Configuration Manual

IDS for IoT to detect DDoS attacks using BiLSTM and cGAN

Hrishikesh Krishnan
Student ID: 22192727

School of Computing
National College of Ireland

Supervisor: Dr. Imran Khan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Hrishikesh Krishnan
Student ID:	22192727
Programme:	MSc Cybersecurity **Year:** 2024
Module:	Practicum
Lecturer:	Dr. Imran Khan
Submission Due Date:	16/09/2024
Project Title:	IDS for IoT to detect DDoS attacks using BiLSTM and cGAN
Word Count:	670 **Page Count:** 7

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: Hrishikesh Krishnan

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

Hrishikesh Krishnan
22192727

1 Introduction

This configuration manual gives the information regarding the applications, software and tools required for developing the BiLSTM model integrating cGAN for data augmentation. This guide is structured as following, where section 2 define system specification used for this project, section 3 lists the software tools and libraries used for developing this model. section 4 describe the configuration of deep learning in steps, the deep learning procedures and steps are included in step 5.

2 System Specification

The experiment setup was done on personal computer. Specification of the system as follows:

- MacBook Pro Mid 2015
- RAM:16 GB
- System Type-64-bit OS, x64-based processor
- Operating System: Mac OS
- Experiment setup: Anaconda Navigator [2], JupyterLab, Google Colab [1], Python 3.9.12 [3]

3 Software Specification

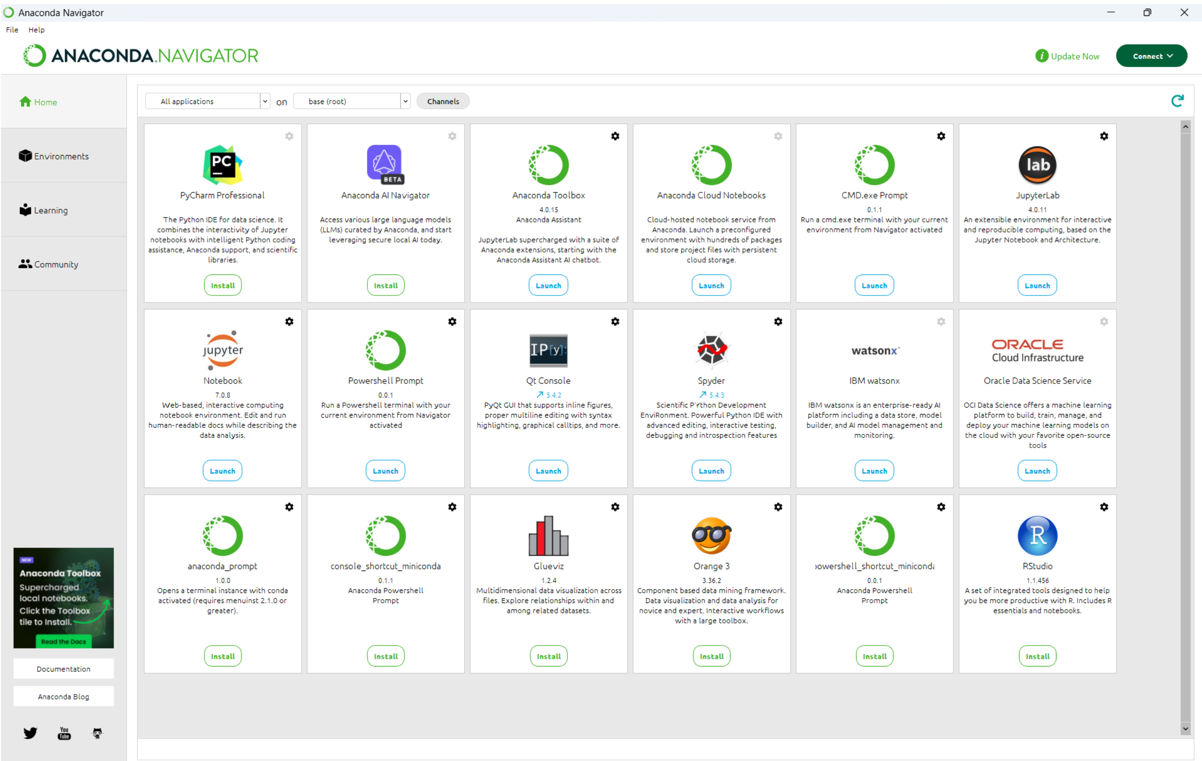
This section discusses the software used to build this model.

