International Journal for Innovative Engineering and Management Research
PEER REVIEWED OPEN ACCESS INTERNATIONAL JOURNAL
www.ijiemr.org

Identification of Harmful Attacks in IoT Using Deep Learning

Jangam Nagaraju, Deekshitha Pingali, Kesireddy Venkata Veera sai, Chintanaboina Anjali
[1]Assistant professor, MLR Institute of Technology
[2,3,4]Department of Cse-Cybersecurity, MLR Institute of Technology
nagarajuj513@gmail.com, p.deekshithareddy2003@gmail.com,
21r21a6226@mlrinstitutions.ac.in, anjalichinthanaboina@gmail.com

ABSTRACT

The swift IoT devices has transformed industries, but it has also made networks more vulnerable to a wide range of cyber threats. This study develops deep learning-based models to identify malicious activity in IoT ecosystems, utilizing a combination of Generative Adversarial Networks (GANs), Capsule Networks, and Multi-Layer Perceptrons (MLPs). GANs are employed to address data imbalance by generating synthetic IoT data, including uncommon attack scenarios. Capsule Networks are used to detect intricate attack patterns by analyzing complex feature relationships. Finally, the MLP classifier leverages these rich representations to accurately differentiate between benign and malicious behaviors. Experimental findings highlight the efficacy of the suggested model in enhancing the security of IoT networks.

Keywords: Internet of Things (IoT), Deep Learning, Cybersecurity, Network Security, Anomaly Detection, Multi-Layer Perceptrons (MLPs), Generative Adversarial Networks (GANs)

INTRODUCTION

The IoT has introduced significant advancements in the way devices and systems interact. With billions of IoT devices deployed worldwide—from smart home systems to industrial automation—IoT has brought about considerable improvements in efficiency and functionality across multiple industries. However, the scale and diversity of these devices pose substantial security challenges. Each IoT device serves as a possible access point for cybercriminals, posing a threat to the network security a critical concern for IoT ecosystems. Securing IoT systems is essential, as a breach could have severe consequences,ranging from unauthorized access to confidential information to the disruption of critical services. The interconnected nature of IoT networks further complicates matters, as a single compromised device can quickly infect the entire ecosystem. Traditional security measures often fall short in such dynamic, resource-constrained environments, where devices continuously join or leave the network. As a result, Intrusion Detection Systems (IDS) based on signature detection methods are typically inadequate against emerging or unknown attack types.

Machine learning (ML), particularly deep learning, has demonstrated promise in overcoming the limitations of traditional approaches by enabling the detection of complex and novel attack patterns. Nonetheless, challenges remain, including the issue of class imbalance in attack datasets and the difficulty in capturing intricate attack patterns. This paper proposes a hybrid deep learning model that integrates Generative Adversarial Networks (GANs), Capsule Networks (CapsNets), and Multi-Layer Perceptrons (MLPs) to enhance the detection of malicious activities in IoT networks. GANs address data imbalance by generating synthetic attack data, CapsNets help identify sophisticated attack patterns, and MLPs are employed to classify benign versus malicious activities.

LITERATURE REVIEW

As IoT continues to expand, its security risks have become a central focus. The large number of interconnected devices creates multiple vulnerabilities, and a breach in one device can compromise the entire network. Traditional security approaches like signature-based detection systems are ineffective against new or evolving attack methods. Machine learning techniques, especially deep learning, have shown significant potential in overcoming these shortcomings. By learning complex attack patterns from large datasets, deep learning models can adapt to new threats. However, IoT security presents additional challenges, such as resource limitations on IoT devices and the constantly changing nature of IoT networks. Various studies have explored different machine learning models to address these challenges. For instance, one study proposes a distributed deep learning framework using DCNN embedded in IoT devices and a cloud-based LSTM network. The DCNN detects phishing and application-layer attacks at the device level, while the LSTM processes data on the cloud to detect Botnet activities and distributed phishing. The proposed hybrid approach in this study builds upon these advancements by leveraging GANs for data augmentation, CapsNets for hierarchical feature detection, and MLPs for classification, ultimately aiming to provide a robust solution for IoT security.

PROPOSED SYSTEM

The proposed system combines GANs, Capsule Networks, and MLPs to create a comprehensive deep learning model capable of detecting cyber-attacks in IoT networks. This approach aims to overcome several key challenges in IoT security, such as handling unstructured data capturing complex attack patterns, and adjusting to the ever-changing nature of IoT networks.

