CNN-LSTM: Network Intrusion Detection System

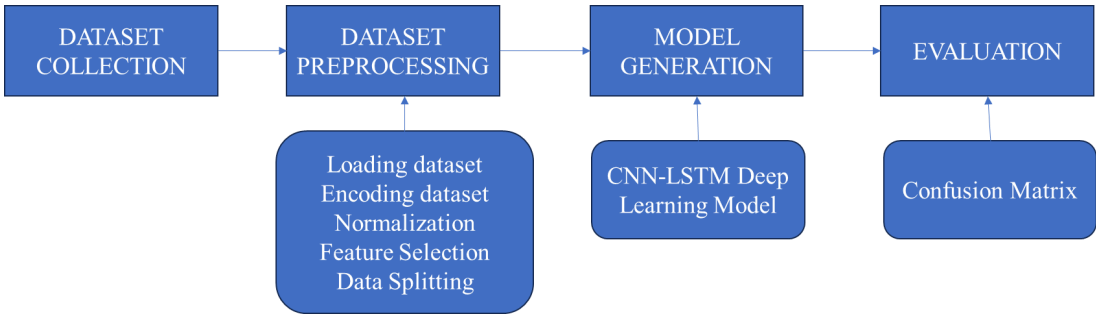
1.Overview

The CNN-LSTM model combines the strengths of Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks to effectively handle data with spatial and temporal dependencies. CNNs are well-suited for feature extraction from structured data like images, while LSTM networks are designed to capture temporal relationships in sequential data.

Dataset:

The UNSW-NB15 dataset is well known for its detailed coverage of contemporary network traffic, encompassing both legitimate and harmful actions. It includes nine different forms of attacks including DoS, Exploits, Fuzzers, Backdoor, and Analysis. The data was gathered from actual traffic sources, making it very suitable for assessing the IDS effectiveness.

2. Model Architecture:



The architecture of the CNN-LSTM model used in this project consists of:

- **Convolutional Layers:** These layers perform feature extraction by applying convolutional filters to the input data, capturing spatial features and patterns.
- **Pooling Layers:** Max-pooling or average-pooling layers reduce the dimensionality of the data, making the model more computationally efficient.
- **LSTM Layers:** The extracted features from the CNN are fed into LSTM layers, which process the sequential dependencies and learn temporal patterns.
- **Dense Layers:** After the LSTM layers, dense (fully connected) layers are used to make predictions.

3. Training and Optimization

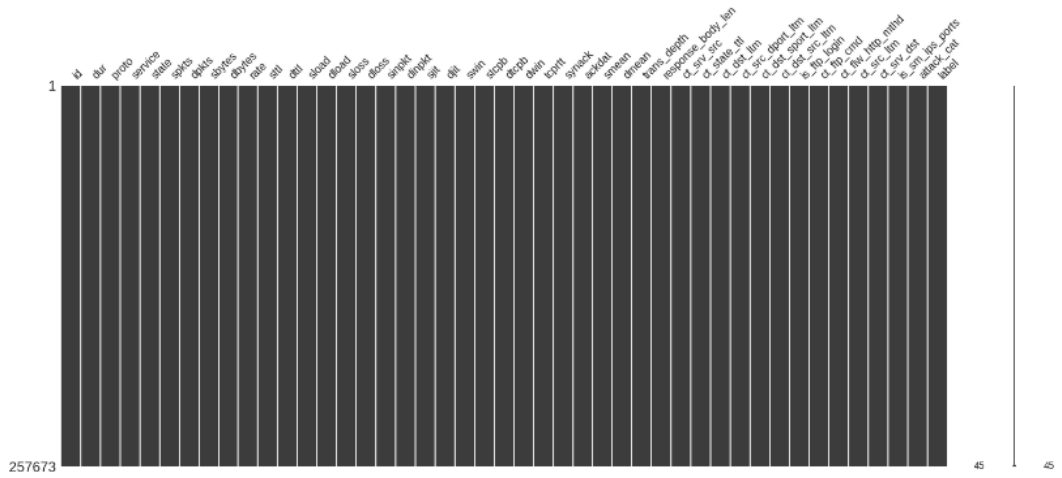
The model is trained using a dataset where features are fed into the CNN layers followed by LSTM for sequence processing. Techniques like early stopping and learning rate scheduling are used to avoid overfitting and optimize the model training process.

