Configuration Manual

MSc Research Project
Programme Name: Master's of Science in Cyber Security

ROSHNI ROY
Student ID: 21203903

School of Computing
National College of Ireland

Supervisor: Michael Pantridge

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	ROSHNI ROY		
Student ID:	21203903		
Programme:	MASTER'S OF SCIENCE IN CYBERSECURITY	**Year:**	2023
Module:	MSC RESEARCH PROJECT		
Supervisor:	MICHAEL PANTRIDGE		
Submission Due Date:	30-01-2024		
Project Title:	DEEP LEARNING EMPOWERED INTRUSION DETECTION:UNMASKING THE PREVENTION CONCEPT		
Word Count:	2526 Page **Count : 22**		

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: ROSHNI ROY

Date: 30-01-2024

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

ROSHNI ROY

21203903

- The following paper demonstrates how to build a deep learning-based intrusion detection system (IDS) utilizing ANN and GRU models.
- It is based on the findings of the project "Deep Learning Empowered Intrusion Detection: Unmasking the Prevention Concept."

SYSTEM PREREQUISITES:

HARDWARE REQUIREMENTS:

- System Processor : i3 / i5 / i7
- Hard Disk : 1TB.
- Ram : 8 GB / 12 GB.
- Any desktop / Laptop system with the above configuration or higher level.

SOFTWARE REQUIREMENTS

- **Operating system** : Windows 8 / 10 (64 bits OS)
- **Programming Language** : Python 3.8
- **Framework** : Anaconda
- **Libraries** : Keras, TensorFlow, OpenCV
- **IDE** : JupyterNotebook
- **TOOLS**
  - Anaconda Navigator
  - Jupyter Notebook
  - Tensor flow, Keras
  - Matplotlip, SKLearn
  - Numpy, Pandas
- **PROGRAMMING LANGUAGES**
  - Python Programming

INSTRUCTIONS ON OBTAINING AND PREPROCESSING THE CICIDS2017 DATASET:

DATASET LINK:

https://data.world/dradar/cicids2017

- CICIDS2017 is an abbreviation for Canadian Institute for Cybersecurity Intrusion Detection Evaluation Dataset.
- It's a benchmark dataset for assessing intrusion detection systems (IDS).
- Accessibility to the general public: Because the dataset is open to the public, it is perfect for study and assessment.
- Tagged network traffic captures (PCAPs) and network flow statistics are included in CICIDS2017.
- These PCAPs and flow statistics indicate that there is both benign and malignant network activity.
- The major goal of CICIDS2017 is to evaluate the efficiency of intrusion detection systems in detecting and mitigating cyber attacks.
- Large Dimensions: The dataset is rather large, allowing for extended testing and evaluation of intrusion detection systems.
- Traffic in the Real World CICIDS2017 network traffic closely reflects real-world traffic patterns, allowing for a more realistic assessment environment for intrusion detection systems.
- CICIDS2017 may be used by researchers to train and evaluate IDS algorithms in real-world circumstances.
- Using information to analyze network traffic patterns aids to the development of more effective and robust intrusion detection systems.
- CICIDS2017 helps cybersecurity research by providing a significant resource for testing and enhancing intrusion detection systems.

  It contributes to an improved overall cybersecurity posture by encouraging the development of more capable intrusion detection systems.
- The CICIDS2017 dataset includes instances of both benign (normal) network traffic and common cyber-attacks.
- Size: The dataset is quite small, at 7.92 MB in size, making it easy to download and study.
- It is distributed under a Public Domain license, which implies that anybody can freely use, modify, and redistribute it.

- The dataset is arranged as a single file with 79 columns representing various network traffic aspects and properties.
- In contrast to standard table designs, CICIDS2017 is created without tables, which may necessitate additional preparation for analysis.
- The purpose of this dataset is to provide real-world examples of network activity, both regular and malicious, as a valuable resource for cybersecurity research and analysis.
- Its accessibility and comprehensive coverage of network activity make it a must-have tool for academics, analysts, and practitioners. It might help them better understand cybersecurity risks, create intrusion detection models, and improve network security.

STEPS FOR DATA SEGREGATION AND NORMALIZATION TO PREPARE IT FOR USE WITH THE MODELS.

DATA SEGREGATION:

Data segregation is critical for improving security by dividing data into independent components with distinct access restrictions. This multi-tiered approach lowers the likelihood of unauthorized access, data breaches, and data leaks, resulting in a more secure information environment.

Requirements for Compliance In firms that handle sensitive information, such as banking, healthcare, and personal information, data segregation is crucial for regulatory compliance. These standards frequently require data separation to protect privacy and ensure appropriate data management, supporting enterprises in avoiding legal and financial implications.

Data classification improves the efficiency of identifying, evaluating, and retrieving particular information, especially in huge datasets. This improved access to pertinent data has the potential to improve decision-making processes as well as overall operational efficiency.

Data segregation contributes to data integrity by ensuring that data is kept in its intended condition. It reduces the potential of mistakes and data quality issues while minimizing unintended modifications or corruption by dividing data into controlled settings.

Data Separation: Data segregation is a systematic approach to data management inside a company. It improves data management, access, and security while addressing security issues, assisting with compliance efforts, and resulting in a more robust and efficient information infrastructure.

DATA NORMALIZATION:

Data Normalization

```
In [35]: scaler = MinMaxScaler()
         scaler = scaler.fit(X.values)
         scaled_X = scaler.transform(X.values)

         df = pd.DataFrame(data=scaled_X,columns=X.columns)
         df['Label'] = y.values.ravel()
         df.head()
```

Figure 1: Data Normalization

- Using MinMaxScaler function is used to ensure that all numerical features in the dataset are scaled to a specified range which can be beneficial for algorithms that rely on feature scaling
- Scaling also helps these algorithms converge faster and perform better on the data.

