. Range [0, 1]. Convert image to uint8 prior to saving to suppress this warnin
g.
Lossy conversion from float64 to uint8. Range [0, 1]. Convert image to uint8 prior to saving to suppress this warnin
g.
Lossy conversion from float64 to uint8. Range [0, 1]. Convert image to uint8 prior to saving to suppress this warnin
g.
Lossy conversion from float64 to uint8. Range [0, 1]. Convert image to uint8 prior to saving to suppress this warnin
g.
Lossy conversion from float64 to uint8. Range [0, 1]. Convert image to uint8 prior to saving to suppress this warnin
g.
Lossy conversion from float64 to uint8. Range [0, 1]. Convert image to uint8 prior to saving to suppress this warnin
g.
Lossy conversion from float64 to uint8. Range [0, 1]. Convert image to uint8 prior to saving to suppress this warnin
```

Figure 10: Image creation

Step 11: Once the images are generated, the training of the model is performed. To train the model, in a separate notebook, import the packages mentioned below.

```
In [1]: from os import listdir
        import numpy as np
        import tensorflow as tf
        import cv2
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, concatenate
        from tensorflow.keras.layers import Conv2D, MaxPooling2D,BatchNormalization
        from sklearn.model_selection import train_test_split
        from tensorflow.keras.utils import to_categorical
        from tensorflow.keras.optimizers import RMSprop
        from tensorflow.keras.models import load_model
        from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
```

Figure 11: Packages in Train Notebook

Step 12: Load images and create two CNN models which can be combined later.

```
In [11]:
model1 = Sequential()

model1.add(Conv2D(32,(3,3), input_shape=(50,50,1), activation='relu'))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(Dropout(0.5))

model1.add(Conv2D(filters=32, kernel_size=(3, 3)))
model1.add(Activation('relu'))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(BatchNormalization())

model1.add(Conv2D(filters=64, kernel_size=(3, 3)))
model1.add(Activation('relu'))
model1.add(MaxPooling2D(pool_size=(2 ,2)))
model1.add(BatchNormalization())
model1.add(Dropout(0.1))

model1.add(Conv2D(filters=128, kernel_size=(3, 3)))
model1.add(Activation('relu'))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(BatchNormalization())


model1.add(Flatten())


model1.compile(loss='categorical_crossentropy', optimizer='adam', metrics = ['accuracy'])

#model1.summary()

In [12]: model2 = Sequential()
model2.add(Conv2D(32,(3,3), input_shape=(50,50,1), activation='relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.5))

model2.add(Conv2D(filters=32, kernel_size=(3, 3)))
model2.add(Activation('relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(BatchNormalization())

model2.add(Conv2D(filters=64, kernel_size=(3, 3)))
model2.add(Activation('relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(BatchNormalization())
model2.add(Dropout(0.1))

model2.add(Flatten())
model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics = ['accuracy'])

#model2.summary()
```

Figure 12: Two CNN models used

```
In [13]: model0 = concatenate([model1.output, model2.output])

x = Dense(512, activation='relu')(model0)
x = Dropout(0.4) (x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.4) (x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.4) (x)
output = Dense(3,activation='softmax')(x)

final_model = tf.keras.Model(inputs=[model1.input, model2.input], outputs=[output])

rm = tf.keras.optimizers.SGD(learning_rate=0.001, momentum=0.8)
rm2 = RMSprop(learning_rate=0.001, rho=0.9)
rm3 = tf.keras.optimizers.Adagrad(learning_rate=0.01)
rm4 = tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=False)
```

Figure 13: Concatenation of the two models

Step 13: Create a model with a specified batch size and epoch value. Draw a graph to check the accuracy after using matplotlib.