4. RESULTS & DISCUSSION

LOADING DATASET ON KAGGLE:

```
/kaggle/input/unsw-nb15/UNSW-NB15_2.csv
/kaggle/input/unsw-nb15/NUSW-NB15_features.csv
/kaggle/input/unsw-nb15/UNSW-NB15_1.csv
/kaggle/input/unsw-nb15/UNSW_NB15_testing-set.csv
/kaggle/input/unsw-nb15/UNSW-NB15_3.csv
/kaggle/input/unsw-nb15/UNSW_NB15_training-set.csv
/kaggle/input/unsw-nb15/UNSW-NB15_LIST_EVENTS.csv
/kaggle/input/unsw-nb15/UNSW-NB15_4.csv
```

MISSING VALUES:



Data is clean and there are no missing values.

PREPROCESSING:

Name	srcip	sport	dstip	dsport	proto	state	dur	sbytes	dbytes	sttl	...	ct_ftp_cmd	ct_srv_src	ct_srv_dst	ct_dst_ltm	ct_src_ltm	ct_src_dport_ltm	ct_dst_spor
0	59.166.0.0	1390	149.171.126.6	53	udp	CON	0.001055	132	164	31	...	0	3	7	1	3	1	
1	59.166.0.0	33661	149.171.126.9	1024	udp	CON	0.036133	528	304	31	...	0	2	4	2	3	1	
2	59.166.0.6	1464	149.171.126.7	53	udp	CON	0.001119	146	178	31	...	0	12	8	1	2	2	
3	59.166.0.5	3593	149.171.126.5	53	udp	CON	0.001209	132	164	31	...	0	6	9	1	1	1	
4	59.166.0.3	49664	149.171.126.0	53	udp	CON	0.001169	146	178	31	...	0	7	9	1	1	1	