DETAILED STEPS TO SET UP THE ANN AND GRU MODELS:

1. ARTIFICIAL NEURAL NETWORK (ANN):

- Machine learning models inspired by the structure of the human brain are known as ANNs. They are made up of linked artificial neurons that are structured into layers that include an input layer, one or more hidden layers, and an output layer. The weights of the connections between neurons can be changed during training to capture data patterns and correlations.

- ANNs employ activation functions to calculate each neuron's output depending on its inputs. Forward propagation (forward propagation) is the process of sending input data through the network to create

predictions, whereas backward propagation (backpropagation) is the process of adjusting the weights depending on the model's mistake to improve its performance. This iterative approach is repeated until the network learns to anticipate correctly.

- ANNs are quite popular.ANNs are extremely adaptable and have achieved success in a wide range of applications, including image recognition, natural language processing, and regression problems. Deep learning, a type of machine learning, often employs deep neural networks with several hidden layers, allowing ANNs to record sophisticated representations of complicated input and therefore gaining broad usage in current AI systems.

Algorithm-1 : ArtificialNeuralNetwork

In [10]:
```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Input, Dense, Dropout, BatchNormalization
from tensorflow.keras.backend import clear_session
```

In [11]:
```
ann_model = Sequential()
ann_model.add(Input(shape=(X_train.shape[1],)))
ann_model.add(Dense(32, activation="relu"))
ann_model.add(Dropout(0.5))
ann_model.add(Dense(64, activation="relu"))
ann_model.add(Dropout(0.5))
ann_model.add(Dense(128, activation='relu'))
ann_model.add(Dense(5, activation='softmax'))
ann_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Figure 2: Importing the necessary sequential feature

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                992

 dropout (Dropout)           (None, 32)                0

 dense_1 (Dense)             (None, 64)                2112

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 128)               8320

 dense_3 (Dense)             (None, 5)                 645

=================================================================
Total params: 12,069
Trainable params: 12,069
Non-trainable params: 0
_________________________________________________________________
```

Figure 3: Model Summary for ANN

2. GATED RECURRENT UNIT (GRU):

- GRUs are a sort of recurrent neural network (RNN) architecture that addresses some of standard RNN's shortcomings in terms of collecting and preserving sequential connections in input. GRUs were developed as a more efficient replacement for long short-term memory (LSTM) networks. GRUs are gated systems that control network data flow.
- An update gate, a reset gate, and a candidate concealed state are among them. GRUs can employ these gates to selectively update and reset the hidden state, allowing the model to capture long-term dependencies while avoiding the vanishing gradient problem seen in ordinary RNNs.
- GRUs have shown success in a variety of applications, most notably natural language processing tasks such as language modeling, machine translation, and sentiment analysis. Because of their architecture, they are well-suited for tasks involving the analysis and modeling of sequential patterns, making them a powerful tool in the field of deep learning.

Algorithm: 2 Gated Recurrent Unit

```
In [26]: from tensorflow.keras import Sequential
         from tensorflow.keras.layers import Flatten,Dense,Dropout,GRU,BatchNormalization

In [27]: gru_model = Sequential()
         gru_model.add(GRU(units=200, return_sequences=True, input_shape=(x_train.shape[1],x_train.shape[2])))
         gru_model.add(GRU(units=200, return_sequences=True))
         gru_model.add(Flatten())
         gru_model.add(Dropout(0.4))
         gru_model.add(Dense(256, activation='relu'))
         gru_model.add(Dense(5, activation='softmax'))
         gru_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Figure 4: Importing the necessary sequential features

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 30, 200)           121800

 gru_1 (GRU)                 (None, 30, 200)           241200

 flatten (Flatten)           (None, 6000)              0

 dropout (Dropout)           (None, 6000)              0

 dense (Dense)               (None, 256)               1536256

 dense_1 (Dense)             (None, 5)                 1285

=================================================================
Total params: 1,900,541
Trainable params: 1,900,541
Non-trainable params: 0
_________________________________________________________________
```

Figure 5: Model Summary for GRU

STEPS TO EVALUATE THE MODELS USING ACCURACY AND LOSS METRICS, CLASSIFICATION REPORT, AND CONFUSION MATRIX:

1. ARTIFICIAL NEURAL NETWORK (ANN)

1.1 CLASSIFICATION REPORT– ANN

This is a classification report on a machine learning model's performance on a dataset with five classes: BENIGN, Brute Force, DDoS, DoS, and PortScan. Precision, recall, and f1-score values for each class are presented, demonstrating the model's ability to properly identify instances of that class. The overall accuracy of the model is 95%, with high precision

and recall for the majority of classes, demonstrating its utility. Notably, the Brute Force and DDoS classes have excellent precision and recall, whereas DoS has far lower precision and recall. The macro and weighted averages illustrate the general balanced performance across classes. In summary, the model achieves an impressive 95% accuracy when distinguishing between different types of network data.

```
In [18]: class_labels = ['BENIGN', 'Brute Force', 'DDoS', 'DoS', 'PortScan']
```

Figure 6: The featured class labels

```
In [20]: print(classification_report(y_true=true_labels,y_pred=ann_pred, target_names=class_labels))
```

	precision	recall	f1-score	support
BENIGN	0.92	0.90	0.91	997
Brute Force	0.97	1.00	0.98	1047
DDoS	0.97	1.00	0.98	993
DoS	0.98	0.90	0.94	1003
PortScan	0.94	0.97	0.95	1077
accuracy			0.95	5117
macro avg	0.95	0.95	0.95	5117
weighted avg	0.95	0.95	0.95	5117

