National College of Ireland

Configuration Manual

MSc Research Project

DEEP LEARNING-BASED INTRUSION DETECTION SYSTEM IN THE INTERNET OF THINGS

Abinash Mishra
X23153903

School of Computing
National College of Ireland

Supervisor: Eugene Mclaughlin

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Abinash Mishra
Student ID:	X23153903
Programme:	Msc Cybersecurity **Year:** 2024.
Module:	Msc Research Practicum
Lecturer:	Eugene Mclaughlin
Submission Due Date:	16-09-2024
Project Title:	DEEP LEARNING-BASED INTRUSION DETECTION SYSTEM IN THE INTERNET OF THINGS
Word Count:	1050 **Page Count:** 8

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: Abinash Mishra

Date: 16-09-2024

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

Abinash Mishra
X23153903

1 Introduction

The deep-learning best technique is implemented to detect the intrusion in the IoT network. This defines the evaluation of the IoT system devices by using the network analysis process. The execution supports to determine the factors that are necessary for the evaluation. The introduction determining approach defines the implementation of data construction process that is usable for the detection of network threats. The detection parameter defines the evaluation elements that are necessary for the execution (Amouri *et al.,* 2020). The deep learning techniques highlights the prototypes such as Linear Regression, SVM, Seqential Neural Network. All the algorithms are applicable to construct the predictive model for the intrusion detection process.

2 Project Overview

The functional parameter defines the relevance of the language and especially of Python in coding and specifically in the execution of deep learning methodical approaches. This evaluation sheds light on some of the fundamental components of Python coding that is relevant in the deployment of deep learning models. This leads to a more formal approach that ensures tasks in terms of a plan can be executed in a step wise manner. Some requirements of this process relate to identifying the correct elements that will enhance the possible success of the outcome. In general, the presented discussion covers all the aspects concerning effective utilization of deep learning parameters towards the prediction of the network behaviors. Also, the incorporation of detection mechanisms sheds light on the necessity of recognizing threats within a network as pointed out by Alkahtani and Aldhyani (2021).

3 Hardware/Software Implementation

1. Hardware
Processor: Intel Core i5
GPU : NVIDIA RTX 3050
Storage: 512 GB
RAM: 16 GB

2. Software
Jupyter Notebook: VERSION 7
Anaconda: 1.10

4 Data Collection

The data is collected form a secondary resource, 'Kaggle' named as IDS 2018 Intrusion CSVs (CSE-CIC-IDS2018) 02-14-2018.csv. This resource provides the necessary data which contains the details of the network parameters with necessary network details (Ge *et al.,* 2021). This defines the flow duration, packet forwarded section, labels, and many more. In this case labels define the section of attack types such as Benign, FTP-Brute Force, and SSH-Brute Force. This is the major target valuable for this investigation.

5 Data Analysis

```
import pandas as pd
import numpy as np
```

Figure 1: Libraries
(Source: Jupyter Notebook)

The libraries determines the evaluation of the key evaluation factors for the initialization of the examination process. This defines the execution of the testing approach which assists in the intrusion detection process.

```
df = pd.read_csv("02-14-2018.csv")
```

```
df
```

Timestamp	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	...	Fwd Seg Size Min	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	Label
14/02/2018 08:31:01	112641719	3	0	0	0	0	0	...	0	0.0	0.0	0	0	56320859.5	139.300036	56320958	56320761	Benign
14/02/2018 08:33:50	112641466	3	0	0	0	0	0	...	0	0.0	0.0	0	0	56320733.0	114.551299	56320814	56320652	Benign
14/02/2018 08:36:39	112638623	3	0	0	0	0	0	...	0	0.0	0.0	0	0	56319311.5	301.934596	56319525	56319098	Benign
14/02/2018 08:40:13	6453966	15	10	1239	2273	744	0	...	32	0.0	0.0	0	0	0.0	0.000000	0	0	Benign
14/02/2018 08:40:23	8804066	14	11	1143	2209	744	0	...	32	0.0	0.0	0	0	0.0	0.000000	0	0	Benign
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
14/02/2018 10:53:23	10156986	5	5	1089	1923	587	0	...	20	0.0	0.0	0	0	0.0	0.000000	0	0	Benign
14/02/2018 10:53:33	117	2	0	0	0	0	0	...	20	0.0	0.0	0	0	0.0	0.000000	0	0	Benign
14/02/2018 10:53:28	5095331	3	1	0	0	0	0	...	20	0.0	0.0	0	0	0.0	0.000000	0	0	Benign
14/02/2018 10:53:28	5235511	3	1	0	0	0	0	...	20	0.0	0.0	0	0	0.0	0.000000	0	0	Benign
14/02/2018 10:53:28	5807256	6	4	327	145	245	0	...	20	291569.0	0.0	291569	291569	5515650.0	0.000000	5515650	5515650	Benign

