Configuration Manual

MSc Research Project
Data Analytics

Jonah Abhijit Papaiah
Student ID: x20189419

School of Computing
National College of Ireland

Supervisor: Prashanth Nayak

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	…… Jonah Abhijit Papaiah……………………………………………………………………
Student ID:	……x20189419……………………………….………………………………………………..
Programme:	……MSc in Data Analytics………… **Year:** ……2023……..
Module:	……MSc in Research Project…………………………….…….……………………….
Supervisor:	……Prashanth Nayak……………………………………….………………
Submission Due Date:	……15 December 2022…………………………………….….………………
Project Title:	…. Enhancing the Classification Accuracy of intrusion Detection system using Auto-encoder Algorithm …….………
Word Count:	………1087……………**Page Count:** …………12…………..…..………

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: ………Jonah Abhijit Papaiah……………………………

Date: …15 December 2022……………………………………………

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

Jonah Abhijit Papaiah
X20189419

1 Introduction

The Configuration Manual lists all the parameters and configurations that were used during this research, including installation details and prerequisites. The handbook contains a step-by-step explanation of how to run the application.

2 System Configuration

In the system configuration section details about the system requirements and software required for the implementation has been discussed.

2.1 Hardware Requirement

Operating System	Windows 10
Installed RAM	16.0 GB
System type	64-bit operating system, x64-based processor
Pen and touch	Pen and touch support with 10 touch points

2.2 Software Requirement

The open-source IDE Jupyter Notebook, which is accessible through the Anaconda Software, was used in this research project. This environment is based on a Python Module. Installing each of these packages is necessary before the project can be built.

3 Installation and Environment Setup

3.1 Python

In this research project python package was used. Since most of the Deep Learning and machine learning projects are supported by its numerous built-in libraries. With a variety of plots, it makes developing and analysing models easier. Installing the most recent version of a python on the machine is the first prerequisite. The package installer can be downloaded from the website using your browser depending on your operating system. Enter “python-version” in the command prompt to confirm that python has been successfully installed from the website after installation, as shown in the below figure.

Figure: Python Website Page

3.2 Anaconda

The anaconda package includes several IDE that are helpful for writing code and analysing outputs from python packages. Anaconda Navigator has a wide variety of IDEs available, the model in this project is constructed in Jupyter Notebook.

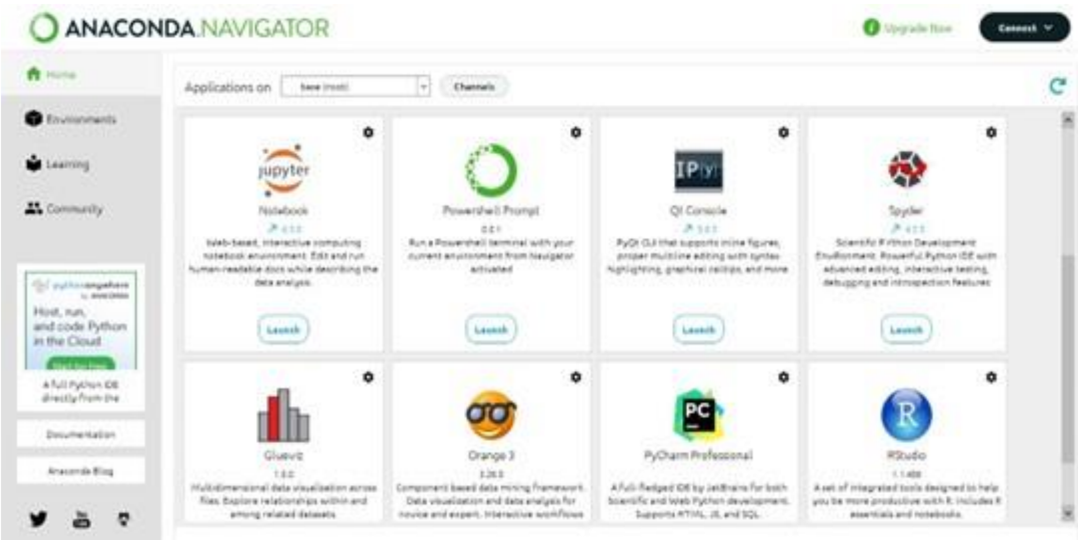

Figure: Anaconda Navigator

3.3 Jupyter Notebook

Jupyter notebook and its tasks are launched in browser tabs from the anaconda navigator. Python notebooks are first created and saved in the .ipynb format. Using the pip command, the python libraries are installed during the execution of code. The libraries numpy, pandas, tensorflow, matplotlib, seaborn, and plotly are the required libraries in this project.