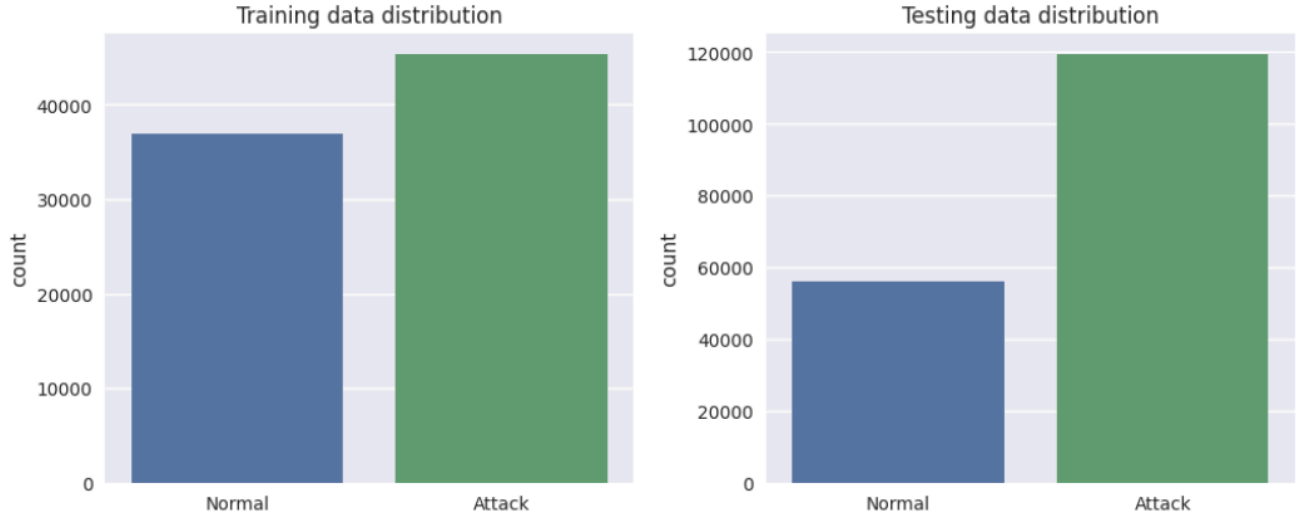
5 rows × 49 columns

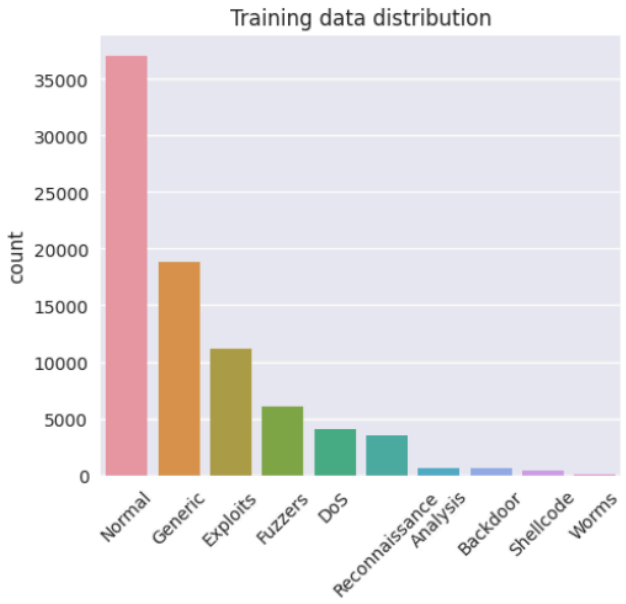
Name	service	ct_flw_http_mthd	is_ftp_login	ct_ftp_cmd	attack_cat	Label
0	dns	0.0	0.0	0	normal	0
1	None	0.0	0.0	0	normal	0
2	dns	0.0	0.0	0	normal	0
3	dns	0.0	0.0	0	normal	0
4	dns	0.0	0.0	0	normal	0
...	...	...	...	...	...	...
2540042	ftp-data	0.0	0.0	0	normal	0
2540043	ftp	0.0	1.0	2	normal	0
2540044	ftp	0.0	1.0	2	normal	0
2540045	http	2.0	0.0	0	normal	0
2540046	pop3	0.0	0.0	0	exploits	1

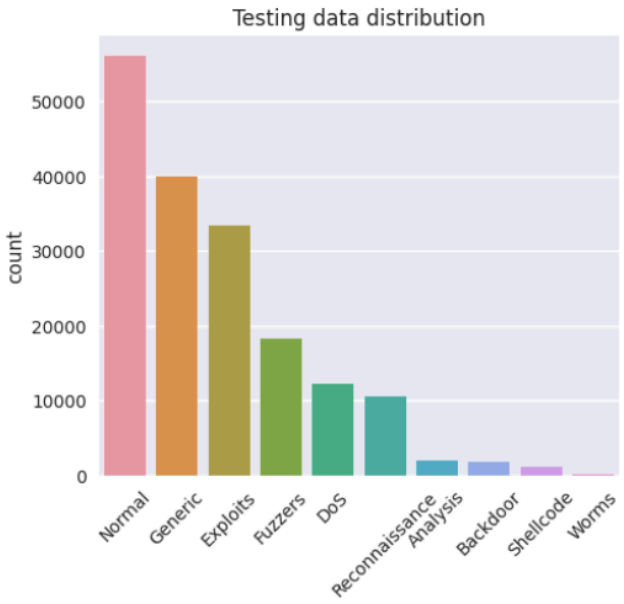
2540047 rows × 6 columns

TRAINING AND TESTING DATASET:

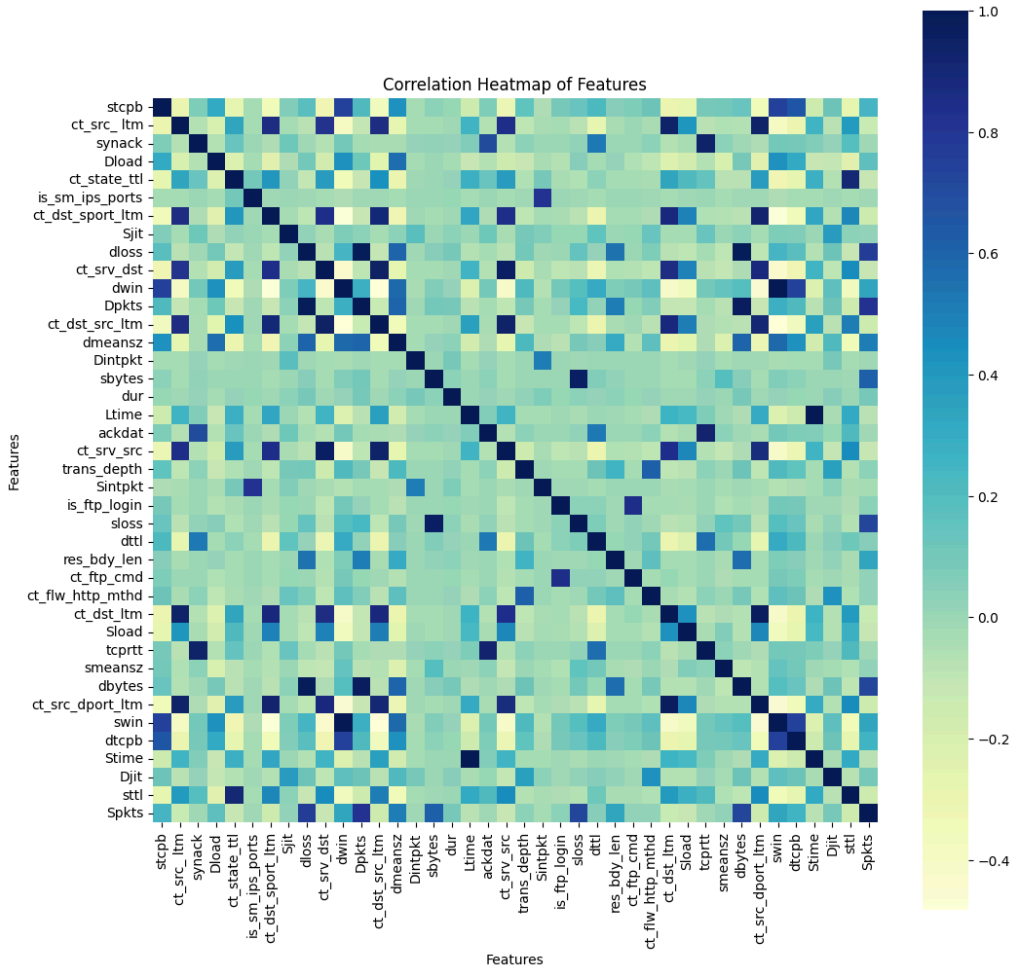
```
Train data length: 82332
Test data length: 175341
```







CORRELATION HEATMAP OF FEATURES:



TRAIN TEST SPLIT:

```
train, test = train_test_split(combined_data,test_size=0.2,random_state=16)
train, val = train_test_split(train,test_size=0.2,random_state=16)
train.shape
```

```
(1625629, 44)
```

```
test.shape
```

```
(508010, 44)
```

```
x_train, y_train = train.drop(columns=['attack_cat']), train[['attack_cat']]
x_test, y_test = test.drop(columns=['attack_cat']), test[['attack_cat']]
x_val, y_val = val.drop(columns=['attack_cat']), val[['attack_cat']]
x_train.shape, y_train.shape
```

```
((1625629, 43), (1625629, 1))
```

```
x_test.shape, y_test.shape
```

```
((508010, 43), (508010, 1))
```

```
x_val.shape, y_val.shape
```

```
((406408, 43), (406408, 1))
```

```
y_train.columns
```

```
Index(['attack_cat'], dtype='object', name='Name')
```

```
attacks = y_train['attack_cat'].unique()
attacks
```

```
array(['normal', 'generic', 'fuzzers', 'exploits', 'dos',
       'reconnaissance', 'backdoor', 'analysis', 'shellcode', 'worms'],
      dtype=object)
```

```
y_train.shape
```

```
(1625629, 10)
```

```
x_train.shape
```

```
(1625629, 204)
```

```
x_test.shape
```

```
(508010, 204)
```

