National College of Ireland

Configuration Manual

MSc Research Project
MSc Data Analytics

Ebin Sujin
Student ID: x23205814

School of Computing
National College of Ireland

Supervisor: Christian Horn

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	EBIN SUJIN
Student ID:	x23205814
Programme:	MSc Data Analytics **Year:** 2024
Module:	MSc Research Project
Lecturer:	Christian Horn
Submission Due Date:	12-12-2024
Project Title:	Configuration Manual
Word Count:	592 **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: EBIN SUJIN

Date: 12-12-2024

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

Ebin Sujin
x23205814

1 Introduction

This configuration guide will present the following required hardware, software, and libraries in order to run deep learning or simple machine learning models in the project. It elaborates on settings and instruments that incorporate the pre-processing of the datasets, feature selection and the optimization of the model applied to discover DDoS attacks.

2 Hardware Specifications

The below figure shows the overview of the hardware of the laptop device used for the project.

Hardware Overview:

Model Name:	MacBook Air
Model Identifier:	Mac14,2
Model Number:	MLY33HN/A
Chip:	Apple M2
Total Number of Cores:	8 (4 performance and 4 efficiency)
Memory:	8 GB
System Firmware Version:	10151.1.1
OS Loader Version:	10151.1.1
Serial Number (system):	FN64PC79FK
Hardware UUID:	8AF02C68-4F98-569F-BB05-6105EAAFF2E7
Provisioning UDID:	00008112-00127844148B401E
Activation Lock Status:	Enabled

Hardware Overview

3 Software and Language

Software:

- Google Colab for training and testing the models.
- Jupyter Notebook for some local experiments.

Programming Language:

- Python

4 Python Libraries Used

- **File Handling & Data Management:** Pandas & Numpy
- **Data Visualization:** Matplotlib, Seaborn, Ploty
- **Machine Learning Utilities:** Scikit-learn, Synthetic Minority Oversampling Technique (SMOTE).
- **Deep Learning Libraries:** TensorFlow, Keras

5 Dataset Collection

The project utilized the UNSW-NB15 dataset, which includes normal and malicious network traffic data. The dataset comprises 2.5 million records across nine attack types and normal traffic, making it suitable for intrusion detection studies.

Link: https://www.kaggle.com/datasets/mrwellsdavid/unsw-nb15

6 Coding Implementation

6.1 Mounting Google Drive:

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive
```

6.2 Importing Libraries:

```
[ ] import numpy as np
    import pandas as pd
    import seaborn as sns
    import plotly.express as px
    import matplotlib.pyplot as plt
    import plotly.graph_objects as go
    import plotly.figure_factory as ff
    from sklearn import ensemble
    from imblearn.over_sampling import SMOTE
    from sklearn.preprocessing import LabelEncoder
    from sklearn.model_selection import train_test_split
    from tensorflow.keras.models import Sequential, Model, load_model
    from tensorflow.keras.optimizers import Adam, SGD, RMSprop
    from sklearn.preprocessing import LabelBinarizer,MinMaxScaler
    from sklearn.linear_model import LogisticRegression
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
    from tensorflow.keras.layers import Dense, Dropout, LSTM, Bidirectional, BatchNormalization, Input, LeakyReLU