4 Data Collection

There are datasets in this project and have been taken from an open-source website, Kaggle. The next sections are split into two sections, one for the KDD-CUP dataset and then for the IDS-CSECIC dataset.

5 Implementation of CSE-CIC-IDS2018 dataset

5.1 Importing Libraries

For the implementation of the model, the required libraries must be imported for a smooth execution. Below figure shows the imported libraries for our study.

```
In [7]:
import pandas as pd
import numpy as np
import plotly.express as px
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.under_sampling import RandomUnderSampler
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

#from tensorflow.python.keras.layers import Conv2D, Conv1D, MaxPooling2D, MaxPooling1D, Flatten, Bc
#from tensorflow.python.keras.models import Sequential
import tensorflow as tf
```

Figure : Importing Required Libraries

5.2 Data Visualization

The data in the dataset is being visualized using various graphs. The below figures show the visualization of data to get a better understanding on the data.

```
px.pie(names = data.Label.value_counts().index, values=data.Label.value_counts().values,hole=0.5,title="Label Distribution")
```

Figure: Pie chart to see the distribution of data

5.3 Data Pre-processing

In the data pre-processing, the null values in the data are being dropped, then the data has been formatted., along with unnecessary columns has been dropped. The below figure shows the part of the code where the data is being pre-processed.

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

```
data['Hour'] = pd.to_datetime(data.Timestamp).dt.hour
data.drop("Timestamp",axis=1,inplace=True)
```

```
labels = data.Label
data.drop(["Protocol","PSH Flag Cnt","Init Fwd Win Byts","Flow Byts/s","Flow Pkts/s", "Label"],axis=1,inplace=True)
```

```
under_sampling = RandomUnderSampler()
features = data.values

features,labels = under_sampling.fit_resample(features,labels)
```

Figure: Data Pre-processing

5.4 Splitting of Train and Test Data

The dataset must be divided into training and validation of data so that the model can be developed. The below figure shows how the data is divided into train and test phase.

```
X_train,X_test,Y_train,Y_test = train_test_split(features,labels,test_size=0.25)
```

```
n_steps= 1
epochs= 15
verbose = 1
batch_size = 64
n_features = X_train.shape[1]
n_outputs = Y_train.shape[1]
x = [i for i in range(1,epochs+1)]
```

Figure: Splitting of Train and Test data

5.5 Model Training

In model training, four models have been used one is Autoencoder, CNN, LSTM and Conv-LSTM. The below figures show the implementation of models.

```
#AutoEncoder Model
def AutoEncoderModel(n_steps,n_features,n_outputs):
    model = tf.keras.models.Sequential(name = 'CNN')

    model.add(tf.keras.layers.Input(shape=(n_steps, n_features)))
    model.add(tf.keras.layers.BatchNormalization())
    # Encoder Layer 1
    model.add(tf.keras.layers.Conv1D(filters=n_features*4, kernel_size=1, activation='relu'))
    # Encoder Layer 2
    model.add(tf.keras.layers.Conv1D(filters=n_features*2, kernel_size=1, activation='relu'))
    # Encoder Layer 3
    model.add(tf.keras.layers.Conv1D(filters=n_features, kernel_size=1, activation='relu'))
    model.add(tf.keras.layers.Dropout(0.2))
    # BottleNeck
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(n_features,activation='relu'))
    model.add(tf.keras.layers.Reshape((n_steps,n_features)))

    # Decoder Layer 1
    model.add(tf.keras.layers.Conv1DTranspose(filters=n_features,kernel_size=1, activation='relu'))
    # Decoder Layer 2
    model.add(tf.keras.layers.Conv1DTranspose(filters=n_features*2,kernel_size=1, activation='relu'))
    # Decoder Layer 3
    model.add(tf.keras.layers.Conv1DTranspose(filters=n_features*4,kernel_size=1, activation='relu'))

    # Classification Layer
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(n_outputs,"softmax"))

    model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy",precision,recall])
    print(model.summary())

    return model
```

```
enc_classifier = AutoEncoderModel(n_steps, n_features,n_outputs)
history_enc = enc_classifier.fit(X_train, Y_train,validation_data=(X_test,Y_test), epochs=epochs, verbose=verbose,batch_size=
```