Figure 7: Classification Report for ANN

1.2 CONFUSION MATRIX-ANN

The confusion matrix for the ANN model shown above provides a complete evaluation of its performance on a dataset of 5117 samples. The algorithm correctly predicted 899 cases in the BENIGN category but misclassified 98 data. In contrast, 1044 of the Brute Force forecasts were right, with only three errors. In the DDOS class, 990 predictions were correct, while three were incorrect. The DOS class had 907 right guesses and 96 wrong predictions. 1045 predictions in the PortScan category were right, but 32 were inaccurate. This matrix gives a thorough assessment of the model's categorization performance, highlighting its strengths and weaknesses. While the model succeeds at recognizing negatives, the misclassifications in both groups indicate potential for improving overall predictive performance accuracy.

```
In [21]: plt.figure(figsize=(10,5))
         plt.rcParams['font.size'] = 15
         sns.heatmap(data=confusion_matrix(y_true=true_labels, y_pred=ann_pred),
                     annot=True,
                     cmap=plt.cm.Blues,
                     fmt = "5d",
                     xticklabels=class_labels,
                     yticklabels=class_labels,
                     cbar=False)
         plt.title(label="Confusion Matrix")
         plt.xticks(rotation=90)
         plt.yticks(rotation=0)
         plt.show()
```

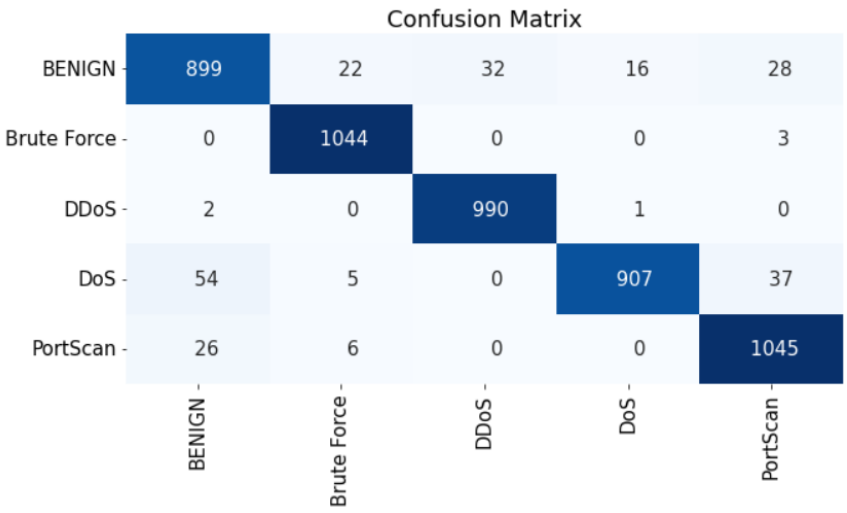

Figure 8: Confusion Matrix for ANN

1.3 ACCUARY PLOT GRAPH -ANN

An accuracy plot depicts how a model performs over time or in different conditions. On the x-axis, epochs or iterations are typically represented, with similar accuracy values depicted on the y-axis. This graph summarizes the model's training data learning and generalization. A rising trend indicates improved performance; nevertheless, oscillations or plateaus may indicate obstacles or the need for changes. Accuracy charts are useful tools for measuring training progress, detecting overfitting and underfitting, and directing model optimization decisions in machine learning and deep learning. Examining these graphs allows researchers and developers to fine-tune models, resulting in more effective and trustworthy machine learning systems.

```
In [13]: history=ann_model.fit(x=X_train.values,y=y_train,batch_size=64,epochs=10,validation_data=(X_test.values,y_test))
```

```
Epoch 1/10
320/320 [==============================] - 1s 2ms/step - loss: 1.0366 - accuracy: 0.5925 - val_loss: 0.4536 - val_accuracy: 0.9
074
Epoch 2/10
320/320 [==============================] - 1s 2ms/step - loss: 0.5100 - accuracy: 0.8394 - val_loss: 0.2614 - val_accuracy: 0.9
134
Epoch 3/10
320/320 [==============================] - 1s 2ms/step - loss: 0.3731 - accuracy: 0.8873 - val_loss: 0.2186 - val_accuracy: 0.9
250
Epoch 4/10
320/320 [==============================] - 1s 2ms/step - loss: 0.3306 - accuracy: 0.8980 - val_loss: 0.1949 - val_accuracy: 0.9
291
Epoch 5/10
320/320 [==============================] - 0s 2ms/step - loss: 0.2943 - accuracy: 0.9061 - val_loss: 0.1732 - val_accuracy: 0.9
279
Epoch 6/10
320/320 [==============================] - 1s 2ms/step - loss: 0.2721 - accuracy: 0.9130 - val_loss: 0.1620 - val_accuracy: 0.9
355
Epoch 7/10
320/320 [==============================] - 1s 2ms/step - loss: 0.2460 - accuracy: 0.9216 - val_loss: 0.1491 - val_accuracy: 0.9
384
Epoch 8/10
320/320 [==============================] - 1s 2ms/step - loss: 0.2359 - accuracy: 0.9258 - val_loss: 0.1322 - val_accuracy: 0.9
369
Epoch 9/10
320/320 [==============================] - 1s 2ms/step - loss: 0.2224 - accuracy: 0.9313 - val_loss: 0.1375 - val_accuracy: 0.9
457
Epoch 10/10
320/320 [==============================] - 1s 2ms/step - loss: 0.2156 - accuracy: 0.9331 - val_loss: 0.1233 - val_accuracy: 0.9
420
```

Figure 9: Train the values in x and y

```
with plt.style.context(style='bmh'):
    plt.figure(figsize=(18,8))
    plt.plot(history.history["accuracy"],label="accuracy",marker="o",markersize=10)
    plt.plot(history.history["val_accuracy"],label="val_accuracy",marker="*",markersize=10)
    plt.title(label="accuracy plot-graphs")
    plt.xlabel(xlabel='Epochs')
    plt.ylabel(ylabel='Accuracy')
    plt.xticks(range(0,10))
    plt.legend()
    plt.show()
```

