Configuration Manual

MSc Research Project
MSCCYB1

Saurabh Sanjayprasad Gupta
Student ID: x20224371

School of Computing
National College of Ireland

Supervisor: Prof. Joel Aleburu

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Saurabh Sanjayprasad Gupta...
Student ID:	X20224371..
Programme:	MSCCYB1.. **Year:** 2022-2023
Module:	MSc Research Project..
Lecturer:	Prof. Joel Aleburu..
Submission Due Date:	18TH September 2023..
Project Title:	DoS Attack Detection and Mitigation through Deep Learning Techniques
Word Count:	1611........................... **Page Count:** 11..

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: Saurabh Gupta..

Date: 18TH September 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

Saurabh Sanjayprasad Gupta

Student ID: x20224371

DoS Attack Detection and Mitigation through Deep Learning Techniques

Introduction

The project configuration manual offers a comprehensive outline and concise insights into the academic internship's research endeavours. The study encompassed the practical application and comparative analysis deep learning algorithms, namely SVM and CLSTM. Within this manual, an exhaustive depiction of the necessary configuration prerequisites is provided, encompassing hardware specifications and software prerequisites. The manual further encompasses a comprehensive display of all files, accompanied by their corresponding code and respective functionalities.

Configurations

This consists the physical components and computer programs needed for a system to function properly.

Hardware:

- Operating System: Windows 11
- Processor: Intel i5 9th Gen processor
- Architecture: 64 Bits
- Storage: 2TB SSD
- Memory: 16 GB

Software:

To do this research, we will need to install the following software and modules.

- Anaconda (*Anaconda | The World's Most Popular Data Science Platform*, no date)
- Jupyter Notebook (*Project Jupyter | Home*, no date)
- Python (*Welcome to Python.org*, 2023)
- Pandas (*pandas - Python Data Analysis Library*, no date)
- Seaborn (Waskom, 2021)
- Pickle (*pickle — Python object serialization*, no date)
- Sklearn (*scikit-learn: machine learning in Python — scikit-learn 1.3.0 documentation*, no date)
- TensorFlow (*TensorFlow*, no date)
- Yagmail (Kooten, no date)

Implementation

Step 1: Install the Anaconda and Jupyter Notebook from its official website

Step 2: Install different python modules with following commands

```
pip install pandas
pip install seaborn
pip install pickle
pip install sklearn
pip install tensorflow
pip install yagmail
```

Step 3: Create a 1_Preprocessing_file.ipynb and import modules

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

import pandas as pd
pd.set_option("display.max_columns",None)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from lib_file import lib_path
import pickle
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

import os
for dirname,_,filenames in os.walk("input"):
    for filename in filenames:
        print(os.path.join(dirname,filename))
```

Step 4: Load the original dataset

```
df=pd.read_csv("input/Intrusion_Data.csv")
```

Step 5: Display the first few rows of a DataSet

```
df.head()
```

Step 6: Returned a tuple containing the number of rows and columns present in the DataSet.

```
df.shape
```

Step 7: Will look if there is any null or infinity value in the DataSet.

```
for feature in df.columns.tolist():
    print('{:<30} -> {} null values'.format(feature, df[feature].isnull().sum()),'\n')
```

```
for feature in df.columns.tolist():
    print('{:<30} -> {} infinity values'.format(feature, df[feature].isin([np.inf, -np.inf]).sum()),'\n')
```

Step 8: To find what types of DoS attacks are there from **label** column

```
class_labels=df['Label'].unique().tolist()
class_labels.sort()
print(class_labels)
```

Step 9: To calculate how many attacks are there for each DoS attack type

```
df['Label'].value_counts()
```

Step 10: Analyse label feature of every DoS attack type using chart

```
chart_data=list(dict(df['Label'].value_counts()).values())
chart_labels=list(dict(df['Label'].value_counts()).keys())
with plt.style.context(style='fivethirtyeight'):
    fig,ax=plt.subplots(nrows=1,ncols=1,figsize=(10,5))
    ax.bar(x=chart_labels,
           height=chart_data)
    for p in ax.patches:
        ax.annotate((p.get_height()), (p.get_x()+0.35, p.get_height()+50))
    ax.set_title(label='Analysing label feature')
    ax.set_xlabel(xlabel='class labels')
    ax.set_ylabel(ylabel='number of records')
    plt.show()
```

Step 11: Display concise information about a DataSet. This information includes the number of non-null values in each column, the data types of each column, and memory usage.