Figure: Autoencoder Model

CNN Model

```python
def CNNmodel(n_steps,n_features,n_outputs):
    model = tf.keras.models.Sequential(name = 'CNN')

    model.add(tf.keras.layers.Input(shape=(n_steps, n_features)))

    model.add(tf.keras.layers.Conv1D(filters=12, kernel_size=1, activation='relu'))
    model.add(tf.keras.layers.Conv1D(filters=8, kernel_size=1, activation='relu'))

    model.add(tf.keras.layers.Flatten())

    model.add(tf.keras.layers.Dense(128,activation='relu'))
    model.add(tf.keras.layers.Dense(n_outputs,"softmax"))

    model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy",precision,recall])
    print(model.summary())
    return model
```

```python
model = CNNmodel(n_steps, n_features,n_outputs)
history_cnn = model.fit(X_train,Y_train,validation_data = (X_test,Y_test),epochs = epochs)
```

Figure: CNN model

LSTM Model

```python
def LSTMmodel(n_steps,n_features,n_outputs):
    model = tf.keras.models.Sequential(name = 'LSTM')

    model.add(tf.keras.layers.Input(shape=(n_steps, n_features)))

    model.add(tf.keras.layers.LSTM(12, activation='relu', return_sequences=True))
    model.add(tf.keras.layers.LSTM(8, activation='relu'))
    model.add(tf.keras.layers.Dense(3,activation='softmax'))

    model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy",precision,recall])
    print(model.summary())
    return model
```

```python
LSTMclassifier =LSTMmodel(n_steps, n_features,n_outputs)
history_LSTM = LSTMclassifier.fit(X_train, Y_train,validation_data=(X_test,Y_test), epochs=epochs, verbose=verbose,batch_size
```

Figure: LSTM model

```python
#Conv-Lstm
# define model
def CONVlstm(n_steps,n_features,n_outputs):
    model = tf.keras.models.Sequential(name = 'CONV-LSTM')

    model.add(tf.keras.layers.Input(shape=(None,n_steps, n_features)))
    model.add(tf.keras.layers.BatchNormalization())

    model.add(tf.keras.layers.TimeDistributed(tf.keras.layers.Conv1D(filters=12, kernel_size=1, activation='relu')))
    model.add(tf.keras.layers.TimeDistributed(tf.keras.layers.Dropout(0.5)))
    model.add(tf.keras.layers.TimeDistributed(tf.keras.layers.Flatten()))

    model.add(tf.keras.layers.LSTM(12, activation='tanh', return_sequences=True))
    model.add(tf.keras.layers.LSTM(8, activation='tanh'))

    model.add(tf.keras.layers.Dense(3,activation='softmax'))

    model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy",precision,recall])
    print(model.summary())
    return model
```

```python
X_train = np.array(X_train).reshape((X_train.shape[0], 1, n_steps, n_features))
X_test = np.array(X_test).reshape((X_test.shape[0] ,1, n_steps, n_features))
CONVlstmclassifier = CONVlstm(n_steps,n_features,n_outputs)
history_CONVlstm = CONVlstmclassifier.fit(X_train[:100000], Y_train[:100000],validation_data=(X_test,Y_test), epochs=epochs,
```