- **Google Colab**
- **Jupyter Notebook**
- **Python 3.9.12**
- **Pandas**
- **NumPy**
- **TensorFlow**
- **Matplotlib.pyplot**
- **seaborn**

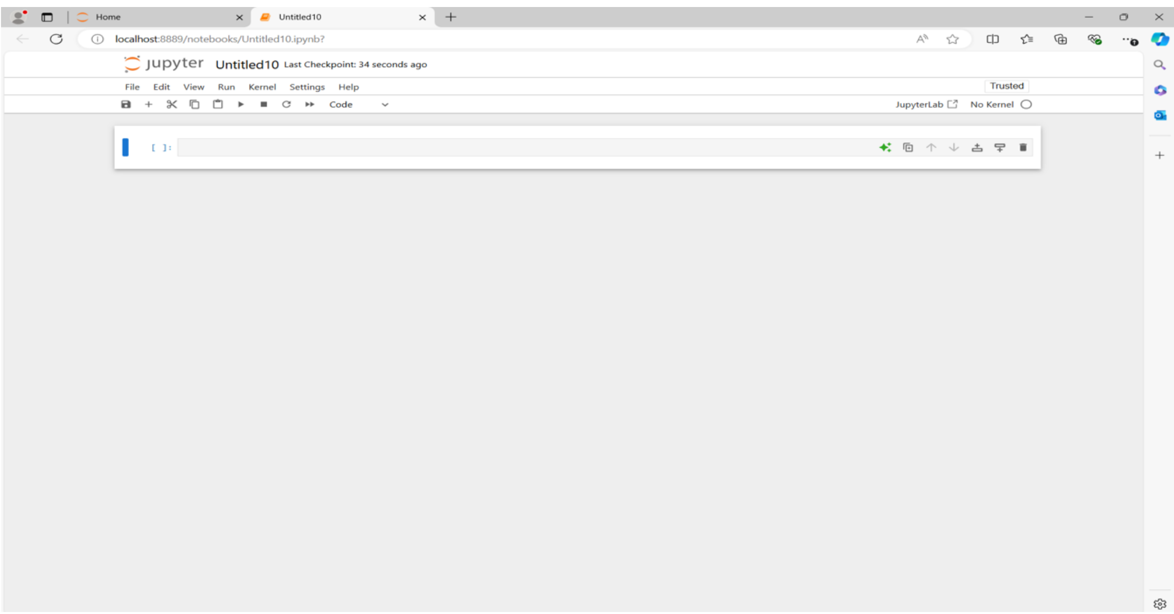
4 Configuration for Deep Learning

1. Download and install Anaconda2.6.0
2. Open Jupyter notebook with Python version and create new notebook in ‘. pynb’ file.

ANACONDA NAVIGATOR



Jupyter Notebook



5 Deep Learning Procedures

1. Dataset required for performing deep learning was downloaded.
2. Necessary libraries were imported.
3. Combine multiple datasets into single dataset.
4. Sampling of data for efficiently performing deep learning.
5. Pre-processing and Encoding Dataset
6. Reshaping and Splitting of Dataset
7. Converting to categorical
8. Defining and Building BiLSTM model
9. Obtain Model Summary
10. Training and evaluating the model
11. Generating Predictions
12. Evaluate the BiLSTM using metrices like accuracy, precision, recall, F-1 score and Classification report
13. Defining Conditional GAN- Generator and Discriminator
14. Training CGAN
15. Retraining BiLSTM using the output generated by CGAN
16. Evaluating the model using accuracy, precision, recall and F-1 score.
17. Visualized the model efficiency using confusion matrix, precision-recall curve and ROC curve.
18. Saved the Developed model.

➢ LIBRARIES

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler, label_binarize
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, confusion_matrix, roc_curve, auc
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Input, Embedding, Reshape, concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
```

➢ DATA SAMPLING

```
total_sample_size = 5000
labels = df['label'].unique()
num_labels = len(labels)
```

```
samples_per_label = total_sample_size // num_labels
```

```
sampled_data = pd.DataFrame()

for label in labels:
    label_data = df[df['label'] == label]
    sample_size = min(len(label_data), samples_per_label)
    sampled_label_data = label_data.sample(sample_size)
    sampled_data = pd.concat([sampled_data, sampled_label_data])
```

➢ DATA PRE-PROCESSING

```
#Preprocessing
data.columns = data.columns.str.strip()

data.replace([np.inf, -np.inf], np.nan, inplace=True)
data.dropna(inplace=True)

label_encoder = LabelEncoder()
data['label'] = label_encoder.fit_transform(data['label'])