```
In [10]: final_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

history = final_model.fit([X_train,X_train], y_train, epochs=5, batch_size=200, verbose=2)
final_model.save('model_3.6L.h5')

Epoch 1/5
1800/1800 - 313s - loss: 0.6174 - accuracy: 0.7159 - 313s/epoch - 174ms/step
Epoch 2/5
1800/1800 - 324s - loss: 0.5669 - accuracy: 0.7362 - 324s/epoch - 180ms/step
Epoch 3/5
1800/1800 - 327s - loss: 0.5595 - accuracy: 0.7396 - 327s/epoch - 182ms/step
Epoch 4/5
1800/1800 - 323s - loss: 0.5563 - accuracy: 0.7427 - 323s/epoch - 180ms/step
Epoch 5/5
1800/1800 - 316s - loss: 0.5532 - accuracy: 0.7447 - 316s/epoch - 176ms/step
```

```
In [11]: import matplotlib.pyplot as plt

         %matplotlib inline
         plt.figure(1, figsize = (15,8))

         plt.subplot(221)
         plt.plot(history.history['accuracy'])
         plt.title('model accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')

         plt.subplot(222)
         plt.plot(history.history['loss'])
         plt.title('model loss')
         plt.ylabel('loss')
         plt.xlabel('epoch')

         plt.show()
```

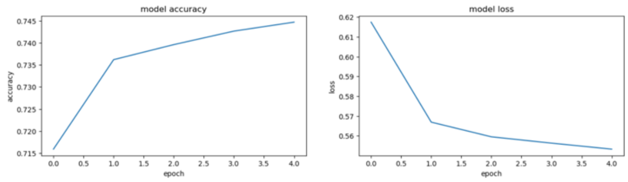

Figure 14: Train model with graph

Step 14: Create a new notebook to test the model, load the images and use the same model created in the train to test the model. The accuracy obtained was 73.5%.

```
In [15]: model= load_model('model_3.6L.h5')
         #model.summary()

In [16]: pred = model.predict([X_test, X_test])

         7500/7500 [==============================] - 66s 9ms/step

In [17]: from sklearn.metrics import accuracy_score

         print(accuracy_score(y_test,pred.round()))

         0.7351708333333333

In [18]: from sklearn.metrics import confusion_matrix, classification_report
         print(classification_report(y_test,pred.round()))

                       precision    recall  f1-score   support

                    0       0.76      0.35      0.48     80095
                    1       0.70      0.94      0.80     79650
                    2       0.76      0.91      0.83     80255

            micro avg       0.74      0.74      0.74    240000
            macro avg       0.74      0.74      0.71    240000
         weighted avg       0.74      0.74      0.71    240000
          samples avg       0.74      0.74      0.74    240000
```

Figure 15: Test Model

Step 15: Similarly, a prediction notebook is created with the model that is previously trained. A separate dataset with the images that can be provided as an input to the model for prediction.

```
In [11]:
         path = "IMG1/Pred/"
         predimgs = loadImages(path)

         data = np.array(predimgs)
         data = data.reshape([-1,50,50,1])


         X_pred = data/255
         print(X_pred.shape)

         (200000, 50, 50, 1)

In [ ]:

In [12]: model= load_model('model_3.6L.h5')
         pred = np.argmax(model.predict([X_pred, X_pred]),axis=1)
         df['Detection']=pred
         df['Detection'] = df['Detection'].replace(k)

         6250/6250 [==============================] - 50s 8ms/step

In [13]: df
Out[13]:
```

	dur	proto	state	tot_pkts	tot_bytes	src_bytes	label	dst_bytes	Detection
7960182	0.169702	1	21	2	514	454	7	60	Malicious traffic
6084324	3391.819824	1	21	4	1856	471	7	1385	Malicious traffic
612710	2428.668945	1	21	14	2278	1382	7	896	Suspicious traffic
8300545	0.003416	0	136	10	1430	955	5	475	Legitimate traffic
4852672	0.337290	1	21	2	135	75	3	60	Suspicious traffic
...	...	...	...	...	...	...	...	...	...
5516910	561.315430	0	136	117	7496	3173	5	4323	Legitimate traffic
14356	0.157974	1	21	2	240	85	1362	155	Malicious traffic
2588838	0.000229	1	21	2	222	83	1362	139	Malicious traffic
2435732	0.000619	1	21	2	255	193	7	62	Legitimate traffic
8974294	0.008163	0	136	10	1348	876	5	472	Suspicious traffic

200000 rows × 9 columns

Figure 16: Prediction

References

Apple Macbook Air (M1, 2020): full specs, tests and user reviews. (n.d.). Retrieved August 8, 2023, from https://nanoreview.net/en/laptop/apple-macbook-air-m1-2020

How to Install Keras on MacOS? - GeeksforGeeks. (n.d.). Retrieved August 8, 2023, from https://www.geeksforgeeks.org/how-to-install-keras-on-macos/