Figure: Convolutional LSTM model

5.6 Performance Analysis

The below code shows the performance of the models. It shows the accuracy, loss, precision and recall.

```python
def plot_accuracy(x,history_enc,history_cnn,history_lstm,history_conv):
    plt.figure(figsize=(15,5))
    plt.subplot(121)
    plt.title("Accuracy comparison")
    plt.plot(x,history_enc.history['accuracy'],'*-',label='AutoEncoder')
    plt.plot(x,history_cnn.history['accuracy'],'*-',label='CNN')
    plt.plot(x,history_lstm.history['accuracy'],'*-',label='LSTM')
    plt.plot(x,history_conv.history['accuracy'],'*-',label='CONV-LSTM')
    plt.legend()
    plt.subplot(122)
    plt.title("Val_Accuracy comparison")
    plt.plot(x,history_enc.history['val_accuracy'],'*-',label='AutoEncoder')
    plt.plot(x,history_cnn.history['val_accuracy'],'*-',label='CNN')
    plt.plot(x,history_lstm.history['val_accuracy'],'*-',label='LSTM')
    plt.plot(x,history_conv.history['val_accuracy'],'*-',label='CONV-LSTM')
    plt.legend()
    plt.show()

def plot_loss(x,history_enc,history_cnn,history_lstm,history_conv):
    plt.figure(figsize=(15,5))
    plt.subplot(121)
    plt.title("LOSS comparison")
    plt.plot(x,history_enc.history['loss'],'*-',label='AutoEncoder')
    plt.plot(x,history_cnn.history['loss'],'*-',label='CNN')
    plt.plot(x,history_lstm.history['loss'],'*-',label='LSTM')
    plt.plot(x,history_conv.history['loss'],'*-',label='CONV-LSTM')
    plt.legend()
    plt.subplot(122)
    plt.title("VAL_LOSS comparison")
    plt.plot(x,history_enc.history['val_loss'],'*-',label='AutoEncoder')
    plt.plot(x,history_cnn.history['val_loss'],'*-',label='CNN')
    plt.plot(x,history_lstm.history['val_loss'],'*-',label='LSTM')
    plt.plot(x,history_conv.history['val_loss'],'*-',label='CONV-LSTM')
    plt.legend()
    plt.show()

def plot_precision(x,history_enc,history_cnn,history_lstm,history_conv):
    plt.figure(figsize=(15,5))
    plt.subplot(121)
    plt.title("PRECISION comparison")
    plt.plot(x,history_enc.history['precision'],'*-',label='AutoEncoder')
    plt.plot(x,history_cnn.history['precision'],'*-',label='CNN')
    plt.plot(x,history_lstm.history['precision'],'*-',label='LSTM')
    plt.plot(x,history_conv.history['precision'],'*-',label='CONV-LSTM')
    plt.legend()
    plt.subplot(122)
    plt.title("VAL_PRECISION comparison")
    plt.plot(x,history_enc.history['val_precision'],'*-',label='AutoEncoder')
    plt.plot(x,history_cnn.history['val_precision'],'*-',label='CNN')
    plt.plot(x,history_lstm.history['val_precision'],'*-',label='LSTM')
    plt.plot(x,history_conv.history['val_precision'],'*-',label='CONV-LSTM')
    plt.legend()
    plt.show()

def plot_recall(x,history_enc,history_cnn,history_lstm,history_conv):
    plt.figure(figsize=(15,5))
    plt.subplot(121)
    plt.title("RECALL comparison")
    plt.plot(x,history_enc.history['recall'],'*-',label='AutoEncoder')
    plt.plot(x,history_cnn.history['recall'],'*-',label='CNN')
    plt.plot(x,history_lstm.history['recall'],'*-',label='LSTM')
    plt.plot(x,history_conv.history['recall'],'*-',label='CONV-LSTM')
    plt.legend()
    plt.subplot(122)
    plt.title("VAL_RECALL comparison")
    plt.plot(x,history_enc.history['val_recall'],'*-',label='AutoEncoder')
    plt.plot(x,history_cnn.history['val_recall'],'*-',label='CNN')
    plt.plot(x,history_lstm.history['val_recall'],'*-',label='LSTM')
    plt.plot(x,history_conv.history['val_recall'],'*-',label='CONV-LSTM')
    plt.legend()
    plt.show()
```

Figure: Evaluation metrics

```python
plot_accuracy(x,history_enc,history_cnn,history_LSTM,history_CONVlstm)
plot_loss(x,history_enc,history_cnn,history_LSTM,history_CONVlstm)
plot_precision(x,history_enc,history_cnn,history_LSTM,history_CONVlstm)
plot_recall(x,history_enc,history_cnn,history_LSTM,history_CONVlstm)
```