Step 1: Data Preprocessing and Analysis

The first step involves collecting and cleaning IoT datasets that include both benign and malicious activity. This includes removing noise, handling missing data, and addressing outliers. Data normalization and scaling are applied to ensure uniformity across numerical features. EDA is performed to gain insights into the data distribution and recognize potential class imbalances between benign and attack instances.

Step 2: Synthetic Data Generation Using GANs

To address data imbalance, GANs are employed to create synthetic attack data, especially for attack types with limited representation. The GAN framework comprises two neural networks: a generator that produces synthetic attack instances and a discriminator that differentiates between real and generated samples. data from synthetic data. This synthetic data generation helps improve the model's ability to detect both common and rare attack patterns.

Step 3: Feature Engineering and Representation

Relevant features, such as network traffic patterns, CPU usage, and device-specific metrics, are extracted from the IoT datasets. Feature selection methods are used to identify the most significant features for attack detection, and dimensionality reduction techniques, such as PCA, are used to reduce the complexity of the data. Feature scaling ensures that the model can effectively learn from the data by normalizing it to a common scale.

Step 4: Modeling with Capsule Networks

Capsule Networks are employed to capture hierarchical relationships within the data. CapsNets are particularly useful in detecting complex attack patterns, such as sequential actions across multiple devices or unusual temporal relationships. The Capsule Network learns these relationships to better understand and identify sophisticated attacks that may not be immediately apparent using traditional machine learning models.

Step 5: Training the MLP Classifier

The Multi-Layer Perceptron (MLP) classifier is used for the final classification task. The MLP receives the feature representations produced by the Capsule Network and classifies IoT activities as either benign or malicious. The MLP is trained using both real and synthetic data to improve classification accuracy. A combination of activation functions, such as ReLU for hidden layers and softmax for the output layer, is used to generate probabilistic outputs.

Step 6: Model Evaluation

The output is evaluated using a range of metrics,based on the TP and FP and TN and FN rate and AUC. Cross-validation is employed to assess the model's generalizability and to avoid overfitting. Performance is benchmarked against other models, such as RF and SVM, to determine the efficacy of the proposed approach.

Step 7: Optimization and Fine-Tuning

Tuning is performed using methods like grid search or random search to identify optimal parameters, including learning rate and batch size. Regularization techniques, such as dropout or weight decay, are used to avoid overfitting.

Step 8: Deployment and Real-Time Monitoring

After the model is trained and optimized, it is deployed in a real-time IoT network monitoring system. The model continuously processes incoming IoT data to identify and mitigate potential attacks as they occur. Additionally, the system is designed to adapt to new attack patterns by periodically retraining the model with fresh data.

METHODOLOGY

IoT is rapidly expanding, with millions of devices now connected to the network. This vast network of devices supports a wide range of smart applications, from home automation to industrial monitoring. However, the proliferation of IoT devices also introduces significant cybersecurity risks. As the number of connected devices increases, so does the attack surface for potential cyberattacks. In response to this challenge, there is growing interest in applying advanced deep learning (DL) techniques to enhance the detection of malicious activities in real time.This article explores how deep learning can be utilized for real-time cyberattack detection in IoT environments. It covers the entire process, from data collection and preprocessing to model selection, training, and deployment, offering a structured methodology to secure IoT systems using cutting-edge deep learning algorithms. DL,, a subset of ML and NN with multiple layers to automatically extract patterns from complex datasets. This makes deep learning particularly well-suited for IoT environments, where devices generate large volumes of diverse data. By training deep learning models to recognize normal behavior patterns, these systems can efficiently identify potential threats, such as DoS attacks, MitM attacks, device hijacking, and data injection. The primary objective of deep learning for IoT security is to enable models to distinguish between benign and malicious activities, classifying network traffic and sensor data as either normal or anomalous.

Deep Learning Models for IoT Attack Detection

Several deep learning architectures can be applied to IoT attack detection, each suited to different types of data—such as time-series, spatial, or mixed datasets. Below are the key models used in this context:

1. CNN

Although CNNs are most commonly associated with image processing, their ability to identify spatial patterns makes them effective for analyzing IoT network traffic. CNNs can automatically detect patterns such as traffic spikes or unusual packet sizes, which are indicative of DoS attacks or network anomalies.