```
df.info()
```

Step 12: Create a dictionary that associates each label with its index.

```
class_dict={}
for idx,label in enumerate(class_labels):
    class_dict[label]=idx
print(class_dict)
```

Step 13: Will select a feature from the dataset which can give us the highest accuracy

```
target_feature = 'class'
all_features = df.columns.tolist()
all_features.remove(target_feature)
corr = df[all_features].corrwith(df[target_feature])
sorted_features = corr.abs().sort_values(ascending=False).index.tolist()
selected_features = sorted_features[:20]
filtered_df = df[[target_feature] + selected_features]
```

Step 14: Will display the first few rows of the data from the **filtered.df**

```
filtered_df.head()
```

Step 15: A Min-Max Scaler is applied to normalizie the data in "filtered_df" without the 'class' label. The scaled data is stored in "data," maintaining the 'class' column.

```
scaler=MinMaxScaler()
scaler=scaler.fit(filtered_df.drop(labels='class',axis=1))
scaled_df=scaler.transform(filtered_df.drop(labels='class',axis=1))
data=pd.DataFrame(data=scaled_df,columns=filtered_df.drop(labels='class',axis=1).columns)
data['class']=filtered_df['class'].values
data.head()
```

Step 16: Save the 'scaler' object to a file named 'Scaler.pkl' in the 'models' folder using the 'pickle' module.

```
with open(file='models/Scaler.pkl',mode='wb') as file:
    pickle.dump(obj=scaler,file=file)
```

Step 17: Extract column names, excluding "class". Save list to 'Important_Columns.pkl' using pickle.

```
imp_cols=data.columns.tolist()
imp_cols.remove("class")

print(imp_cols)

with open(file='models/Important_Columns.pkl',mode='wb') as file:
    pickle.dump(obj=imp_cols,file=file)
```

Step 18: Now separate the columns from dataset where X will be all rows and all columns except last columns and y will be all rows and last column.

```
X=data.iloc[:,:-1]
y=data.iloc[:,-1:]
```

Step 19: Display the beginning of data in "X" and “y”

```
X.head()
```

```
y.head()
```

Step 20: Using the SMOTE technique, oversample data for balancing. Transform into a DataSet, shuffle, and displayed a sample from the dataset.

```
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
X_smote, y_smote = oversample.fit_resample(X.values, y.values.ravel())

data=pd.DataFrame(data=X_smote,columns=X.columns)
data['class']=y_smote
data=data.sample(frac=1).reset_index(drop=True)
data.head()
```

Step 21: Now will generate a bar chart to analyse and display the distribution of class labels in a dataset. We will use the matplotlib library to create the chart and includes labels and annotations for each bar, illustrating the count of records for each class.

```
chart_data=list(dict(data['class'].value_counts()).values())
chart_labels=list(dict(data['class'].value_counts()).keys())
chart_labels=[str(item) for item in chart_labels]
with plt.style.context(style='ggplot'):
    fig,ax=plt.subplots(nrows=1,ncols=1,figsize=(10,5))
    ax.bar(x=chart_labels,
           height=chart_data)
    for p in ax.patches:
        ax.annotate((p.get_height()), (p.get_x()+0.35, p.get_height()+50))
    ax.set_title(label='Analysing label feature')
    ax.set_xlabel(xlabel='class labels')
    ax.set_ylabel(ylabel='number of records')
    plt.show()
```

Step 22: Retrieving the shape of data

```
data.shape
```

Step 23: Now separate the columns from dataset where X will be all rows and all columns except last columns and y will be all rows and last column.