Figure: Plotting the graphs

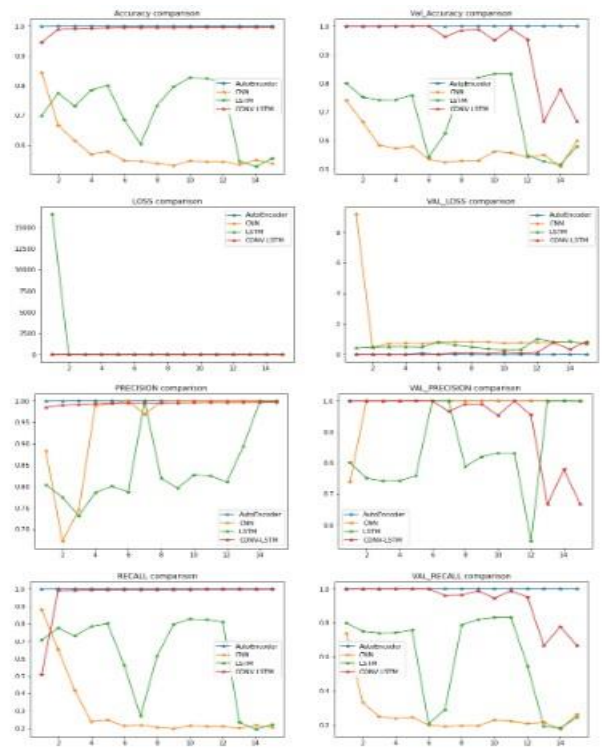

Figure: Performance Results

6 Implementation of NSL-KDD-CUP-99 dataset

6.1 Importing Libraries

Firstly, libraries need to be downloaded into the working environment for better execution of the model. Below figure shows the imported libraries for this dataset.

```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
import sys
import numpy as np
from sklearn import preprocessing
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns
import plotly.express as px
from sklearn.decomposition import PCA
from imblearn.under_sampling import RandomUnderSampler
from sklearn.preprocessing import MinMaxScaler
```

Figure: Importing libraries

6.2 Data Distribution

The below figures show the distribution of data, classification of data, dropping null values and replacing label types.

```
In [9]:  print('Label distribution Data :')
         print(data['label'].value_counts())
```

```
Label distribution Data :
smurf.              701913
neptune.            268254
normal.             243113
satan.               19795
ipsweep.             15587
portsweep.           12978
nmap.                 2885
warezclient.          1281
back.                  563
teardrop.              257
pod.                    68
guess_passwd.           65
buffer_overflow.        35
warezmaster.            25
imap.                   15
rootkit.                12
ftp_write.              10
loadmodule.             10
```

Figure: Data Distribution

```
In [10]: # renaming attack types
label_types = {
    'normal': 'normal',
'back': 'dos',
'buffer_overflow': 'u2r',
'ftp_write': 'r2l',
'guess_passwd': 'r2l',
'imap': 'r2l',
'ipsweep': 'probe',
'land': 'dos',
'loadmodule': 'u2r',
'multihop': 'r2l',
'neptune': 'dos',
'nmap': 'probe',
'perl': 'u2r',
'phf': 'r2l',
'pod': 'dos',
'portsweep': 'probe',
'rootkit': 'u2r',
'satan': 'probe',
'smurf': 'dos',
'spy': 'r2l',
'teardrop': 'dos',
'warezclient': 'r2l',
'warezmaster': 'r2l',
    }

# Dropping null values
data.dropna(inplace=True,axis=1)
#Replacing label types
data['label'] = data.label.apply(lambda r:label_types[r[:-1]])
```

Figure: Classifying the data

6.3 Data Visualization

The below figures show how the data is visualized.

```
In [13]: counts = data.label.value_counts()
px.pie(data,names=counts.index,values = counts.values,title="Label Visualisation")
```

Figure: Label count

```
In [14]: print('Categorical Features in Data :')
for col_name in data.columns:
    if data[col_name].dtypes == 'object' :
        unique_cat = len(data[col_name].unique())
        print("Feature '{col_name}' has {unique_cat} categories".format(col_name=col_name, unique_c

Categorical Features in Data :
Feature 'protocol_type' has 3 categories
Feature 'service' has 69 categories
Feature 'flag' has 11 categories
Feature 'label' has 5 categories
```