    import warnings
    warnings.filterwarnings('ignore')
```

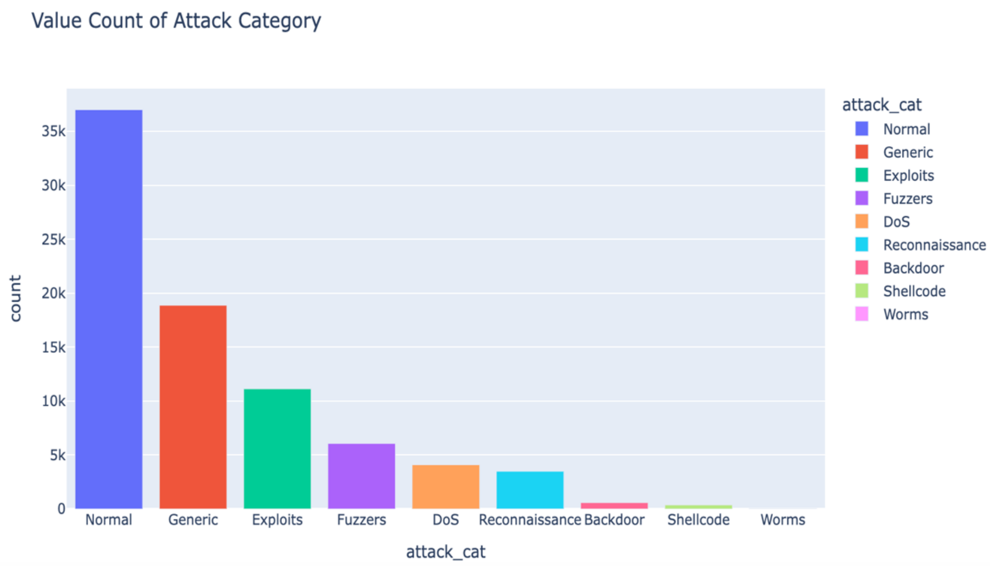
6.3 Data Loading:

```
[ ] dataframe = pd.read_csv('/content/drive/MyDrive/network_intrusion_unsw/Data/UNSW_NB15_training-set.csv')
    dataframe.head()
```

	id	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	...	ct_dst_sport_ltm	ct_dst_src_ltm	is_ftp_login	ct_ftp_cmd	ct_flw_http_mthd	c
0	1	0.000011	udp	-	INT	2	0	496	0	90909.0902	...	1	2	0	0	0	
1	2	0.000008	udp	-	INT	2	0	1762	0	125000.0003	...	1	2	0	0	0	
2	3	0.000005	udp	-	INT	2	0	1068	0	200000.0051	...	1	3	0	0	0	
3	4	0.000006	udp	-	INT	2	0	900	0	166666.6608	...	1	3	0	0	0	
4	5	0.000010	udp	-	INT	2	0	2126	0	100000.0025	...	1	3	0	0	0	

6.4 Data Analysis (EDA)

```
[ ] tempdf = dataframe['attack_cat'].value_counts().reset_index()
    fig = px.bar(tempdf, y='count', x='attack_cat', color='attack_cat',title="Value Count of Attack Category")
    fig.show()
```



The figure shows the distribution of attack categories, with "Normal" traffic having the highest count and categories like "Backdoor" and "Worms" having the fewest.

6.5 Data Pre-processing

```
[ ] #splitting data into X and y
    X = dataframe.drop(['attack_cat','label'], axis='columns')
    y = dataframe['attack_cat']
```

```
[ ] col = X.select_dtypes(exclude=['float64','int64']).columns.tolist()
    col
```

```
['proto', 'state']
```

```
[ ] #label encoding(converting categorical value to numeric)
    le = LabelEncoder()
    X[col] = X[col].apply(le.fit_transform)
    X.head()
```

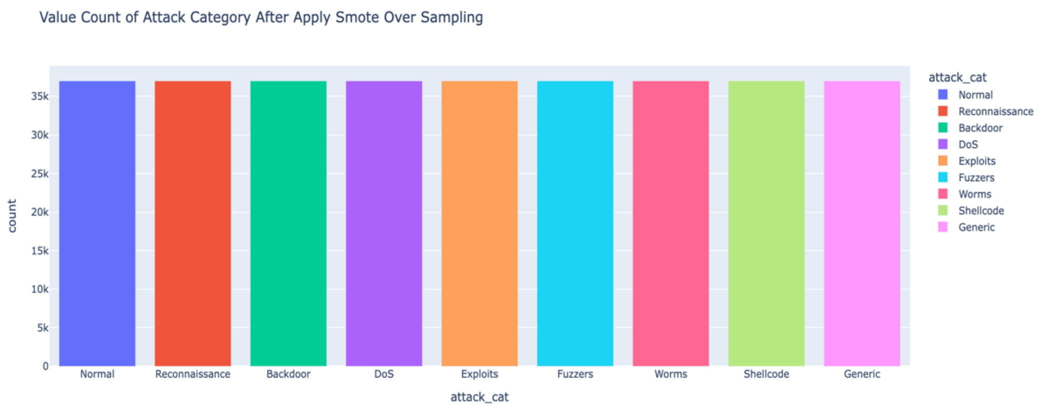
This code prepares the dataset by splitting it into features (X) and target labels (y). It identifies non-numeric categorical columns and applies **label encoding** to convert them into numeric values, making the data suitable for machine learning models. This step ensures all features in X are numeric, while y contains the target attack categories, ready for training and classification tasks.