CNN-based Detection Algorithm:

- **Input Layer**: The model ingests IoT data in the form of a matrix where each row represents a specific feature (e.g., packet size, timestamp, protocol).
- **Convolutional Layers**: These layers scan the input data using filters to identify local patterns like traffic anomalies or burst packets.
- **ReLU**: The ReLU introduces non-linearity to help the model capture more complex patterns.
- **Pooling Layers**: It reduces the attributes of feature maps while preserving significant information, making the model more efficient.
- **Fully Connected Layers**: These layers interpret high-level features and map them to output classes (normal or attack).
- **Output Layer**: A softmax function classifies the data as either normal or anomalous based on the learned features.

2. RNN and LSTM Networks

RNNs and LSTMs are designed for sequential data, making them ideal for analyzing time-series data, such as IoT sensor readings or network logs. Since many attacks unfold gradually, these models can capture temporal dependencies and detect slow-evolving anomalies.

RNN/LSTM-based Detection Algorithm:

- **Input Sequence**: The model processes a series of time-stamped data points (e.g., sensor readings or network packets).
- **Hidden States (RNN)**: In an RNN, the hidden state is updated at each time step, integrating the current input with prior data to capture dependencies over time.
- **LSTM Networks**: LSTMs address the vanishing gradient problem of traditional RNNs. They use gates to control the flow of information, allowing them to retain information over longer periods.
- **Fully Connected Layers**: The LSTM output will go by fully connected layers, which map the features to a final classification (normal or attack).
- **Output Layer**: A softmax or sigmoid function classifies the sequence as either normal or anomalous, based on temporal patterns.

3. Autoencoders for Anomaly Detection

Autoencoders are unsupervised NN that learn to encode data into a reduce space and then reconstruct it. Since IoT systems generate mostly normal data with few anomalous events, autoencoders can be trained exclusively on normal data. They detect anomalies by measuring reconstruction errors, with significant deviations suggesting a potential attack.

Autoencoder-based Detection Algorithm:

- **Encoder**: It reduces the input data to the lower-dimensional.
- **Decoder**: It reconstructs the I/p data from the compressed latent representation.
- **Loss Function**: The reconstruction error (e.g., mean squared error) is computed, reflecting the difference between the original and reconstructed data.
- **Anomaly Detection**: If the reconstruction error exceeds a certain threshold, the data is flagged as anomalous, indicating a possible attack.

4. GANs

GANs are typically used for generating synthetic data, but their discriminator component can be employed for attack detection. The user provided a data, is tells about real and malicious data, making it an effective tool for detecting cyberattacks.

GAN-based Detection Algorithm:

- **Generator**: The generator creates synthetic data that mimics normal IoT traffic or sensor readings.
- **Discriminator**: It is trained between real and synthetic data, learning to identify normal versus attack patterns.
- **Adversarial Training**: As the generator improves in producing realistic synthetic data, the discriminator becomes better at detecting malicious activities.

- **Classification**: Once trained, the discriminator classifies incoming data as normal or anomalous based on the learned characteristics of IoT network behavior.

Training Deep Learning Models

Once an appropriate deep learning model is selected, it is trained using labeled datasets that contain both normal and attack data. The training process includes data preprocessing, feature extraction, and model fine-tuning to optimize performance. During this phase, the model learns to recognize attack patterns and classify incoming data accurately.

Real-Time Attack Detection and Deployment

After training, the deep learning model is deployed for real-time attack detection in IoT environments. The model continuously monitors data streams from IoT devices or network traffic, classifying data as either benign or malicious. If an attack is detected, the system can trigger an alert and take appropriate action, such as isolating compromised devices or blocking malicious traffic.