Figure 2: Data read
(Source: Jupyter Notebook)

The CSV data read functionality is implemented to read the collected secondary CSV data which contains necessary information about the network.

```
print(df.isnull().sum())
```

```
Dst Port         0
Protocol         0
Timestamp        0
Flow Duration    0
Tot Fwd Pkts     0
                ..
Idle Mean        0
Idle Std         0
Idle Max         0
Idle Min         0
Label            0
Length: 80, dtype: int64
```

Figure 3: Null check
(Source: Jupyter Notebook)

The null checking section to check the null factors present in the collected data. This initialize the cleaning of the data.

```
df = df.dropna()
```

```
df['Protocol'] = df['Protocol'].astype('category').cat.codes
```

Figure 4: Drop column
(Source: Jupyter Notebook)

Drop approach is implemented to remove or eliminate those columns which are not required for the determination. The type conversion approach is implemented to convert the type of a particular column, 'Prototype' into integer format.

```
print(df.dtypes)
```

```
Dst Port                    int64
Protocol                     int8
Timestamp          datetime64[ns]
Flow Duration              int64
Tot Fwd Pkts               int64
                             ...
Idle Mean                float64
Idle Std                 float64
Idle Max                   int64
Idle Min                   int64
Label                     object
Length: 80, dtype: object
```

Figure 5: Data type
(Source: Jupyter Notebook)

The type of the data defines the integer, date/time, float, and the object type data types.

```
df.describe()
```

	Dst Port	Protocol	Timestamp	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	...	Fv
count	1.046298e+06	1.046298e+06	1046298	1.046298e+06	1.046298e+06	1.046298e+06	1.046298e+06	1.046298e+06	1.046298e+06	1.046298e+06	...	1.(
mean	4.776915e+03	1.186851e+00	2018-02-14 05:54:15.973600	6.269168e+06	6.215780e+00	7.226879e+00	4.489686e+02	4.531644e+03	1.749535e+02	8.407793e+00	...	2.1
min	0.000000e+00	0.000000e+00	1970-01-05 03:01:17	-9.190110e+11	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	...	0.(
25%	2.200000e+01	1.000000e+00	2018-02-14 03:09:29	7.000000e+00	1.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	...	0.(
50%	5.300000e+01	1.000000e+00	2018-02-14 09:45:59	1.033000e+03	2.000000e+00	1.000000e+00	3.600000e+01	5.500000e+01	3.400000e+01	0.000000e+00	...	0.(
75%	4.430000e+02	1.000000e+00	2018-02-14 11:17:33	4.079898e+05	7.000000e+00	6.000000e+00	4.550000e+02	7.880000e+02	2.000000e+02	0.000000e+00	...	4.(
max	6.553300e+04	2.000000e+00	2018-02-14 12:59:59	1.200000e+08	5.115000e+03	9.198000e+03	8.591554e+06	1.339773e+07	6.444000e+04	1.460000e+03	...	1.(
std	1.430012e+04	4.179120e-01	NaN	1.261662e+09	4.452645e+01	1.049817e+02	1.575251e+04	1.516667e+05	2.878687e+02	1.950004e+01	...	5.5

8 rows × 79 columns

Figure 6: Descriptive statistics
(Source: Jupyter Notebook)

Descriptive determination evaluates the statistical determining factors that assist in the investigation.

```
sns.set(rc={'figure.figsize':(12, 6)})
plt.xlabel('Attack Type')
sns.set_theme()
ax = sns.countplot(x='Label', data=df)
ax.set(xlabel='Attack Type', ylabel='Number of Attacks')
plt.show()
```