```
[ ] # scale data
    col = X.columns
    minmaxtranform = MinMaxScaler()
    X = minmaxtranform.fit_transform(X)
    X
```

This code normalizes the feature dataset X using **MinMaxScaler**, scaling all values to a range of 0 to 1. This ensures all features are on the same scale, improving model training efficiency and accuracy by preventing larger features from dominating the learning process.

```
[ ] #data balancing
    orsamp = SMOTE()
    X, y = orsamp.fit_resample(X, y)
```

```
[ ] tempdf = y.value_counts().reset_index()
    fig = px.bar(tempdf, y='count', x='attack_cat', color='attack_cat',title="Value Count of Attack Category After Apply Smote Over Sampling")
    fig.show()
```



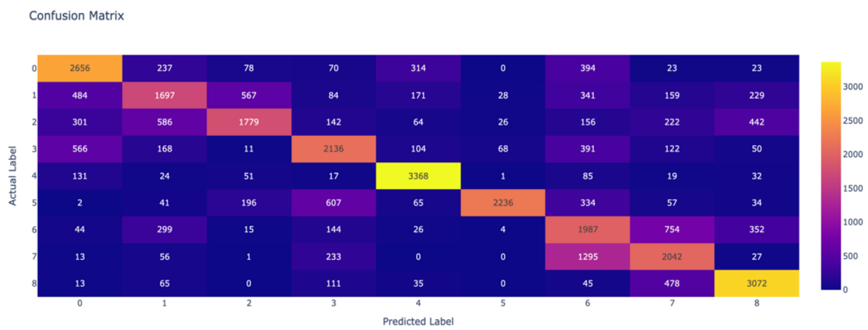
The data extracted from the repository are imbalanced in nature; so we need to balance the data, as all the respective algorithm can be performed on them. For this purpose, SMOTE is applied which is a naive method of duplicating minority example. In the dataset, the dataset is sampled by oversampling the smaller classes, so that the problem of data imbalance is solved.

7 Machine Learning Models

Case Study 1: Logistic Regression Model

```
[ ] lrmodel = LogisticRegression()
    lrmodel.fit(X_train,y_train)
    y_pred = lrmodel.predict(X_test)
```

```
[ ] #confusion Matrix
    matrix=confusion_matrix(y_test, y_pred)
    fig = px.imshow(matrix, text_auto=True, aspect="auto", title="Confusion Matrix", labels=dict(x="Predicted Label", y="Actual Label"))
    fig.show()
```



Classification Report:

```
[ ] #Classification Report
    print("Classification Report : ")
    print(classification_report(y_test, y_pred))
```

```
Classification Report :
                precision    recall  f1-score   support

      Backdoor       0.63      0.70      0.66      3795
           DoS       0.53      0.45      0.49      3760
      Exploits       0.66      0.48      0.55      3718
       Fuzzers       0.60      0.59      0.60      3616
       Generic       0.81      0.90      0.86      3728
        Normal       0.95      0.63      0.75      3572
Reconnaissance       0.40      0.55      0.46      3625
     Shellcode       0.53      0.56      0.54      3667
         Worms       0.72      0.80      0.76      3819

      accuracy                           0.63     33300
     macro avg       0.65      0.63      0.63     33300
  weighted avg       0.65      0.63      0.63     33300
```

Output of Logistic Regression Model
Accuracy Score: 0.63

Case Study 2: Decision Tree Classifier

```
[ ] dtcmodel = DecisionTreeClassifier(max_depth=6)
    dtcmodel.fit(X_train,y_train)
    y_pred = dtcmodel.predict(X_test)
```