TRAIN DATA:

Name	proto	state	dur	sbytes	dbytes	sttl	dttl	sloss	dloss	service	...	is_ftp_login	ct_ftp_cmd	ct_srv_src	ct_srv_dst	ct_dst_ltm	ct_src_ltm	ct_src_dport_ltm	ct_dst_sp
1070804	tcp	FIN	0.438857	4776	3080	31	29	7	7	None	...	0.0	0	6	3	4	5	1	
1054569	tcp	CON	0.026163	2230	13900	31	29	7	10	None	...	0.0	0	2	7	2	1	1	
1548299	tcp	FIN	0.014954	2854	26584	31	29	7	16	None	...	0.0	0	9	6	4	8	1	
2305760	udp	INT	0.000009	114	0	254	0	0	0	dns	...	0.0	0	33	33	17	17	17	
1989356	udp	CON	0.001562	544	304	31	29	0	0	None	...	0.0	0	6	7	5	6	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
358801	tcp	FIN	0.046476	320	1890	31	29	1	2	ftp-data	...	0.0	0	3	5	3	7	1	
556159	tcp	FIN	0.027755	5928	8010	31	29	14	17	ssh	...	0.0	0	1	1	8	3	1	
1408939	udp	INT	0.000003	264	0	60	0	0	0	dns	...	0.0	0	39	39	18	18	18	
1894282	tcp	FIN	0.057911	2766	27392	31	29	7	16	None	...	0.0	0	2	2	3	4	1	
2148776	tcp	FIN	0.263385	1920	4416	31	29	6	6	None	...	0.0	0	9	4	6	9	2	

1625629 rows × 44 columns

TEST DATA:

Name	proto	state	dur	sbytes	dbytes	sttl	dttl	sloss	dloss	service	...	is_ftp_login	ct_ftp_cmd	ct_srv_src	ct_srv_dst	ct_dst_ltm	ct_src_ltm	ct_src_dport_ltm	ct_dst_sp
1070804	tcp	FIN	0.438857	4776	3080	31	29	7	7	None	...	0.0	0	6	3	4	5	1	
1054569	tcp	CON	0.026163	2230	13900	31	29	7	10	None	...	0.0	0	2	7	2	1	1	
1548299	tcp	FIN	0.014954	2854	26584	31	29	7	16	None	...	0.0	0	9	6	4	8	1	
2305760	udp	INT	0.000009	114	0	254	0	0	0	dns	...	0.0	0	33	33	17	17	17	
1989356	udp	CON	0.001562	544	304	31	29	0	0	None	...	0.0	0	6	7	5	6	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
358801	tcp	FIN	0.046476	320	1890	31	29	1	2	ftp-data	...	0.0	0	3	5	3	7	1	
556159	tcp	FIN	0.027755	5928	8010	31	29	14	17	ssh	...	0.0	0	1	1	8	3	1	
1408939	udp	INT	0.000003	264	0	60	0	0	0	dns	...	0.0	0	39	39	18	18	18	
1894282	tcp	FIN	0.057911	2766	27392	31	29	7	16	None	...	0.0	0	2	2	3	4	1	
2148776	tcp	FIN	0.263385	1920	4416	31	29	6	6	None	...	0.0	0	9	4	6	9	2	

1625629 rows × 44 columns

x_train and x_test:

```
print(x_train)
```

```
[[ 0.          0.          0.         ... -0.42934342 -0.41959965
  -0.34135356]
 [ 0.          0.          0.         ... -0.42934342 -0.41959965
  -0.51906841]
 [ 0.          0.          0.         ... -0.42934342 -0.41959965
  -0.51906841]
 ...
 [ 0.          0.          0.         ...  1.5752582   2.33470341
   2.85751364]
 [ 0.          0.          0.         ... -0.42934342 -0.41959965
  -0.51906841]
 [ 0.          0.          0.         ... -0.31142568 -0.25758182
  -0.43021098]]
```