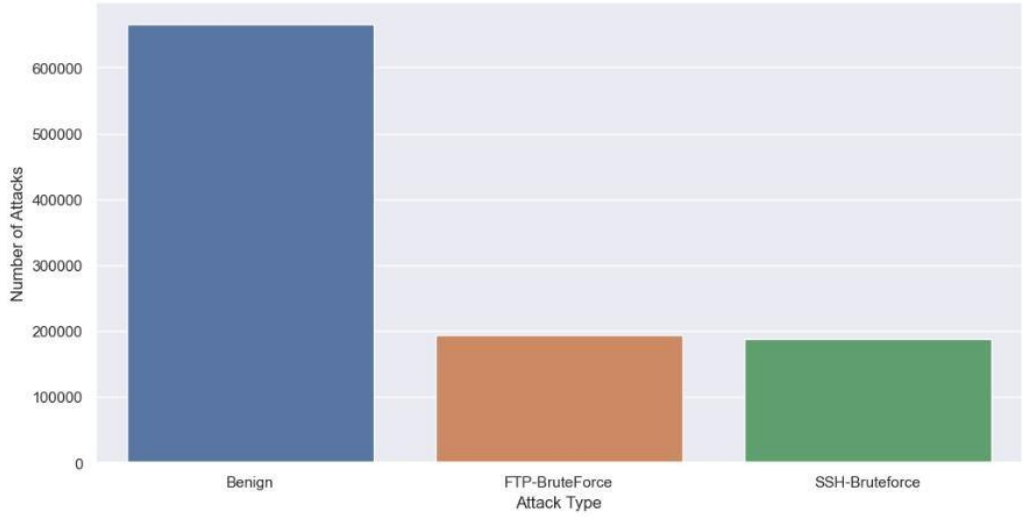

Figure 7: Visualization of various attacks
(Source: Jupyter Notebook)

The visualization defines the count of the number of Benign, FTP-Brute Force, and SSH-Brute Force types. In this case, Benign has maximum value of evaluation.

```
test_dataset = train_dataset.sample(frac=0.1)
target_train = train_dataset['Label']
target_test = test_dataset['Label']
target_train.unique(), target_test.unique()
```

```
(array([0, 1, 2]), array([1, 0, 2]))
```

Figure 8: Data preprocessing
(Source: Jupyter Notebook)

The data preprocessing determines the executable parameters such as test, and training data. This determines the target data section for the evaluation of the intrusion.

```
X = df.drop(columns=non_numeric_columns)
y = df['Label']
```

```
X.replace([np.inf, -np.inf], np.nan, inplace=True)
```

```
X.fillna(X.mean(), inplace=True)
```

```
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
```

```
y_categorical = to_categorical(y_encoded)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y_categorical, test_size=0.2, random_state=42)
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Figure 9: Data setting
(Source: Jupyter Notebook)

The data setting approach is implemented to set the data for the processing. This highlights the test, and train data spliting approach. The encoding method is implemented to encode the data section. The scaling process is implemented to transform the scaler part of the data that are usable for the execution.

```
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(y_train.shape[1], activation='softmax'))
```

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

```
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2)
```

```
Epoch 1/50
20926/20926 ━━━━━━━━━━━━━━━━━━━━ 19s 866us/step - accuracy: 0.9971 - loss: 0.0153 - val_accuracy: 0.9999 - val_loss: 9.2178e-04
Epoch 2/50
20926/20926 ━━━━━━━━━━━━━━━━━━━━ 16s 772us/step - accuracy: 0.9999 - loss: 6.8428e-04 - val_accuracy: 1.0000 - val_loss: 2.3830e-04
Epoch 3/50
20926/20926 ━━━━━━━━━━━━━━━━━━━━ 16s 777us/step - accuracy: 1.0000 - loss: 4.0820e-04 - val_accuracy: 0.9999 - val_loss: 0.0012
Epoch 4/50
20926/20926 ━━━━━━━━━━━━━━━━━━━━ 16s 774us/step - accuracy: 0.9999 - loss: 7.2139e-04 - val_accuracy: 1.0000 - val_loss: 2.9506e-04
Epoch 5/50
20926/20926 ━━━━━━━━━━━━━━━━━━━━ 16s 768us/step - accuracy: 1.0000 - loss: 4.5895e-04 - val_accuracy: 1.0000 - val_loss: 7.3996e-04
```

Figure 10: DL model setting and evaluation
(Source: Jupyter Notebook)

The deep learning model/prototype initialization approach is implemented in this point. This also defines the model fit approach and also the model execution functionality by using epochs.

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Accuracy: {accuracy*100:.2f}%')
```

```
6540/6540 [==============================] - 9s 1ms/step - loss: 8.1711e-04 - accuracy: 1.0000
Accuracy: 100.00%
```

```
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)
```

```
6540/6540 [==============================] - 9s 1ms/step
```