```
[ ] #confusion Matrix
    matrix=confusion_matrix(y_test, y_pred)
    fig = px.imshow(matrix, text_auto=True, aspect="auto", title="Confusion Matrix", labels=dict(x="Predicted Label", y="Actual Label"))
    fig.show()
```

Classification Report:

```
[ ] #Classification Report
    print("Classification Report : ")
    print(classification_report(y_test, y_pred))
```

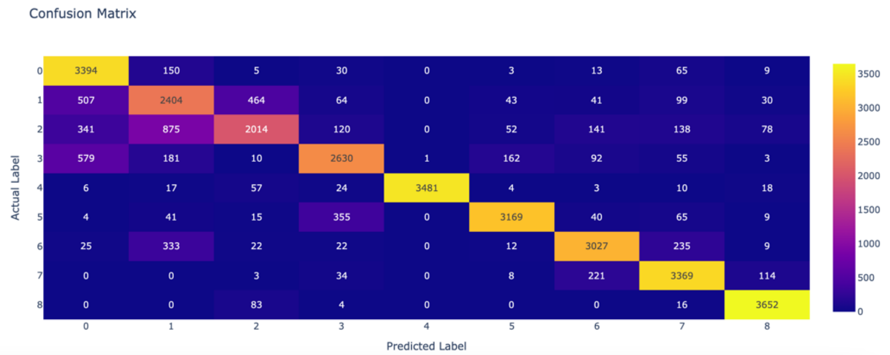
```
Classification Report :
                precision    recall  f1-score   support

      Backdoor       0.68      0.82      0.74      3795
           DoS       0.61      0.45      0.52      3760
      Exploits       0.61      0.49      0.54      3718
       Fuzzers       0.64      0.64      0.64      3616
       Generic       1.00      0.95      0.97      3728
        Normal       0.76      0.77      0.77      3572
Reconnaissance       0.53      0.75      0.62      3625
     Shellcode       0.47      0.50      0.49      3667
         Worms       0.93      0.78      0.85      3819

      accuracy                           0.68     33300
     macro avg       0.69      0.68      0.68     33300
  weighted avg       0.70      0.68      0.68     33300
```

Output of Decision Tree Classifier
Accuracy Score: 0.68

8 Deep Learning without Autoencoder Feature Extraction

Case Study 3: Long Short Term Memory (LSTM)

```
[ ] #confusion Matrix
    matrix=confusion_matrix(y_test_org, y_pred)
    fig = px.imshow(matrix, text_auto=True, aspect="auto", title="Confusion Matrix", labels=dict(x="Predicted Label", y="Actual Label"))
    fig.show()
```



Classification Report:

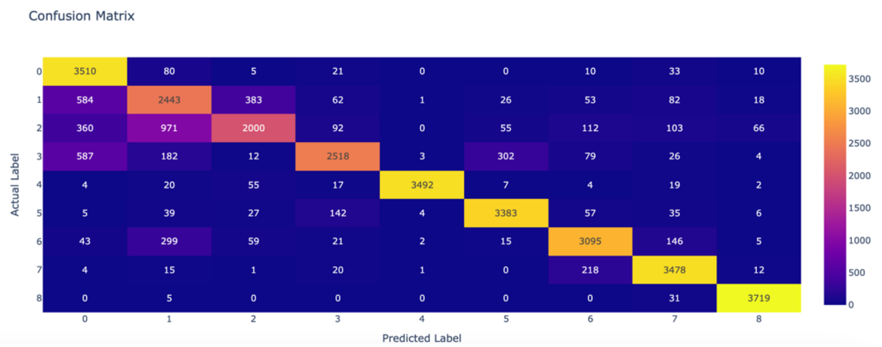
```
[ ] #Classification Report
    print("Classification Report : ")
    print(classification_report(y_test_org, y_pred))
```

```
Classification Report :
              precision    recall  f1-score   support

           0       0.70      0.93      0.80      3669
           1       0.60      0.66      0.63      3652
           2       0.75      0.54      0.63      3759
           3       0.80      0.71      0.75      3713
           4       1.00      0.96      0.98      3620
           5       0.92      0.86      0.89      3698
           6       0.85      0.82      0.83      3685
           7       0.83      0.90      0.86      3749
           8       0.93      0.97      0.95      3755

    accuracy                           0.82     33300
   macro avg       0.82      0.82      0.81     33300
weighted avg       0.82      0.82      0.81     33300
```

Output of LSTM without Autoencoder
Accuracy Score: 0.82

Case Study 4: Bidirectional Long Short Term Memory (Bi-LSTM)