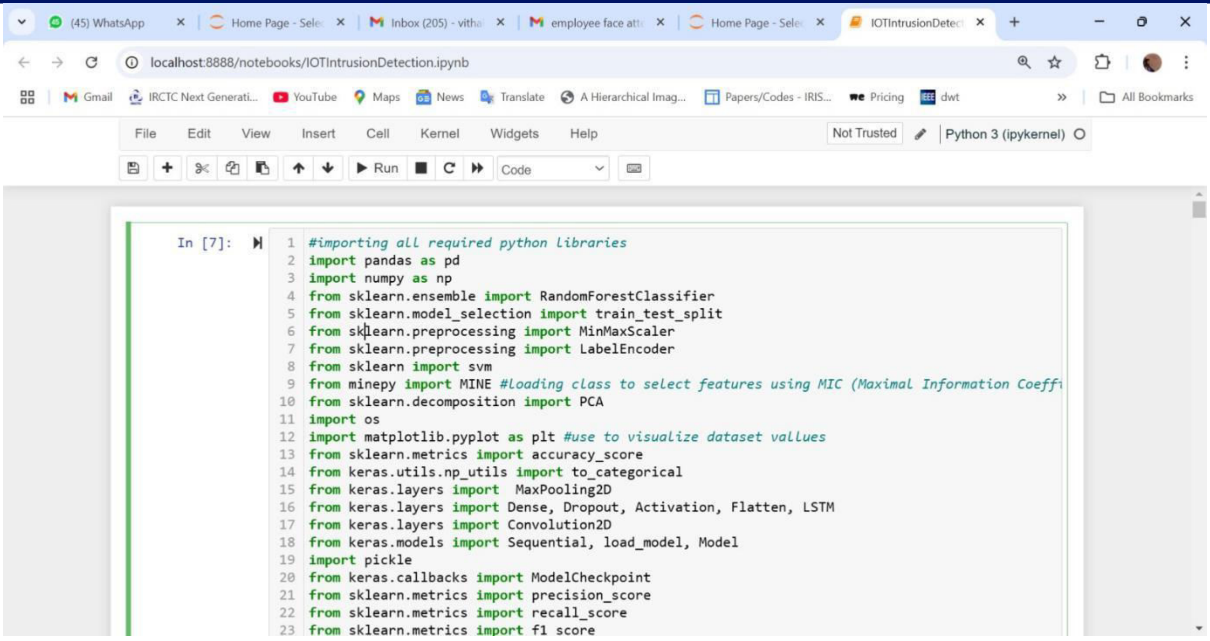
By leveraging advanced deep learning techniques like CNNs, RNNs, LSTMs, autoencoders, and GANs, IoT systems can achieve automated and highly efficient attack detection. These models not only improve the adaptability and scalability of IoT security but also provide a proactive approach to mitigating cyber threats in real-time.

RESULTS AND DISCUSSION

While the DNN model is to check cyberattacks in IoT systems offers significant advantages, such as improved detection accuracy and scalability, there must be so many tasks that need to be overcome such as:

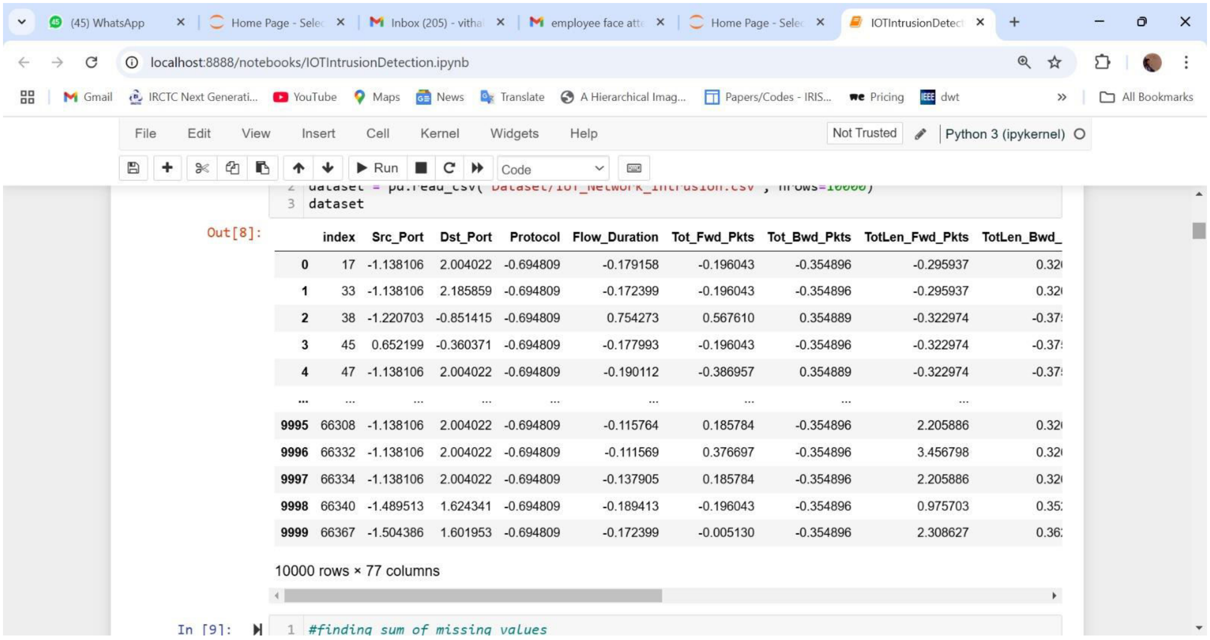
- **Computational Overhead**: Deep learning models, particularly those based on large neural networks, can require significant computational resources, which may be a concern for IoT devices with limited processing power.
- **Data Imbalance**: The vast majority of IoT data is normal, with only a small percentage representing malicious activities. This imbalance can make it difficult for models to learn effective attack patterns without overfitting.
- **Model Interpretability**: Deep learning models, especially DNN is taken "black boxes," making it difficult to interpret why a particular decision was made. This lack of transparency is a challenge for security professionals who need to understand and trust the model’s predictions.