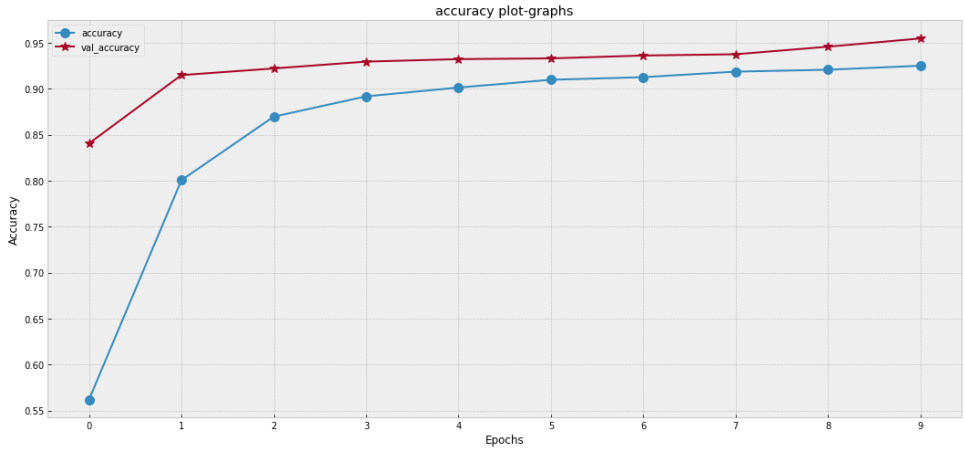

Figure 10: Accuracy plot Graph

1.4 LOSS PLOT GRAPH – ANN

A loss plot graph illustrates the loss function values of a machine learning model across time or iterations. The x-axis is typically used to indicate epochs or iterations, whereas the y-axis is used to show loss values. The loss function quantifies the model's performance and aims to minimize it throughout training. A decreasing trend in the loss within the plot implies improved model performance, indicating a gradual reduction in errors. Sudden spikes or changes may signal issues like overfitting or convergence issues. Regularly monitoring the loss plot is crucial for understanding the model's learning dynamics and assisting with the optimization process. The loss plot study gives valuable data that influences model adjustments such as hyperparameter tweaking or architectural changes, ultimately contributing to the development of more accurate and robust machine learning models.

```
plt.figure(figsize=(18,8))
plt.plot(history.history["loss"],label="loss",marker="o",markersize=10)
plt.plot(history.history["val_loss"],label="val_loss",marker="*",markersize=10)
plt.title(label="loss plot-graphs")
plt.xlabel(xlabel='Epochs')
plt.ylabel(ylabel='Loss')
plt.xticks(range(0,10))
plt.legend()
plt.show()
```

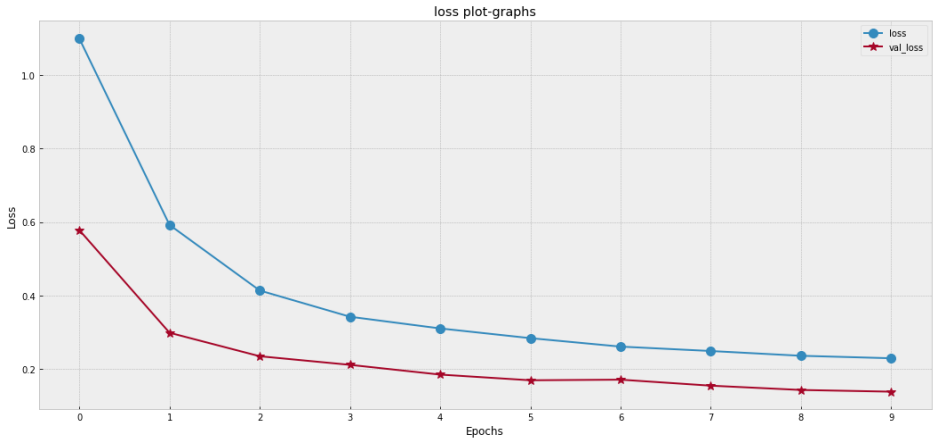

Figure 11: Loss plot Graph

2. GATED RECURRENT UNIT (GRU)

2.1 CLASSIFICATION REPORT– GRU

The performance of a model on a dataset with five unique classes is evaluated in this classification report: BENIGN, Brute Force, DDoS, DoS, and PortScan. The model is extremely exact, with an overall accuracy of 99%. Each class has excellent accuracy, recall, and f1-score values, showing that the model is capable of accurately recognizing instances of that class. Notably, the BENIGN class obtains flawless precision, whilst the other classes achieve near-perfect results, proving the model's superior performance. The model's overall excellent performance is confirmed by macro and weighted averages, which demonstrate its capacity to properly detect cases across several classes. In conclusion, the model obtains 99% accuracy, demonstrating its superiority in recognizing different samples within the dataset**.**

```
In [35]: print(classification_report(y_true=true_labels,y_pred=gru_pred, target_names=class_labels))
```

	precision	recall	f1-score	support
BENIGN	0.98	0.96	0.97	1026
Brute Force	0.99	1.00	0.99	1032
DDoS	0.98	1.00	0.99	1008
DoS	1.00	0.99	0.99	1006
PortScan	0.99	1.00	0.99	1045
accuracy			0.99	5117
macro avg	0.99	0.99	0.99	5117
weighted avg	0.99	0.99	0.99	5117

Figure 12: Classification report for GRU

2.2 CONFUSION MATRIX FOR GRU:

The confusion matrix of the GRU model provides an in-depth evaluation of its performance on a dataset of 5117 samples. The program correctly predicted 966 BENIGN events but incorrectly categorized 31 data. In comparison, 1044 out of 1044 Brute Force predictions were right, with only three errors. There were 991 right guesses and 2 erroneous predictions in the DDOS class. There were 999 right guesses and 4 incorrect predictions in the DOS class. There were 1071 successful guesses and 6 erroneous predictions in the PortScan category. This matrix examines the model's categorization abilities in depth, exposing its strengths and weaknesses. Although the model properly classifies negatives, the observed misclassifications in both categories indicate opportunities for improvement in overall accuracy.

```
In [36]: plt.figure(figsize=(10,5))
plt.rcParams['font.size'] = 15
sns.heatmap(data=confusion_matrix(y_true=true_labels, y_pred=gru_pred),
            annot=True,
            cmap=plt.cm.Blues,
            fmt = "5d",
            xticklabels=class_labels,
            yticklabels=class_labels,
            cbar=False)
plt.title(label="Confusion Matrix")
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.show()
```