```
X=data.iloc[:,:-1]
y=data.iloc[:,-1:]
```

Step 24: Display the beginning of data in "X" and “y”

```
X.head()
```

```
y.head()
```

Step 25: Now we will split dataset into training and testing sets using the train_test_split function. It ensures a random yet consistent division for evaluation. The shapes of the resulting sets are then printed.

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42,shuffle=True,stratify=y)
print(X_train.shape,X_test.shape,y_train.shape,y_test.shape)
```

Step 26: Now will save the training and testing data along with their labels in separate CSV files within a folder named "splitted_data", excluding the index column.

```
X_train.to_csv("splitted_data/X_train.csv",index=False)
X_test.to_csv("splitted_data/X_test.csv",index=False)
y_train.to_csv("splitted_data/y_train.csv",index=False)
y_test.to_csv("splitted_data/y_test.csv",index=False)
```

Step 27: Now we create a new file called 2_Training_File and will import the modules.

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

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
plt.rcParams["font.size"]=15
pd.set_option("display.max_columns",None)
from lib_file import lib_path
import tensorflow
from tensorflow.keras.utils import to_categorical
import pickle

import os
for dirname,_,filenames in os.walk('splitted_data'):
    for filename in filenames:
        print(os.path.join(dirname,filename))
```

Step 28: Will load training and testing data using CSV files: X_train, X_test, y_train, y_test from the 'splitted_data' folder.

```
X_train=pd.read_csv('splitted_data/X_train.csv')
X_test=pd.read_csv('splitted_data/X_test.csv')
y_train=pd.read_csv('splitted_data/y_train.csv')
y_test=pd.read_csv('splitted_data/y_test.csv')
```

Step 29: Display the beginning of data in "X_train", “X_test”, “y_train” and “y_test”.

```
X_train.head()
```

```
X_test.head()
```

```
y_train.head()
```

```
y_test.head()
```

Step 30: Now we will perform the SupportVectorMachine algorithm.

Now we will use SVM model from the scikit-learn library, fitting it with training data. It will measure the time taken for execution.

```
%%time

from sklearn.svm import SVC
SVC_model=SVC(max_iter=500)
SVC_model=SVC_model.fit(X_train.values, y_train.values.ravel())
```

We will measure the time taken, it will use trained SVC model to predict values from test data, and prints the predictions as a list.

```
%%time

SVC_pred=SVC_model.predict(X_test.values)
print(SVC_pred.tolist())
```

Now will convert true labels from y_test into a list and will print them.

```
true_labels=y_test.values.ravel()
print(true_labels.tolist())
```

Step 31: Result analysis of SVM

Will print the labels.

```
class_labels=['Benign', 'MSSQL', 'Syn', 'UDP']
print(class_labels)
```

Now will calculates and display the validation accuracy (in percentage) of the Support Vector Classifier model using true labels and its predictions.

```
SVC_accuracy=accuracy_score(y_true=true_labels,y_pred=SVC_pred)
print(f"Validation accuracy of SupportVectorClassifier is {SVC_accuracy*100.0:.2f}%")
```

Will generate a report comparing predicted and true labels using the Support Vector Classifier (SVC), and print it. The report includes precision, recall, and F1-score for each class.

```
print(classification_report(y_true=true_labels,y_pred=SVC_pred,target_names=class_labels))
```

Displayed a heatmap using Seaborn and Matplotlib to show a confusion matrix from predicted and true labels using Support Vector Classifier, with class labels and counts.

```
plt.figure(figsize=(5,5))
plt.rcParams['font.size']=15
ax=sns.heatmap(data=confusion_matrix(y_true=true_labels,y_pred=SVC_pred),
               xticklabels=class_labels,
               yticklabels=class_labels,
               annot=True,
               fmt='4d',
               cmap=plt.cm.Blues,
               cbar=False,
               linecolor='black',
               linewidths=5)
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.title(label='SupportVectorClassifier confusion matrix')
plt.show()
```

Will save the 'SVC_model' using pickle to the 'SupportVectorClassifier.pkl' file located in the 'models' directory.

```
with open(file='models/SupportVectorClassifier.pkl',mode='wb') as file:
    pickle.dump(obj=SVC_model,file=file)
```

Step 32: Now we will perform the ConvolutionalLongShortTermMemory algorithm.

Will convert training and testing labels into categorical format using "to_categorical" function and display the first 5 entries of transformed training and testing labels.

```
y_train=to_categorical(y_train.values)
y_test=to_categorical(y_test.values)

print(y_train[:5])
print(y_test[:5])
```

Now we will reshape training and testing data into a specific format, adding a dimension and then prints the new shapes of the transformed datasets.