```
[ ] #confusion Matrix
    matrix=confusion_matrix(y_test_org, y_pred)
    fig = px.imshow(matrix, text_auto=True, aspect="auto", title="Confusion Matrix", labels=dict(x="Predicted Label", y="Actual Label"))
    fig.show()
```



Classification Report:

```
[ ] #Classification Report
    print("Classification Report : ")
    print(classification_report(y_test_org, y_pred))
```

```
Classification Report :
              precision    recall  f1-score   support

           0       0.69      0.96      0.80      3669
           1       0.60      0.67      0.63      3652
           2       0.79      0.53      0.63      3759
           3       0.87      0.68      0.76      3713
           4       1.00      0.96      0.98      3620
           5       0.89      0.91      0.90      3698
           6       0.85      0.84      0.85      3685
           7       0.88      0.93      0.90      3749
           8       0.97      0.99      0.98      3755

    accuracy                           0.83     33300
   macro avg       0.84      0.83      0.83     33300
weighted avg       0.84      0.83      0.83     33300
```

Output of Bi-LSTM without Autoencoder
Accuracy Score: 0.83

9 Deep Learning with Autoencoder Feature Extraction

```
[ ] # define encoder
    n_inputs = int(X_train.shape[1])
    visible = Input(shape=(n_inputs,))
    # encoder level 1
    e = Dense(n_inputs*2)(visible)
    e = BatchNormalization()(e)
    e = LeakyReLU()(e)
    # encoder level 2
    e = Dense(n_inputs)(e)
    e = BatchNormalization()(e)
    e = LeakyReLU()(e)
    # bottleneck
    n_bottleneck = round(float(n_inputs) / 2.0)
    bottleneck = Dense(n_bottleneck)(e)
    #n_bottleneck = n_inputs
    #bottleneck = Dense(n_bottleneck)(e)

    # define decoder, level 1
    d = Dense(n_inputs)(bottleneck)
    d = BatchNormalization()(d)
    d = LeakyReLU()(d)
    # decoder level 2
    d = Dense(n_inputs*2)(d)
    d = BatchNormalization()(d)
    d = LeakyReLU()(d)
    # output layer
    output = Dense(n_inputs, activation='linear')(d)
    # define autoencoder model
    model = Model(inputs=visible, outputs=output)
    # compile autoencoder model
    model.compile(optimizer='adam', loss='mse',metrics=['mse'])
    model.summary()
```

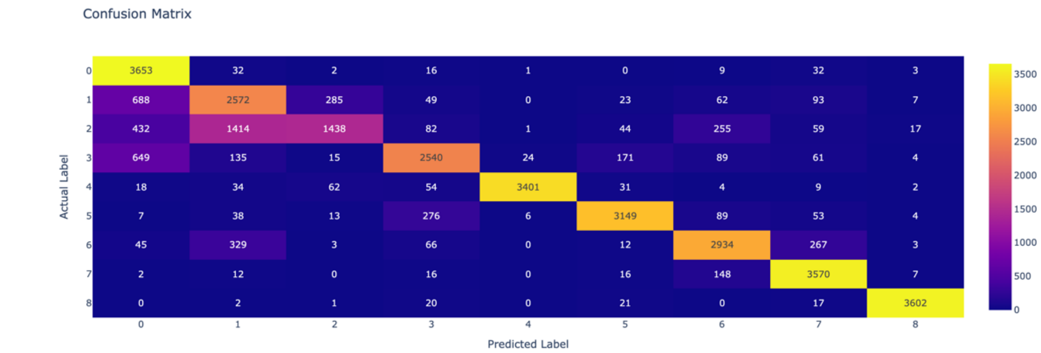
Training Autoencoder Model for Feature Extraction