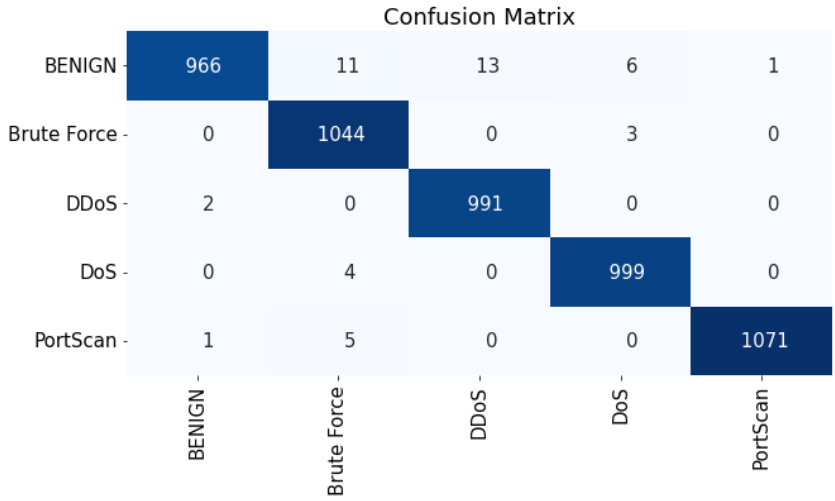

Figure 13: confusion matrix for GRU

2.3 ACCURACY PLOT GRAPH FOR GRU :

An accuracy plot illustrates the performance of a model over time or across configurations. The x-axis represents iterations or epochs, while the y-axis represents accuracy scores, providing a fast summary of learning and generalization. This graphical tool is essential in machine learning for tracking progress, detecting overfitting and underfitting, and directing optimization options for better application performance.

```
In [29]: history=gru_model.fit(x=x_train,y=y_train,batch_size=32,epochs=10,validation_data=(x_test,y_test))
```

```
Epoch 1/10
640/640 [==============================] - 43s 63ms/step - loss: 0.3571 - accuracy: 0.8849 - val_loss: 0.2028 - val_accuracy:
0.9429
Epoch 2/10
640/640 [==============================] - 41s 63ms/step - loss: 0.1745 - accuracy: 0.9427 - val_loss: 0.1212 - val_accuracy:
0.9539
Epoch 3/10
640/640 [==============================] - 40s 63ms/step - loss: 0.1352 - accuracy: 0.9533 - val_loss: 0.0931 - val_accuracy:
0.9595
Epoch 4/10
640/640 [==============================] - 40s 63ms/step - loss: 0.1065 - accuracy: 0.9658 - val_loss: 0.0852 - val_accuracy:
0.9646
Epoch 5/10
640/640 [==============================] - 39s 60ms/step - loss: 0.0831 - accuracy: 0.9738 - val_loss: 0.0569 - val_accuracy:
0.9789
Epoch 6/10
640/640 [==============================] - 38s 59ms/step - loss: 0.0714 - accuracy: 0.9767 - val_loss: 0.0578 - val_accuracy:
0.9810
Epoch 7/10
640/640 [==============================] - 38s 59ms/step - loss: 0.0573 - accuracy: 0.9812 - val_loss: 0.0515 - val_accuracy:
0.9810
Epoch 8/10
640/640 [==============================] - 38s 59ms/step - loss: 0.0482 - accuracy: 0.9844 - val_loss: 0.0549 - val_accuracy:
0.9824
Epoch 9/10
640/640 [==============================] - 37s 58ms/step - loss: 0.0484 - accuracy: 0.9858 - val_loss: 0.0641 - val_accuracy:
0.9828
Epoch 10/10
640/640 [==============================] - 38s 60ms/step - loss: 0.0429 - accuracy: 0.9877 - val_loss: 0.0484 - val_accuracy:
0.9881
```

Figure 14: train the values x and y

```
with plt.style.context(style='fivethirtyeight'):
    plt.figure(figsize=(18,8))
    plt.plot(history.history["accuracy"],label="accuracy",marker="o",markersize=10)
    plt.plot(history.history["val_accuracy"],label="val_accuracy",marker="*",markersize=10)
    plt.title(label="accuracy plot-graphs")
    plt.xlabel(xlabel='Epochs')
    plt.ylabel(ylabel='Accuracy')
    plt.xticks(range(0,10))
    plt.legend()
    plt.show()
```

