National College of Ireland

Configuration Manual

Network Intrusion Detection System using CNN-LSTM Hybrid Network

MSc Research Project
MSc in Cybersecurity

Ajeeser Kokkali
Student ID: 20112491

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Ajeeser Kokkali

Student ID: 20112491

Programme: MSc in Cybersecurity **Year:** 2022

Module: Research Project

Lecturer: Vikas Sahni

Submission Due Date: 26/04/2022

Project Title: Network Intrusion Detection System using CNN-LSTM Hybrid Network

Word Count: ……1028…… **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: ……Ajeeser Kokkali……

Date: ……24/04/2022……

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

Ajeeser Kokkali
20112491

Introduction

The purpose of this manual is to lay out the procedures for carrying out the research project as well as the setup of the equipment used to construct and operate the models. The phases entail downloading and installing the necessary software and packages, as well as the minimal configuration necessary for the project to function properly.

Software:

Python ver. 3.9
Google Colaboratory
Jupyter Notebook
TensorFlow
Anaconda

Hardware:

The following is the hardware setup of the machine on which I completed my project.
OS: Windows 11.
Processor: Intel I7
RAM: 16GB
Storage (SSD): 1TB.

Intrusion detection using CNN-LSTM on NSL-KDD dataset

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

The dataset and model are located in the /content/drive/ mount point directory.

Importing the libraries:

As illustrated in Figure 1, pandas and numpy libraries were imported for data pre-processing, as well as the 'matplotlib' and ‘seaborn’ libraries for visualization and animation. Finally, the sklearn library was imported which is a valuable package for machine learning in Python.

MLPClassifier stands for Multi-layer Perceptron Classifier imported from sklearn neural network library is linked to a Neural Network by its name. Unlike other classification methods such as Support Vectors or Naive Bayes Classifier, MLPClassifier does classification using an underlying Neural Network. The train_test_split imported from

sklearn model selection is utilized to measure their performance when machine learning techniques are employed to make predictions on data that was not used to train the model.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from joblib import dump, load
from sklearn.metrics import accuracy_score, f1_score, precision_score,recall_score
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from scipy import interp
from itertools import cycle
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
import sklearn.metrics as metrics
```

Figure 1: Scikit libraries

All of these libraries were brought in to aid in the development of deep learning architecture. To develop the model, a variety of library functions were used. The sequence function was imported from the keras.preprocessing library for listing sequences, and the 'Sequential' function was loaded from the keras.models library for starting the CNN model. The keras.layers library was used to import the Dense, Dropout, Activation, and Embedding methods.

```
import pandas as pd
import numpy as np
import sys
import keras
import sklearn
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Embedding
from keras.layers import LSTM, SimpleRNN, GRU, Bidirectional, BatchNormalization,Convolution1D,MaxPooling1D, Reshape, GlobalAveragePooling1D,Flatten
from keras.utils import to_categorical
import sklearn.preprocessing
from sklearn import metrics
from scipy.stats import zscore
from tensorflow.keras.utils import get_file, plot_model
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
print(pd.__version__)
print(np.__version__)
print(sys.version)
print(sklearn.__version__)
```

Figure 2: Keras libraries

Load the dataset

Here, we used the pandas (pd) function 'pd.read csv and then dataset filename' to load the NSL-KDD dataset. So this is the complete dataset.

```
#Loading training set into dataframe
df = pd.read_csv('/content/drive/My Drive/NSL_KDD/NSL_KDD_dataset/KDDTrain.txt', header=None)
df.head()
```

Figure 3: Dataset loading

One-hot encoding

For category variables with no such ordinal relationship, the integer encoding is insufficient. In fact, allowing the model to assume a natural ordering across categories and then encoding it could lead to poor performance or unexpected results (predictions halfway between categories). In this case, a one-hot encoding can be used to encode the integer representation. The integer encoded variable is removed and a new binary variable is added for each unique integer value.

```
#One-hot encoding
def one_hot(df, cols):
    """
    @param df pandas DataFrame
    @param cols a list of columns to encode
    @return a DataFrame with one-hot encoding
    """
    for each in cols:
        dummies = pd.get_dummies(df[each], prefix=each, drop_first=False)
        df = pd.concat([df, dummies], axis=1)
        df = df.drop(each, 1)
    return df
```

Figure 4: One-hot encoding

Normalize

In machine learning, normalization is the process of converting data into the range [0, 1] (or any other range) or simply onto the unit sphere. If one of the variables is in the 1000s and the other is in the 0.1s, the distance will be dominated by the first variable. Normalization and standardization may be advantageous in this situation.