Figure 11: Loss, accuracy, and prediction of the model
(Source: Jupyter Notebook)

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
y = df['Label']
```

```
X.replace([np.inf, -np.inf], np.nan, inplace=True)
```

```
X.fillna(X.mean(), inplace=True)
```

```
model_linear = LinearRegression()
model_linear.fit(X_train, y_train)
```

```
▾ LinearRegression

LinearRegression()
```

Figure 12: Linear Regression
(Source: Jupyter Notebook)

The above code is explaining the preparing data, handling mising data and infinite values where:

model_linear = LinearRegression(): It initializes a Linear Regression model.
model_linear.fit(X_train, y_train): It trains the Linear Regression model by using the training data X_train and y_train.

```
mse_linear = mean_squared_error(y_test, y_pred_linear)
rmse_linear = mean_squared_error(y_test, y_pred_linear, squared=False)
r2_linear = r2_score(y_test, y_pred_linear)

print("Linear Regression:")
print("Mean squared Error:", rmse_linear)
print(f"R-squared: {r2_linear:.2f}")
```

Figure 13: Evaluation Metrics for Linear Regression Model
(Source: Jupyter Notebook)

The above code evaluates the performance of a Linear Regression model using three metrics: Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2). MSE measures the average squared difference between actual and predicted values, RMSE provides with the standard deviation of these errors, and R^2 indicates that how well the model explains the variability of the response data.

```
from sklearn.svm import SVC

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

svm_model = SVC(kernel='linear', C=1.0, random_state=42)

svm_model.fit(X_train, y_train)
```

```
SVC(kernel='linear', random_state=42)
```

Figure 14: Support Vector Machine (SVM) Model Training Process
(Source: Jupyter Notebook)

The above figure 14, shows that the training of a Support Vector Machine (SVM) model was using a linear kernel. It begins by encoding the target labels, splitting the dataset into training and testing sets, and standardizing the features. The SVM model is then initialized and then trained on the training data, with a specified regularization parameter C and with a random state for reproducibility.

```
def evaluate_model(y_test, y_pred, model_name):
    print(f"Model: {model_name}")
    print(f"Accuracy: {accuracy_score(y_test, y_pred) * 100:.2f}%")
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))
    print("\n")
```

```
# Evaluate each model
evaluate_model(y_test, y_pred, "Neural Network")
evaluate_model(y_test, y_pred_svm, "SVM")
evaluate_model(y_test, y_pred_lr_class, "Linear Regression")
```

Figure 15: Model Evaluation Function for Classification Models
(Source: Jupyter Notebook)

This code defines a function by evaluating the model by printing the evaluation metrics for different classification models w hich includes accuracy, classification report and confusion

matrix. It is then used to evaluate three models that are a Neural Network, an SVM, and a Linear Regression model which provides a comprehensive comparison of their performance on the test dataset.

References

Amouri, A., Alaparthy, V.T. and Morgera, S.D., 2020. A machine learning based intrusion dete ction system for mobile Internet of Things. Sensors, 20(2), p.461.

Alkahtani, H. and Aldhyani, T.H., 2021. Intrusion Detection System to Advance Internet of Things Infrastructure-Based Deep Learning Algorithms. Complexity, 2021(1), p.5579851.

Ge, M., Syed, N.F., Fu, X., Baig, Z. and Robles-Kelly, A., 2021. Towards a deep learning-driven intrusion detection approach for Internet of Things. *Computer Networks*, *186*, p.107784.