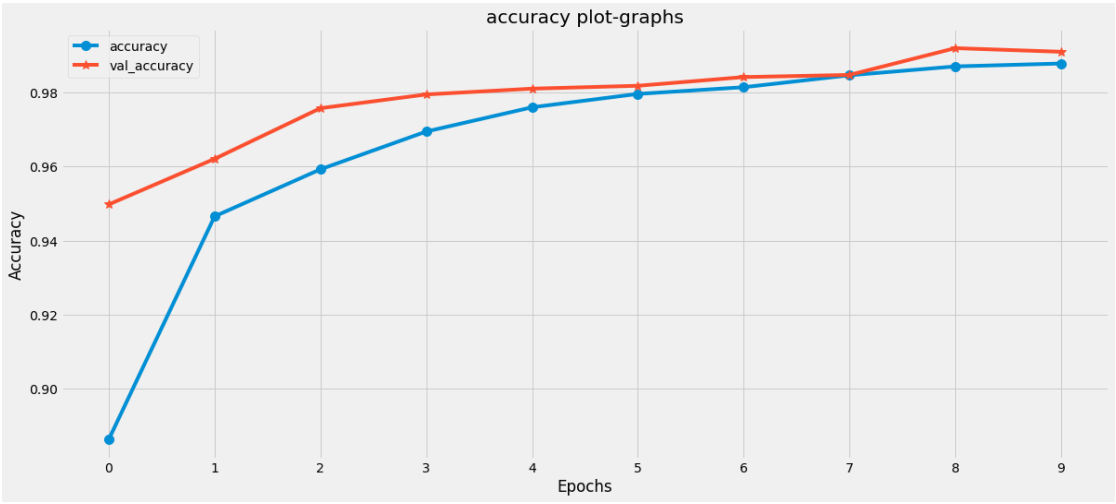

Figure 15: Accuracy plot Graph

2.4 LOSS PLOT GRAPH FOR GRU

A loss plot depicts the evolution of a machine learning model's loss function over time. The x-axis represents iterations or epochs, whereas the y-axis represents loss values. Training aims to lower the loss function, with a declining trend indicating better performance and fewer mistakes. Sudden increases may indicate a problem, such as overfitting. Monitoring the loss plot on a regular basis is critical for understanding the model's learning dynamics, directing optimization, and improving overall accuracy and robustness.

```
plt.figure(figsize=(18,8))
plt.plot(history.history["loss"],label="loss",marker="o",markersize=10)
plt.plot(history.history["val_loss"],label="val_loss",marker="*",markersize=10)
plt.title(label="loss plot-graphs")
plt.xlabel(xlabel='Epochs')
plt.ylabel(ylabel='Loss')
plt.xticks(range(0,10))
plt.legend()
plt.show()
```

Figure 16: Loss plot Graph for GRU

TESTING:

1.SERVER SIDE TESTING :

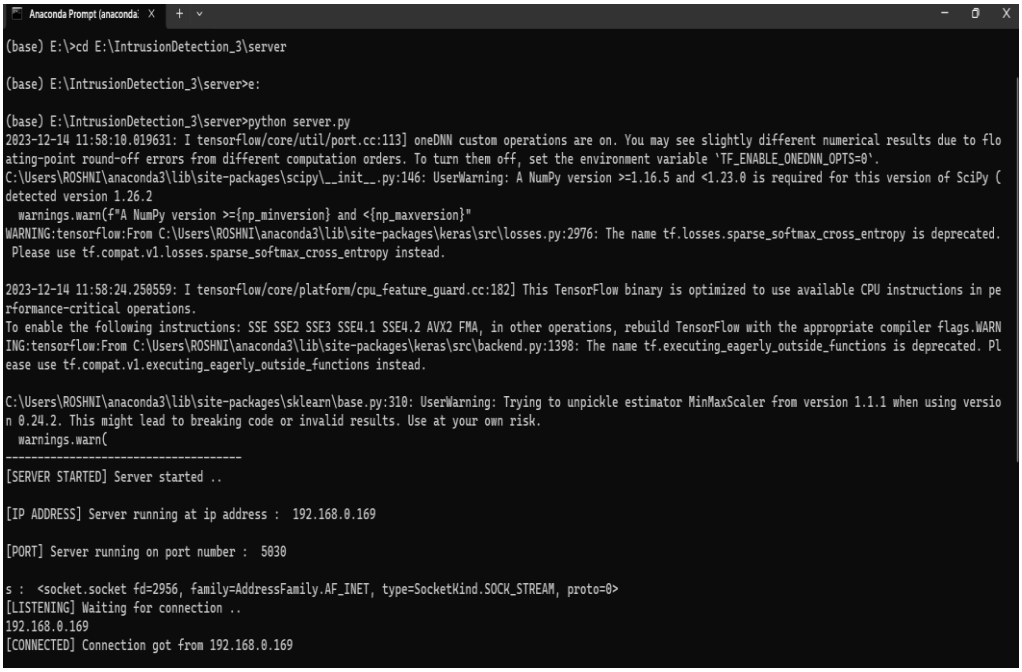

Figure 17: Getting connection from the client side

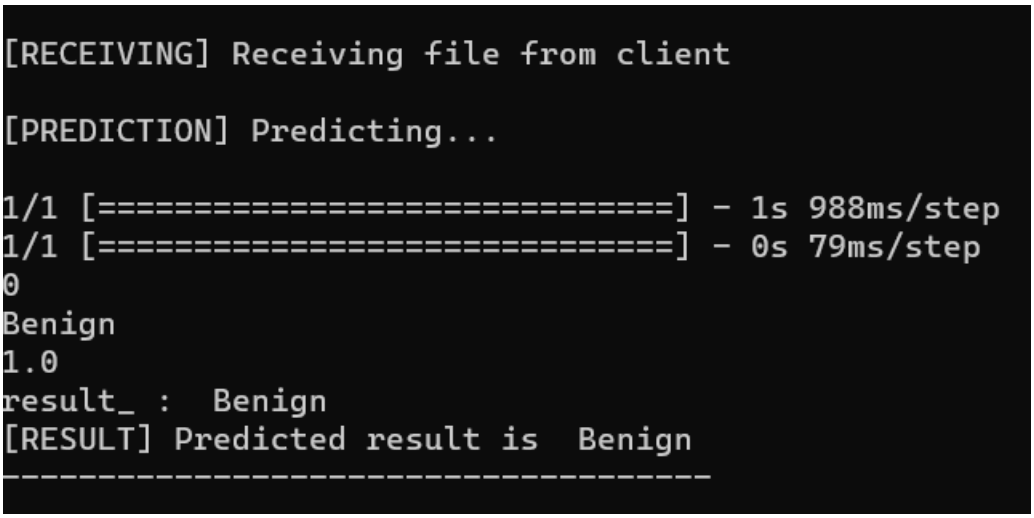

Figure 18:result of the Normal file

```
192.168.0.169
[CONNECTED] Connection got from 192.168.0.169

[RECEIVING] Receiving file from client

[PREDICTION] Predicting...

1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 40ms/step
2
DDoS
0.8952705
result_ :  DDoS
[RESULT] Predicted result is  DDoS
-----------------------------------

Email sent successfully
```