+ Code + Markdown

```
print(x_test)
```

```
[[ 0.          0.          0.         ... -0.42934342 -0.41959965
  -0.51906841]
 [ 0.          0.          0.         ... -0.42934342 -0.41959965
  -0.51906841]
 [ 0.          0.          0.         ...  2.4006824   3.4688282
   2.14665427]
 ...
 [ 0.          0.          0.         ... -0.42934342 -0.41959965
  -0.25249614]
 [ 0.          0.          0.         ... -0.42934342 -0.41959965
  -0.51906841]
 [ 0.          0.          0.         ...  1.45734046  2.17268558
   0.90265035]]
```

UNIQUE COUNTS:

```
unique_values, counts = np.unique(y_train, return_counts=True)
for value, count in zip(unique_values, counts):
    print(f"Value: {value}, Count: {count}")
```

```
Value: analysis, Count: 1716
Value: backdoor, Count: 1499
Value: dos, Count: 10454
Value: exploits, Count: 28640
Value: fuzzers, Count: 15494
Value: generic, Count: 137574
Value: normal, Count: 1420187
Value: reconnaissance, Count: 8985
Value: shellcode, Count: 979
Value: worms, Count: 101
```

```
unique_values, counts = np.unique(y_test, return_counts=True)


for value, count in zip(unique_values, counts):
    print(f"Value: {value}, Count: {count}")
```

```
Value: analysis, Count: 544
Value: backdoor, Count: 470
Value: dos, Count: 3288
Value: exploits, Count: 8709
Value: fuzzers, Count: 4928
Value: generic, Count: 43187
Value: normal, Count: 443714
Value: reconnaissance, Count: 2813
Value: shellcode, Count: 320
Value: worms, Count: 37
```

TRAINING MODEL:

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv1d_6 (Conv1D)	(None, 204, 16)	32
max_pooling1d_6 (MaxPooling1D)	(None, 102, 16)	0
batch_normalization_6 (BatchNormalization)	(None, 102, 16)	64
lstm_6 (LSTM)	(None, 102, 16)	2,112
conv1d_7 (Conv1D)	(None, 100, 32)	1,568
max_pooling1d_7 (MaxPooling1D)	(None, 50, 32)	0

batch_normalization_7 (BatchNormalization)	(None, 50, 32)	128
lstm_7 (LSTM)	(None, 50, 32)	8,320
conv1d_8 (Conv1D)	(None, 46, 64)	10,304
max_pooling1d_8 (MaxPooling1D)	(None, 23, 64)	0
batch_normalization_8 (BatchNormalization)	(None, 23, 64)	256
lstm_8 (LSTM)	(None, 64)	33,024
dense_4 (Dense)	(None, 64)	4,160
dropout_2 (Dropout)	(None, 64)	0
dense_5 (Dense)	(None, 10)	650

Total params: 60,618 (236.79 KB)

Trainable params: 60,394 (235.91 KB)

Non-trainable params: 224 (896.00 B)

ACCURACY:

```
history = model.fit(x_train, y_train, epochs=2, batch_size=512,validation_data=(x_val,y_val))

 #Evaluate the model
test_loss, test_accuracy, test_precision, test_recall = model.evaluate(x_test, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
print("Test Precision:", test_precision)
print("Test Recall:", test_recall)
```

```
Epoch 1/2
3176/3176 ━━━━━━━━━━━━━━━━━━━━ 1010s 315ms/step - accuracy: 0.9647 - loss: 0.1082 - precision_3: 0.9871 - recall_3: 0.9457 - val_accuracy:
0.9757 - val_loss: 0.0620 - val_precision_3: 0.9892 - val_recall_3: 0.9664
Epoch 2/2
3176/3176 ━━━━━━━━━━━━━━━━━━━━ 990s 312ms/step - accuracy: 0.9756 - loss: 0.0623 - precision_3: 0.9896 - recall_3: 0.9657 - val_accuracy:
0.9761 - val_loss: 0.0597 - val_precision_3: 0.9909 - val_recall_3: 0.9660
15876/15876 ━━━━━━━━━━━━━━━━━━━━ 413s 26ms/step - accuracy: 0.9757 - loss: 0.0603 - precision_3: 0.9908 - recall_3: 0.9655
Test Loss: 0.05975496396422386
Test Accuracy: 0.9759394526481628
Test Precision: 0.9908214807510376
Test Recall: 0.9660046100616455
```