```
x_train=np.reshape(a=X_train.values,newshape=(X_train.shape[0],X_train.shape[1],1))
x_test=np.reshape(a=X_test.values,newshape=(X_test.shape[0],X_test.shape[1],1))

print(x_train.shape)
print(x_test.shape)
```

Willimport TensorFlow and its components: Sequential, Embedding, Conv1D, BatchNormalization, MaxPool1D, Bidirectional, LSTM, Input, Flatten, Dropout, Dense for creating an application.

```
import tensorflow
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, BatchNormalization, MaxPool1D, Bidirectional, LSTM
from tensorflow.keras.layers import Input, Flatten, Dropout, Dense
```

Will set up a neural network model using Keras with various convolutional and bidirectional LSTM layers for sequence data classification. The model includes several convolutional layers with max pooling, followed by LSTM layers, dropout, and dense layers. It's compiled for categorical cross-entropy loss optimization using the Adam optimizer.

```
model = Sequential()
model.add(Input(shape=(x_train.shape[1], x_train.shape[2])))
model.add(Conv1D(filters=64,kernel_size=3,activation='relu',padding='same'))
model.add(Conv1D(filters=64,kernel_size=3,activation='relu',padding='same'))
model.add(MaxPool1D(pool_size=2))
model.add(Conv1D(filters=128,kernel_size=3,activation='relu',padding='same'))
model.add(Conv1D(filters=128,kernel_size=3,activation='relu',padding='same'))
model.add(MaxPool1D())
model.add(Conv1D(filters=256,kernel_size=3,activation='relu',padding='same'))
model.add(Conv1D(filters=256,kernel_size=3,activation='relu',padding='same'))
model.add(MaxPool1D())
model.add(Conv1D(filters=512,kernel_size=3,activation='relu',padding='same'))
model.add(Conv1D(filters=512,kernel_size=3,activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPool1D())
model.add(Bidirectional(LSTM(units=200,return_sequences=True,recurrent_dropout=0,activation='tanh',recurrent_activation='sigmoid',unroll=False,use_bias=True)))
model.add(Bidirectional(LSTM(units=200,return_sequences=True,recurrent_dropout=0,activation='tanh',recurrent_activation='sigmoid',unroll=False,use_bias=True)))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(512, activation='relu'))
model.add(Dense(4, activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer="adam",metrics=['accuracy'])
```

Will print the summary of the model's configuration and layers.

```
model.summary()
```

Now will train a model for 20 epochs using training data, with batch size 64, and evaluates on validation data.

```
history=model.fit(x=x_train,y=y_train,epochs=20,batch_size=64,validation_data=(x_test,y_test))
```

We will use the 'fivethirtyeight' style to create a side-by-side plot with training and validation accuracy on the left and training and validation loss on the right. It also visualizes the progression of these metrics over epochs.

```
with plt.style.context(style='fivethirtyeight'):
    plt.figure(figsize=(18,8))
    plt.rcParams['font.size']=15
    plt.subplot(121)
    plt.plot(history.history['accuracy'],label='training accuracy')
    plt.plot(history.history['val_accuracy'],label='validation accuracy')
    plt.title(label='accuracy plots')
    plt.xlabel(xlabel='Epochs')
    plt.ylabel(ylabel='Accuracy')
    plt.legend()
    plt.subplot(122)
    plt.plot(history.history['loss'],label='training loss')
    plt.plot(history.history['val_loss'],label='validation loss')
    plt.title(label='loss plots')
    plt.xlabel(xlabel='Epochs')
    plt.ylabel(ylabel='Loss')
    plt.legend()
    plt.show()
```

Now will generate predictions for the model using the test data and specified batch size also displayed the resulting probabilities using the 'model_probabilities' variable.

```
model_probabilities=model.predict(x_test,batch_size=64,verbose=1)
print(model_probabilities)
```

We will now calculate the predicted labels by selecting the highest probability index from model probabilities, then prints and lists those labels.

```
predicted_labels=np.argmax(model_probabilities,axis=1)
print(list(predicted_labels))
```

Retrieving the highest-value labels from the y_test data, converts them into a list, and prints the list of true labels.