Figure 19:Prediction result of the attacked file

2. CLIENT SIDE TESTING :

```
Anaconda Prompt (anaconda:

(base) C:\Users\ROSHNI>e:

(base) E:\>cd E:\IntrusionDetection_3\client

(base) E:\IntrusionDetection_3\client>e:

(base) E:\IntrusionDetection_3\client>python client.py
enter ip address: 192.168.0.169
Enter input file name : file_1.csv
[SUCCESS] File sent successfully

(base) E:\IntrusionDetection_3\client>python client.py
enter ip address: 192.168.0.169
Enter input file name : file_13.csv
[SUCCESS] File sent successfully

(base) E:\IntrusionDetection_3\client>python client.py
enter ip address: 192.168.0.169
Enter input file name : file_23.csv
[SUCCESS] File sent successfully
```

Figure 20:Client side statements

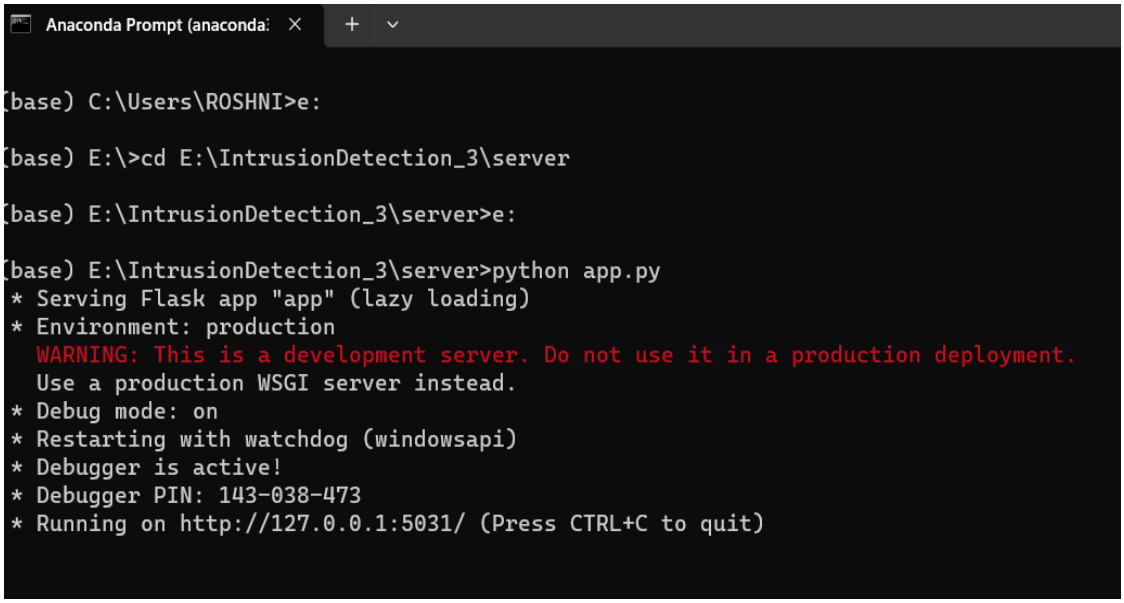

Figure 21:Importing files from libraries and used secret key to encrypt data

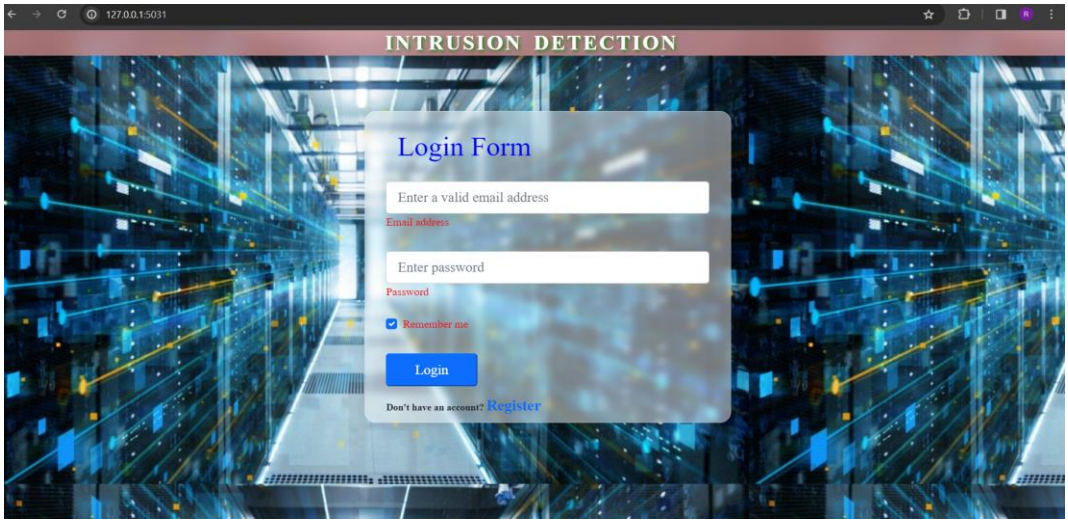

Figure 22: Login form of the forntend process

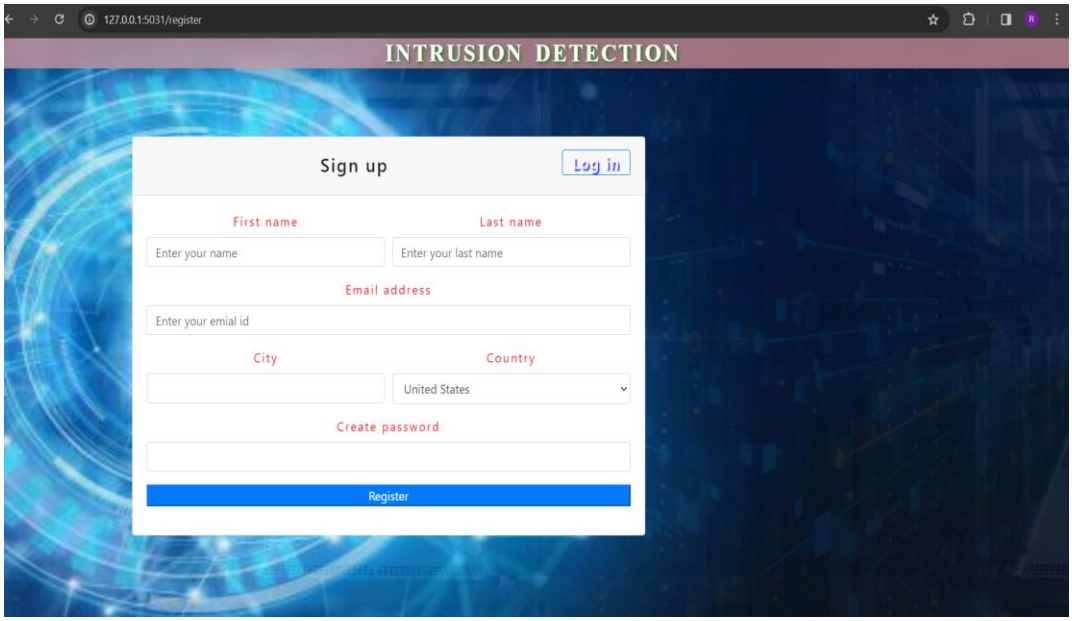

Figure 23: Registration form

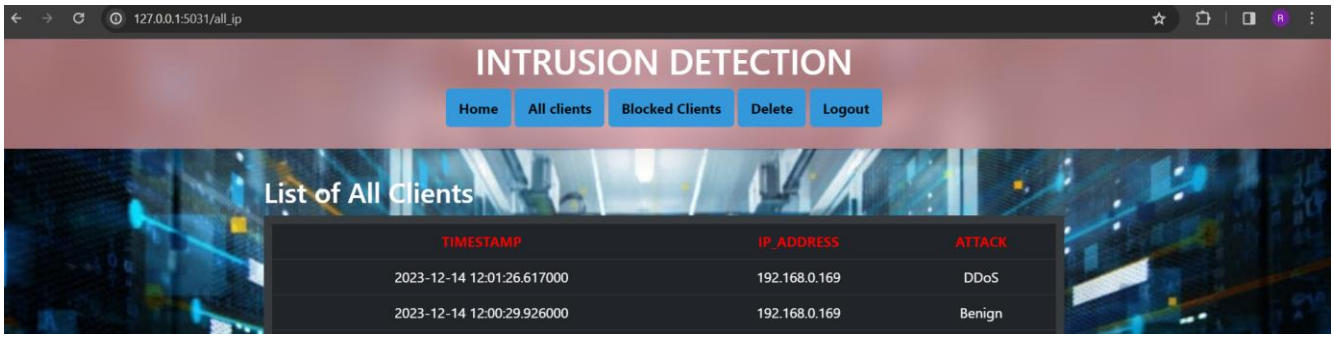

Figure 24: Displays the all clients

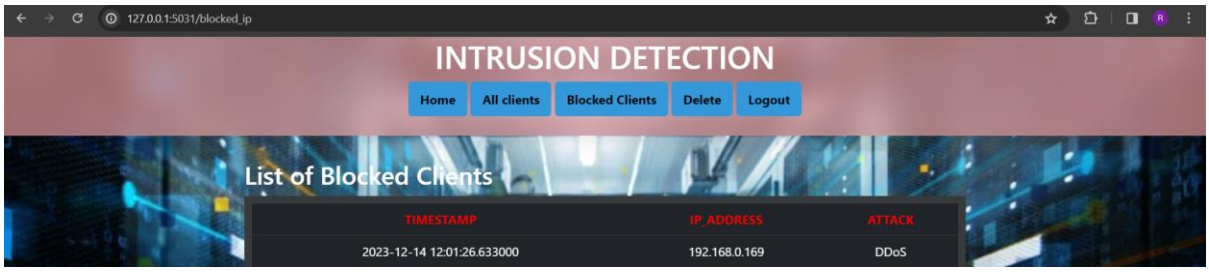

Figure 25: Blocked clients

RESULT ANALYSIS:

This research successfully achieved its aim of enhancing intrusion detection system (IDS) effectiveness through advanced deep learning techniques, particularly the artificial neural network (ANN) and Gated Recurrent Units (GRU) algorithms. The meticulous evaluation using the CICIDS2017 dataset, encompassing diverse intrusion classes, showcased the power of deep learning in significantly improving accuracy and efficiency in intrusion detection. The trained ANN and GRU models demonstrated exceptional performance, achieving

accuracy rates of 95.47% and 99.10%, respectively, in classifying five distinct intrusion types.

Artificial Neural Network(ANN):

```
In [18]: class_labels = ['BENIGN', 'Brute Force', 'DDoS', 'DoS', 'PortScan']
```

Accuracy Score

```
In [19]: ann_model_accuracy=accuracy_score(y_true=true_labels,y_pred=ann_pred)
         print("Validation accuracy of ArtificialNeuralNetwork model is {:.2f}%".format(ann_model_accuracy*100))

         Validation accuracy of ArtificialNeuralNetwork model is 94.20%
```

Gated Recurrent Unit (GRU):

```
In [34]: gru_model_accuracy=accuracy_score(y_true=true_labels,y_pred=gru_pred)
         print("Validation accuracy of GatedRecurrentUnit model is {:.2f}%".format(gru_model_accuracy*100))

         Validation accuracy of GatedRecurrentUnit model is 98.81%
```