- **Test Accuracy:** 97.59 %
- **Test Precision:** 99.08 %
- **Test Recall:** 96.60 %

ACCURACY: Average K-Fold Cross-Validation Results (on Validation Set):

```
y_train
```

```
array([[1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       ...,
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.]])
```

```
from sklearn.model_selection import StratifiedKFold
kfold = StratifiedKFold(n_splits=2, shuffle=True, random_state=42)
y_train_labels = np.argmax(y_train, axis=1)
y_train_labels
```

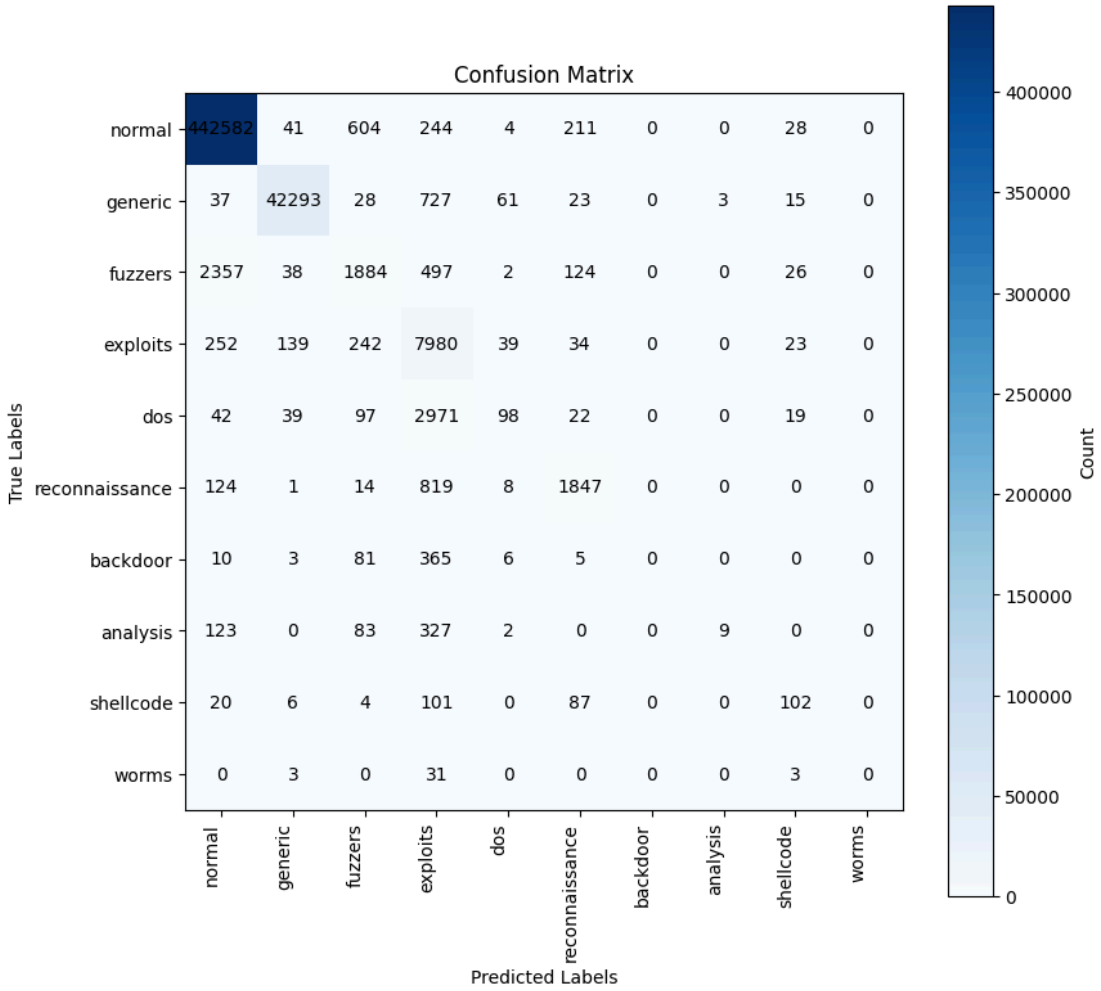
```
array([0, 0, 0, ..., 0, 0, 0])
```

```
scores = []
model = create_model()
for train_index, val_index in kfold.split(x_train, y_train_labels):
    X_train_inner, X_val_inner = x_train[train_index], x_train[val_index]
    y_train_inner, y_val_inner = y_train[train_index], y_train[val_index]

    model.fit(X_train_inner, y_train_inner, epochs=2, batch_size=1024,validation_data=(X_val_inner, y_val_inner))
    test_loss, test_acc, precision, recall = model.evaluate(x_val, y_val)
    scores.append([test_loss, test_acc, precision, recall])

print("Average K-Fold Cross-Validation Results (on Validation Set):")
print("Loss:", np.mean([score[0] for score in scores]))
print("Accuracy:", np.mean([score[1] for score in scores]))
print("Precision:", np.mean([score[2] for score in scores]))
print("Recall:", np.mean([score[3] for score in scores]))
```

```
Epoch 1/2
794/794 ━━━━━━━━━━━━━━━━━━━━ 549s 680ms/step - accuracy: 0.9447 - loss: 0.1940 - precision_4: 0.9775 - recall_4: 0.9063 - val_accuracy: 0.9
716 - val_loss: 0.0721 - val_precision_4: 0.9909 - val_recall_4: 0.9589
Epoch 2/2