```
[ ] encoder = Model(inputs=visible, outputs=bottleneck)
    encoder.summary()
    #encoder.save('/content/drive/MyDrive/network_intrusion_unsw/Models/encoder_smote.h5')
```

We have created this encoder_smote.h5 file to store the model for the Autoencoder. Whenever the Autoencoder performs feature reduction and selection, 1-2 features shuffle because the data in the train-test split is randomized every time during selection. So, we have stored the features selected by the Autoencoder in an encoder_smote.h5 file to ensure they remain the same in every execution. If the features keep changing every time, the accuracies of the models were changing. Hence, we stored it in an encoder_smote.h5 file.

Case Study 5: Long Short Term Memory (LSTM)

```
[ ] #confusion Matrix
    matrix=confusion_matrix(y_test_org, y_pred)
    fig = px.imshow(matrix, text_auto=True, aspect="auto", title="Confusion Matrix", labels=dict(x="Predicted Label", y="Actual Label"))
    fig.show()
```



Classification Report:

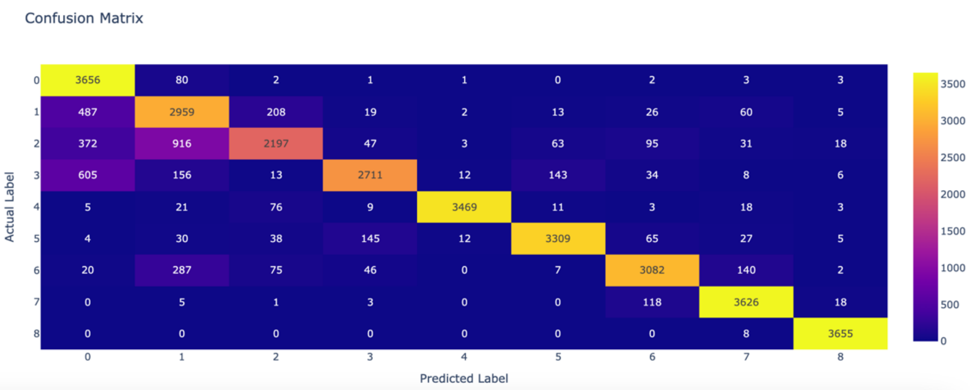
```
[ ] #Classification Report
    print("Classification Report : ")
    print(classification_report(y_test_org, y_pred))
```

```
Classification Report :
              precision    recall  f1-score   support

           0       0.66      0.97      0.79      3748
           1       0.56      0.68      0.62      3779
           2       0.79      0.38      0.52      3742
           3       0.81      0.69      0.75      3688
           4       0.99      0.94      0.97      3615
           5       0.91      0.87      0.89      3635
           6       0.82      0.80      0.81      3659
           7       0.86      0.95      0.90      3771
           8       0.99      0.98      0.99      3663

    accuracy                           0.81     33300
   macro avg       0.82      0.81      0.80     33300
weighted avg       0.82      0.81      0.80     33300
```

Output of LSTM with Autoencoder
Accuracy Score: 0.81

Case Study 6: Bidirectional Long Short Term Memory (Bi-LSTM)

```
[ ] #confusion Matrix
    matrix=confusion_matrix(y_test_org, y_pred)
    fig = px.imshow(matrix, text_auto=True, aspect="auto", title="Confusion Matrix", labels=dict(x="Predicted Label", y="Actual Label"))
    fig.show()
```



Classification Report:

```
[ ] #Classification Report
    print("Classification Report : ")
    print(classification_report(y_test_org, y_pred))
```

```
Classification Report :
              precision    recall  f1-score   support

           0       0.71      0.98      0.82      3748
           1       0.66      0.78      0.72      3779
           2       0.84      0.59      0.69      3742
           3       0.91      0.74      0.81      3688
           4       0.99      0.96      0.98      3615
           5       0.93      0.91      0.92      3635
           6       0.90      0.84      0.87      3659
           7       0.92      0.96      0.94      3771
           8       0.98      1.00      0.99      3663

    accuracy                           0.86     33300
   macro avg       0.87      0.86      0.86     33300
weighted avg       0.87      0.86      0.86     33300
```

Output of Bi-LSTM with Autoencoder
Accuracy Score: 0.86