Figure: Categorical Features in Data

```
In [15]: print('Distribution of categories in service:')
counts = data['service'].value_counts().head(10)
px.bar(x=counts.index,y = counts.values,color = counts.values,title="service feature Visualisation"
```

Figure: Distribution of Categories in Service

```
In [16]:  print('Distribution of categories in protocol_type:')
          counts = data['protocol_type'].value_counts()
          px.bar(x=counts.index,y = counts.values,color = counts.values,title="protocol_type feature Visualis
```

Figure: Distribution of Categories in protocol type

6.4 Data Pre-processing

The below figures show the data pre-processing.

Data Preprocessing

```
In [11]:  cols_to_drop = []
          # Checking if any feature has just one unique value
          for col in data.columns:
              if len(data[col].value_counts().index) < 2 :
                  cols_to_drop.append(col)

          print("Columns having only 1 unique value : ",cols_to_drop)
          data.drop(cols_to_drop,axis=1,inplace=True)

          Columns having only 1 unique value :  ['num_outbound_cmds', 'is_host_login']
```

Figure: Checking for unique values

```
In [14]:  data.label.value_counts()

Out[14]:  dos       971058
          normal    243113
          probe      51245
          r2l         1412
          u2r           60
          Name: label, dtype: int64
```

Figure: Label value count

```
n [17]:  data.isna().sum()

Out[17]: duration                        0
         protocol_type                   0
         service                         0
         flag                            0
         src_bytes                       0
         dst_bytes                       0
         land                            0
         wrong_fragment                  0
         urgent                          0
         hot                             0
         num_failed_logins               0
         logged_in                       0
         num_compromised                 0
         root_shell                      0
         su_attempted                    0
         num_root                        0
         num_file_creations              0
         num_shells                      0
         num_access_files                0
         is_guest_login                  0
         count                           0
         srv_count                       0
         serror_rate                     0
         srv_serror_rate                 0
         rerror_rate                     0
         srv_rerror_rate                 0
         same_srv_rate                   0
         diff_srv_rate                   0
         srv_diff_host_rate              0
         dst_host_count                  0
         dst_host_srv_count              0
         dst_host_same_srv_rate          0
         dst_host_diff_srv_rate          0
         dst_host_same_src_port_rate     0
         dst_host_srv_diff_host_rate     0
         dst_host_serror_rate            0
         dst_host_srv_serror_rate        0
         dst_host_rerror_rate            0
         dst_host_srv_rerror_rate        0
         label                           0
         dtype: int64
```

Figure: Checking for null values

6.5 Model Comparison

Four models have been used for the implementation of dataset. They are Autoencoder, CNN, LSTM and Conv-LSTM. The below figures show the models implementation.

```
In [29]:
#AutoEncoder Model
def AutoEncoderModel(n_steps,n_features,n_outputs):
    model = tf.keras.models.Sequential(name = 'CNN')

    model.add(tf.keras.layers.Input(shape=(n_steps, n_features)))
    # Encoder Layer 1
    model.add(tf.keras.layers.Conv1D(filters=n_features*4, kernel_size=1, activation='relu'))
    # Encoder Layer 2
    model.add(tf.keras.layers.Conv1D(filters=n_features*2, kernel_size=1, activation='relu'))
    # Encoder Layer 3
    model.add(tf.keras.layers.Conv1D(filters=n_features, kernel_size=1, activation='relu'))

    # BottleNeck
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(n_features,activation='relu'))
    model.add(tf.keras.layers.Reshape((n_steps,n_features)))

    # Decoder Layer 1
    model.add(tf.keras.layers.Conv1DTranspose(filters=n_features,kernel_size=1, activation='relu'))
    # Decoder Layer 2
    model.add(tf.keras.layers.Conv1DTranspose(filters=n_features*2,kernel_size=1, activation='relu'))
    # Decoder Layer 3
    model.add(tf.keras.layers.Conv1DTranspose(filters=n_features*4,kernel_size=1, activation='relu'))

    # Classification Layer
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(n_outputs,"softmax"))

    model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy",precision,recall])
    print(model.summary())

    return model
```