794/794 ━━━━━━━━━━━━━━━━━━━━ 547s 689ms/step - accuracy: 0.9733 - loss: 0.0689 - precision_4: 0.9901 - recall_4: 0.9616 - val_accuracy: 0.9
751 - val_loss: 0.0630 - val_precision_4: 0.9906 - val_recall_4: 0.9638
12701/12701 ━━━━━━━━━━━━━━━━━━━━ 317s 25ms/step - accuracy: 0.9756 - loss: 0.0620 - precision_4: 0.9910 - recall_4: 0.9646
Epoch 1/2
794/794 ━━━━━━━━━━━━━━━━━━━━ 536s 675ms/step - accuracy: 0.9747 - loss: 0.0639 - precision_4: 0.9895 - recall_4: 0.9641 - val_accuracy: 0.9
761 - val_loss: 0.0606 - val_precision_4: 0.9895 - val_recall_4: 0.9665
Epoch 2/2
794/794 ━━━━━━━━━━━━━━━━━━━━ 537s 677ms/step - accuracy: 0.9756 - loss: 0.0616 - precision_4: 0.9897 - recall_4: 0.9658 - val_accuracy: 0.9
765 - val_loss: 0.0590 - val_precision_4: 0.9894 - val_recall_4: 0.9674
12701/12701 ━━━━━━━━━━━━━━━━━━━━ 334s 26ms/step - accuracy: 0.9766 - loss: 0.0587 - precision_4: 0.9894 - recall_4: 0.9675
Average K-Fold Cross-Validation Results (on Validation Set):
Loss: 0.060666393488645554
Accuracy: 0.9758727550506592
Precision: 0.9900006651878357
Recall: 0.9658618867397308
```

- **Accuracy:** 97.58 %
- **Precision:** 99.00 %
- **Recall:** 96.58 %

CONFUSION MATRIX:



Class Labels:

- Label 0: Represents normal network traffic.
- Label 1: Represents attack traffic, including various intrusion types like DoS, exploits, reconnaissance, etc.

```
unique_values, counts = np.unique(y_test, return_counts=True)

# Print the unique values and their corresponding counts
for value, count in zip(unique_values, counts):
    print(f"Value: {value}, Count: {count}")
```

```
Value: 0.0, Count: 4572090
Value: 1.0, Count: 508010
```