```
true_labels=np.argmax(y_test,axis=1)
print(list(true_labels))
```

Step 33: Result Analysis of CLSTM

Now will calculates and display the validation accuracy (in percentage) of the CLSTM model using true labels and its predictions.

```
CLSTM_accuracy=accuracy_score(y_true=true_labels,y_pred=predicted_labels)
print(f"The validation accuracy of ConvolutionalLongShort-TermMemory model is {CLSTM_accuracy*100.0:.2f}%")
```

Will generate a report comparing predicted and true labels using the CLSTM, and print it. The report includes precision, recall, and F1-score for each class.

```
print(classification_report(y_true=true_labels,y_pred=predicted_labels,target_names=class_labels))
```

Displayed a heatmap using Seaborn and Matplotlib to show a confusion matrix from predicted and true labels using CLSTM, with class labels and counts.

```
plt.figure(figsize=(5,5))
plt.rcParams['font.size']=15
ax=sns.heatmap(data=confusion_matrix(y_true=true_labels,y_pred=predicted_labels),
               xticklabels=class_labels,
               yticklabels=class_labels,
               annot=True,
               fmt='4d',
               cmap=plt.cm.Blues,
               cbar=False,
               linecolor='black',
               linewidths=5)
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.title(label='CLSTM confusion matrix')
plt.show()
```

Will save the 'CLSTM model' using pickle to the 'ConvolutionalLongShortTermMemory_model.h5' file located in the 'models' directory.

```
model.save(filepath="models/ConvolutionalLongShortTermMemory_model.h5")
```

Step 34: Now we will compare the accuracy score of both SVC and CLSTM model

```
x=['SVM','CLSTM']
y=[SVC_accuracy,CLSTM_accuracy]

with plt.style.context(style="fivethirtyeight"):
    plt.figure(figsize=(5, 5))
    plt.bar(height=y, x=x)
    for i in range(len(x)):
        plt.annotate(f"{y[i]:.4f}", (x[i], y[i]), ha="center", va="bottom")
    plt.title("Accuracy Comparison of Used Models",fontsize=25)
    plt.xlabel("Algorithms")
    plt.ylabel("Accuracy")
    plt.xticks(rotation=90)
    plt.show()
```

Step 35: Will create a new file called “3_Testing_file.ipynb” and import the modules

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

import pandas as pd
pd.set_option("display.max_columns",None)
import numpy as np
from IPython.core.display import display, HTML
import pickle
import yagmail
from tensorflow.keras.models import load_model
```

Step 36: Loading a CLSTM model from the specified file path, configuring it without compilation, then set up its loss, optimizer, and accuracy metrics for training.

```
model=load_model(filepath="models/ConvolutionalLongShortTermMemory_model.h5",compile=False)
model.compile(loss='categorical_crossentropy',optimizer="adam",metrics=['accuracy'])
```

Step 37: Loading the 'Scaler.pkl' file from the 'models' directory using the 'rb' mode and store its content in the 'scaler' variable using the 'pickle.load()' function.

```
with open(file="models/Scaler.pkl",mode="rb") as file:
    scaler=pickle.load(file=file)
```

Step 38: Load the 'Important_Columns.pkl' file from the 'models' directory using the 'rb' mode and store its content in the 'imp_cols' variable using the 'pickle.load()' function.

```
with open(file="models/Important_Columns.pkl",mode="rb") as file:
    imp_cols=pickle.load(file=file)
```

Will print the imp_cols

```
print(imp_cols)
```

Step 39: Now we will specify the class labels

```
class_labels=['Benign', 'MSSQL', 'Syn', 'UDP']
```

Step 40: Read data from "user_input/file_19.csv" using Pandas and display the first few rows of the dataset. (Can read data from any file between file_0.csv and file_19.csv

```
input_=pd.read_csv("user_input/file_19.csv")
input_.head()
```

Step 41: Will select important columns from the input data, then scale and transform them using a specified scaler also create a DataSet with the scaled data and display the first few rows.

```
input_=input_[imp_cols]
inp_cols=input_.columns
scaled_input_=scaler.transform(input_)
input_df=pd.DataFrame(data=scaled_input_,columns=inp_cols)
input_df.head()
```

Step 42: Now we will predict the class of an input using the model. If the predicted class is not "Benign," it sends an email alert. Then displayed the predicted class and label using a Convolutional Long Short-Term Memory model. This process helps detect intrusions and notify users.