Figure: Autoencoder Model

```
In [30]:
# CNN modle
# define model
def CNNmodel(n_steps,n_features,n_outputs):
    model = tf.keras.models.Sequential(name = 'CNN')

    model.add(tf.keras.layers.Input(shape=(n_steps, n_features)))
    model.add(tf.keras.layers.BatchNormalization())

    model.add(tf.keras.layers.Conv1D(filters=12, kernel_size=1, activation='relu'))
    model.add(tf.keras.layers.Flatten())

    model.add(tf.keras.layers.Dense(128,activation='relu'))
    model.add(tf.keras.layers.Dense(3,"softmax"))

    model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy",precision,recall])
    print(model.summary())
    return model
```

Figure: CNN model

```
In [31]:
#LSTM model
# define model
def LSTMmodel(n_steps,n_features,n_outputs):
    model = tf.keras.models.Sequential(name = 'LSTM')

    model.add(tf.keras.layers.Input(shape=(n_steps, n_features)))
    model.add(tf.keras.layers.BatchNormalization())

    model.add(tf.keras.layers.LSTM(12, activation='relu', return_sequences=True))
    model.add(tf.keras.layers.LSTM(8, activation='relu'))
    model.add(tf.keras.layers.Dense(3,activation='softmax'))

    model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy",precision,recall])
    print(model.summary())
    return model
```

Figure: LSTM model

```
In [32]:  #Conv-Lstm
          # define model
          def CONVlstm(n_steps,n_features,n_outputs):
              model = tf.keras.models.Sequential(name = 'CONV-LSTM')

              model.add(tf.keras.layers.Input(shape=(None,n_steps, n_features)))
              model.add(tf.keras.layers.BatchNormalization())

              model.add(tf.keras.layers.TimeDistributed(tf.keras.layers.Conv1D(filters=12, kernel_size=1, activation='relu')))
              model.add(tf.keras.layers.TimeDistributed(tf.keras.layers.Dropout(0.5)))
              model.add(tf.keras.layers.TimeDistributed(tf.keras.layers.Flatten()))

              model.add(tf.keras.layers.LSTM(12, activation='tanh', return_sequences=True))
              model.add(tf.keras.layers.LSTM(8, activation='tanh'))

              model.add(tf.keras.layers.Dense(3,activation='softmax'))

              model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy",precision,recall])
              print(model.summary())
              return model
```

Figure: Convolutional LSTM model

6.6 Performance Analysis

The below codes show the end results of the data. It shows the accuracy, precision, loss and recall.