```
#Function to min-max normalize
def normalize(df, cols):
    """
    @param df pandas DataFrame
    @param cols a list of columns to encode
    @return a DataFrame with normalized specified features
    """
    result = df.copy() # do not touch the original df
    for feature_name in cols:
        max_value = df[feature_name].max()
        min_value = df[feature_name].min()
        if max_value > min_value:
            result[feature_name] = (df[feature_name] - min_value) / (max_value - min_value)
    return result
```

Figure 5: Normalizing

CNN-LSTM model

LSTM output size=128 (As a result, each of the 128 outputs will be used as an input layer. As a result, the first iteration will take 128 and the second iteration will take 128 and convert it into another input for the Neural network. So, for a better outcome and accuracy, we're going to combine that LSTM with CNN. As you can see, 64 units, 5 hidden layers, and padding are all the same. The 'relu' activation function is utilized. Here, a convolution neural network is employed, followed by two maxpooling layers, an LSTM layer, and Dropout. We can guess whether the answer is yes or no based on this. So, in the end, activation = SoftMax is used.

```
#CNN+LSTM
batch_size = 128
model = Sequential()
model.add(Convolution1D(64, kernel_size=122, border_mode="same",activation="relu",input_shape=(122, 1)))
model.add(MaxPooling1D(pool_length=(5)))
model.add(BatchNormalization())
model.add(Bidirectional(LSTM(64, return_sequences=False)))
model.add(Reshape((128, 1), input_shape = (128, )))

model.add(MaxPooling1D(pool_length=(5)))
model.add(BatchNormalization())
model.add(Bidirectional(LSTM(128, return_sequences=False)))

model.add(Dropout(0.5))
model.add(Dense(5))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

Figure 6: CNN-LSTM model

Model fitting

"Cnn.fit" is loaded in the model fitting. so that the exact numbers of loss, accuracy, and validation accuracy can be seen in each iteration The iteration will be terminated if the accuracy does not improve. So, at the end, I set epoch to 2 to ensure that it runs for two iterations. So I saved and loaded the model at this point.

```
history = model.fit(x_train_1, y_train_1,validation_data=(x_test_2,y_test_2), epochs=2)
```

Performance metrics

The accuracy result was obtained using the function 'accuracy score,' which I imported from the 'sklearn.metrics' library, but I also imported several additional functions, such as classification report and confusion matrix, and printed each of the three outcomes separately. The same scikit learn library can be used to acquire the confusion matrix, thus imported the confusion matrix function from the sklearn.metrics library.

```
pred = model.predict(x_test_2)
pred = np.argmax(pred,axis=1)
y_eval = np.argmax(y_test_2,axis=1)
score = metrics.accuracy_score(y_eval, pred)
oos_pred.append(score)
print("Validation score: {}".format(score))


cnf_matrix =confusion_matrix(y_eval, pred)
FP = cnf_matrix.sum(axis=0) - np.diag(cnf_matrix)
FN = cnf_matrix.sum(axis=1) - np.diag(cnf_matrix)
TP = np.diag(cnf_matrix)
TN = cnf_matrix.sum() - (FP + FN + TP)
specificity = TP/(TP+FN)
sensitive = TN/(TN+FP)
print("CNN+LSTM:Specificity:" + str(specificity[0]))
print("CNN+LSTM:Sensitivity:" + str(sensitive[0]))
```

Figure 7: Validation score

```
#confusion Matrix
matrix =confusion_matrix(y_eval, pred)
class_names=['DoS','Normal','Probe','R2L','U2R']
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()

target_names = ['DoS','Normal','Probe','R2L','U2R']
print(classification_report(y_eval, pred, target_names=target_names))
```

Figure 8: Confusion matrix

```
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
#Train and validation accuracy
plt.plot(epochs, acc, 'y', label='Training accurarcy')
plt.plot(epochs, val_acc, 'g', label='Validation accurarcy')
plt.title('Training and Validation accurarcy')
plt.legend()

plt.figure()
#Train and validation loss
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'g', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```

Figure 9: Train and validation accuracy and loss