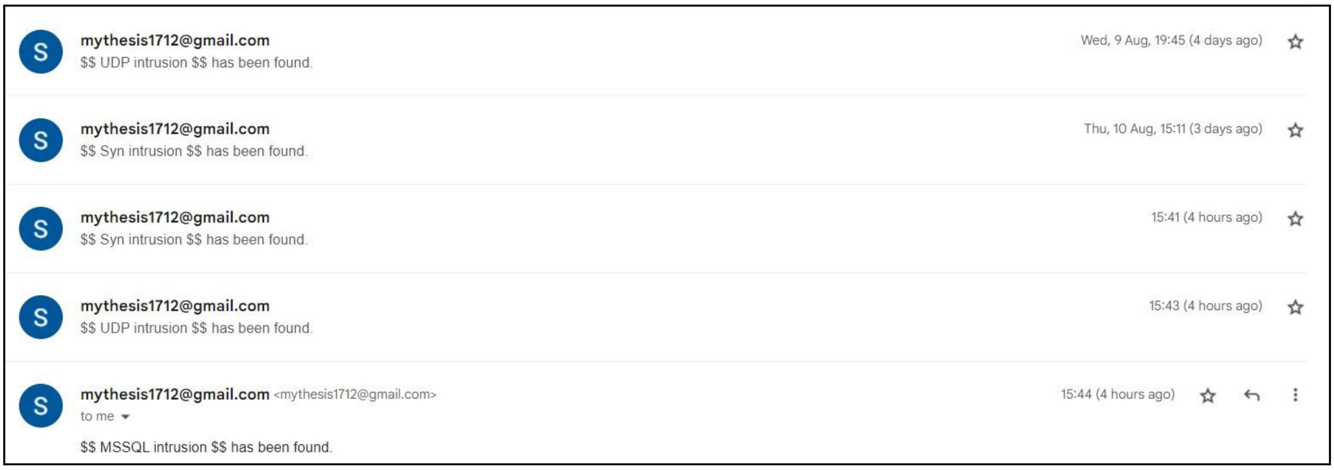
```
model_pred=model.predict(input_df.values)

model_pred_class=np.argmax(model_pred[0])

model_pred_label=class_labels[model_pred_class]

if model_pred_label!='Benign':
    user = yagmail.SMTP(user='mythesis1712@gmail.com',password='rmuxqrevkmtqmcgx')
    user.send(to='mythesis1712@gmail.com',subject='$$ALERT$$',contents=f"$$ {model_pred_label}  intrusion $$ has been found.")
    print("Email sent sucessfully")
display("ConvolutionalLongShortTermMemory model predicted class is:",HTML(str(model_pred_class)))
display("ConvolutionalLongShortTermMemory model predicted label is:",HTML(str(model_pred_label)))
```

As you see, I got a mail for all attack types except Bening



References:

Anaconda | The World's Most Popular Data Science Platform (no date). Available at: https://www.anaconda.com/ (Accessed: 14 August 2023).

Kooten, P. van (no date) 'yagmail: Yet Another GMAIL client'. Available at: https://github.com/kootenpv/yagmail (Accessed: 13 August 2023).

pandas - Python Data Analysis Library (no date). Available at: https://pandas.pydata.org/ (Accessed: 13 August 2023).

pickle — Python object serialization (no date) *Python documentation*. Available at: https://docs.python.org/3/library/pickle.html (Accessed: 13 August 2023).

Project Jupyter | Home (no date). Available at: https://jupyter.org/ (Accessed: 14 August 2023).

scikit-learn: machine learning in Python — scikit-learn 1.3.0 documentation (no date). Available at: https://scikit-learn.org/stable/ (Accessed: 13 August 2023).

TensorFlow (no date) *TensorFlow*. Available at: https://www.tensorflow.org/ (Accessed: 13 August 2023).

Waskom, M. (2021) 'seaborn: statistical data visualization', *Journal of Open Source Software*, 6(60), p. 3021. Available at: https://doi.org/10.21105/joss.03021.

Welcome to Python.org (2023) *Python.org*. Available at: https://www.python.org/ (Accessed: 13 August 2023).