Model Comparison

```
def plot_accuracy(x,history_enc,history_cnn,history_lstm,history_conv):
    plt.figure(figsize=(15,5))
    plt.subplot(121)
    plt.title("Accuracy comparison")
    plt.plot(x,history_enc.history['accuracy'],'*-',label='AutoEncoder')
    plt.plot(x,history_cnn.history['accuracy'],'*-',label='CNN')
    plt.plot(x,history_lstm.history['accuracy'],'*-',label='LSTM')
    plt.plot(x,history_conv.history['accuracy'],'*-',label='CONV-LSTM')
    plt.legend()
    plt.subplot(122)
    plt.title("Val_Accuracy comparison")
    plt.plot(x,history_enc.history['val_accuracy'],'*-',label='AutoEncoder')
    plt.plot(x,history_cnn.history['val_accuracy'],'*-',label='CNN')
    plt.plot(x,history_lstm.history['val_accuracy'],'*-',label='LSTM')
    plt.plot(x,history_conv.history['val_accuracy'],'*-',label='CONV-LSTM')
    plt.legend()
    plt.show()

def plot_loss(x,history_enc,history_cnn,history_lstm,history_conv):
    plt.figure(figsize=(15,5))
    plt.subplot(121)
    plt.title("LOSS comparison")
    plt.plot(x,history_enc.history['loss'],'*-',label='AutoEncoder')
    plt.plot(x,history_cnn.history['loss'],label='CNN')
    plt.plot(x,history_lstm.history['loss'],label='RNN')
    plt.plot(x,history_conv.history['loss'],label='CONV-LSTM')
    plt.legend()
    plt.subplot(122)
    plt.title("VAL_LOSS comparison")
    plt.plot(x,history_enc.history['val_loss'],'*-',label='AutoEncoder')
    plt.plot(x,history_cnn.history['val_loss'],label='CNN')
    plt.plot(x,history_lstm.history['val_loss'],label='RNN')
    plt.plot(x,history_conv.history['val_loss'],label='CONV-LSTM')
    plt.legend()
    plt.show()

def plot_precision(x,history_enc,history_cnn,history_lstm,history_conv):
    plt.figure(figsize=(15,5))
    plt.subplot(121)
    plt.title("PRECISION comparison")
    plt.plot(x,history_enc.history['precision'],'*-',label='AutoEncoder')
    plt.plot(x,history_cnn.history['precision'],label='CNN')
    plt.plot(x,history_lstm.history['precision'],label='RNN')
    plt.plot(x,history_conv.history['precision'],label='CONV-LSTM')
    plt.legend()
    plt.subplot(122)
    plt.title("VAL_PRECISION comparison")
    plt.plot(x,history_enc.history['val_precision'],'*-',label='AutoEncoder')
    plt.plot(x,history_cnn.history['val_precision'],label='CNN')
    plt.plot(x,history_lstm.history['val_precision'],label='RNN')
    plt.plot(x,history_conv.history['val_precision'],label='CONV-LSTM')
    plt.legend()
    plt.show()

def plot_recall(x,history_enc,history_cnn,history_lstm,history_conv):
    plt.figure(figsize=(15,5))
    plt.subplot(121)
    plt.title("RECALL comparison")
    plt.plot(x,history_enc.history['recall'],'*-',label='AutoEncoder')
    plt.plot(x,history_cnn.history['recall'],label='CNN')
    plt.plot(x,history_lstm.history['recall'],label='RNN')
    plt.plot(x,history_conv.history['recall'],label='CONV-LSTM')
    plt.legend()
    plt.subplot(122)
    plt.title("VAL_RECALL comparison")
    plt.plot(x,history_enc.history['val_recall'],'*-',label='AutoEncoder')
    plt.plot(x,history_cnn.history['val_recall'],label='CNN')
    plt.plot(x,history_lstm.history['val_recall'],label='RNN')
    plt.plot(x,history_conv.history['val_recall'],label='CONV-LSTM')
    plt.legend()
    plt.show()
```

Figure: Evaluation metrics

```
In [37]: plot_accuracy(x,history_enc,history_cnn,history_LSTM,history_CONV1stm)
         plot_loss(x,history_enc,history_cnn,history_LSTM,history_CONV1stm)
         plot_precision(x,history_enc,history_cnn,history_LSTM,history_CONV1stm)
         plot_recall(x,history_enc,history_cnn,history_LSTM,history_CONV1stm)
```

Figure: Plotting the graphs

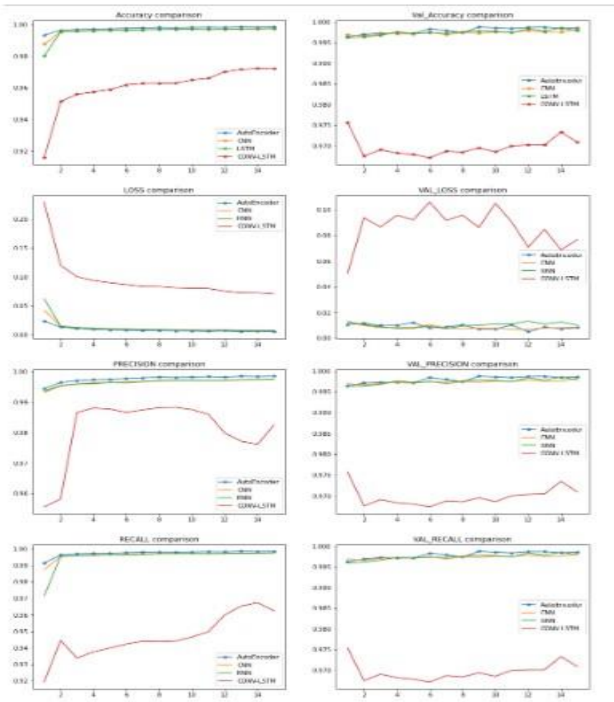

Figure: Performance Results