X = data.drop('label', axis=1).select_dtypes(include='number')
y = data['label']


scaler = StandardScaler()
X = scaler.fit_transform(X)
```

➢ DATA SPLITTING

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

➢ MODEL DEFINING

```
# BiLSTM model

def build_rlstm_model(input_shape, num_classes):
    model = Sequential()
    model.add(Bidirectional(LSTM(128, return_sequences=True), input_shape=input_shape))
    model.add(Dropout(0.3))
    model.add(Bidirectional(LSTM(128)))
    model.add(Dropout(0.3))
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(num_classes, activation='softmax'))
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model



input_shape = (X_train.shape[1], X_train.shape[2])
rlstm_model = build_rlstm_model(input_shape, num_classes)
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
```

➢ **TRAIN BiLSTM**

```
# Training
rlstm_model.fit(X_train, y_train_cat, epochs=50, batch_size=64, validation_split=0.2, callbacks=[early_stopping])
```

➢ **EVALUATION**

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

➢ **CLASSIFICATION REPORT**

```
print(classification_report(y_test, y_pred, target_names=label_encoder.classes_))
```

➢ **cGAN GENERATOR**

```
# Generator

def build_generator(latent_dim, n_classes, seq_length, n_features):
    noise = Input(shape=(latent_dim,))
    label = Input(shape=(1,))
    label_embedding = Embedding(n_classes, latent_dim, input_length=1)(label)
    label_embedding = Reshape((latent_dim,))(label_embedding)
    model_input = concatenate([noise, label_embedding])
    x = Dense(256, activation='relu')(model_input)
    x = Dense(seq_length * n_features, activation='sigmoid')(x)
    x = Reshape((seq_length, n_features))(x)
    model = Model([noise, label], x)
    return model
```

➢ **cGAN DISCRIMINATOR**

```
# Discriminator

def build_discriminator(n_classes, seq_length, n_features):
    sample = Input(shape=(seq_length, n_features))
    label = Input(shape=(1,))
    label_embedding = Embedding(n_classes, seq_length * n_features, input_length=1)(label)
    label_embedding = Reshape((seq_length, n_features))(label_embedding)
    model_input = concatenate([sample, label_embedding])
    x = Dense(256, activation='relu')(model_input)
    x = Dense(1, activation='sigmoid')(x)
    model = Model([sample, label], x)
    return model
```

➢ TRAIN CGAN

```python
# Training CGAN
epochs = 20000
batch_size = 64
for epoch in range(epochs):

    idx = np.random.randint(0, X_train.shape[0], batch_size)
    real_samples = X_train[idx]
    labels = y_train_np[idx].reshape(-1, 1)
    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    gen_samples = generator.predict([noise, labels])

    d_loss_real = discriminator.train_on_batch([real_samples, labels], np.ones((batch_size, 1)))
    d_loss_fake = discriminator.train_on_batch([gen_samples, labels], np.zeros((batch_size, 1)))
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    valid_y = np.ones((batch_size, 1))
    g_loss = cgan.train_on_batch([noise, labels], valid_y)

    if epoch % 1000 == 0:
        print(f"{epoch} [D loss: {d_loss[0]} | D accuracy: {100*d_loss[1]}] [G loss: {g_loss}]")
```

➢ CONFUSION MATRIX

```python
# Confusion Matrix

conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

➢ ROC CURVE

```python
# ROC Curve

y_test_bin = label_binarize(y_test, classes=[0, 1, 2, 3])[:, 0]

y_scores = y_pred_prob[:, 0]

fpr, tpr, _ = roc_curve(y_test_bin, y_scores)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

- **PRECISION RECALL CURVE**

```
# Precision Recall Curve

y_test_bin = label_binarize(y_test, classes=[0, 1, 2, 3])[:, 0]

precision, recall, _ = precision_recall_curve(y_test_bin, y_pred_prob[:, 0])
average_precision = average_precision_score(y_test_bin, y_pred_prob[:, 0])

plt.figure()
plt.plot(recall, precision, color='blue', lw=2, label='Precision-Recall curve (AP = %0.2f)' % average_precision)
plt.fill_between(recall, precision, alpha=0.2, color='blue')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Precision-Recall curve')
plt.legend(loc="lower left")
plt.show()
```

- **SAVING THE MODEL**

```
# Save model

cgan.save('full_model.h5')
```

6 References

[1] Googlecolab, Available at: colab.google

[2] Anaconda, 2022.Anaconda [online] Available at: Download Anaconda Distribution | Anaconda

[3] Python Download [Online] Available at: Download Python | Python.org