Despite these challenges, deep learning remains a promising approach to securing IoT networks. By addressing issues like computational efficiency, data imbalance, and interpretability, deep learning-based systems can provide a robust and scalable solution for safeguarding the growing number of IoT devices against evolving cyber threats.



```
In [7]:
#importing all required python libraries
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn import svm
from minepy import MINE #loading class to select features using MIC (Maximal Information Coeffi
from sklearn.decomposition import PCA
import os
import matplotlib.pyplot as plt #use to visualize dataset values
from sklearn.metrics import accuracy_score
from keras.utils.np_utils import to_categorical
from keras.layers import  MaxPooling2D
from keras.layers import Dense, Dropout, Activation, Flatten, LSTM
from keras.layers import Convolution2D
from keras.models import Sequential, load_model, Model
import pickle
from keras.callbacks import ModelCheckpoint
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
```

Fig 1. In above screen importing required python packages and classes



```
dataset = pd.read_csv("Dataset/IoT_Network_Intrusion.csv", nrows=10000)
dataset
```

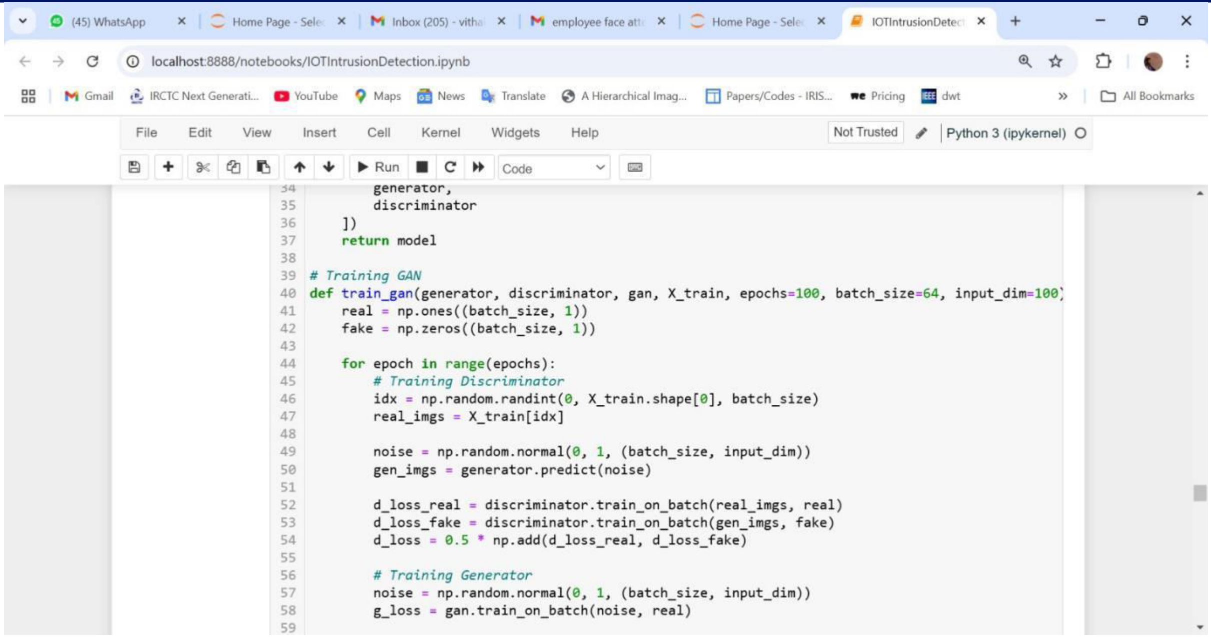
Out[8]:

	index	Src_Port	Dst_Port	Protocol	Flow_Duration	Tot_Fwd_Pkts	Tot_Bwd_Pkts	TotLen_Fwd_Pkts	TotLen_Bwd_
0	17	-1.138106	2.004022	-0.694809	-0.179158	-0.196043	-0.354896	-0.295937	0.32
1	33	-1.138106	2.185859	-0.694809	-0.172399	-0.196043	-0.354896	-0.295937	0.32
2	38	-1.220703	-0.851415	-0.694809	0.754273	0.567610	0.354889	-0.322974	-0.37
3	45	0.652199	-0.360371	-0.694809	-0.177993	-0.196043	-0.354896	-0.322974	-0.37
4	47	-1.138106	2.004022	-0.694809	-0.190112	-0.386957	0.354889	-0.322974	-0.37
...	...	...	...	...	...	...	...	...	
9995	66308	-1.138106	2.004022	-0.694809	-0.115764	0.185784	-0.354896	2.205886	0.32
9996	66332	-1.138106	2.004022	-0.694809	-0.111569	0.376697	-0.354896	3.456798	0.32
9997	66334	-1.138106	2.004022	-0.694809	-0.137905	0.185784	-0.354896	2.205886	0.32
9998	66340	-1.489513	1.624341	-0.694809	-0.189413	-0.196043	-0.354896	0.975703	0.35
9999	66367	-1.504386	1.601953	-0.694809	-0.172399	-0.005130	-0.354896	2.308627	0.36

10000 rows × 77 columns

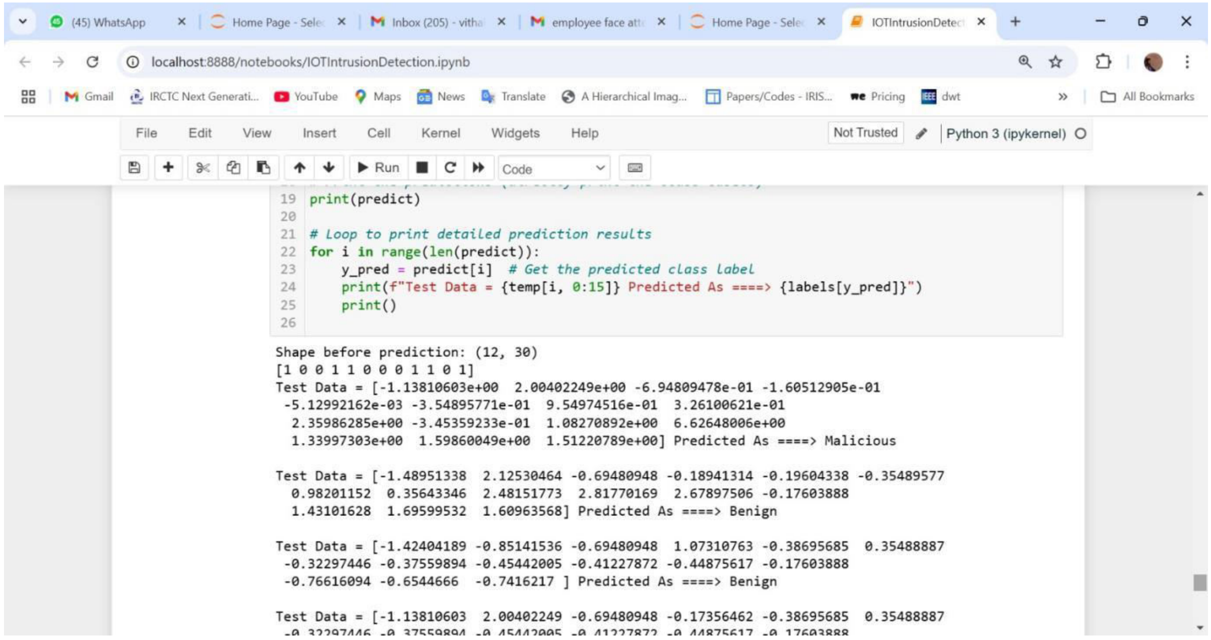
```
In [9]:
#finding sum of missing values
```

Fig 2. In above screen defining function to find and display different labels found in dataset



```
        generator,
        discriminator
    ])
    return model

# Training GAN
def train_gan(generator, discriminator, gan, X_train, epochs=100, batch_size=64, input_dim=100)
    real = np.ones((batch_size, 1))
    fake = np.zeros((batch_size, 1))

    for epoch in range(epochs):
        # Training Discriminator
        idx = np.random.randint(0, X_train.shape[0], batch_size)
        real_imgs = X_train[idx]

        noise = np.random.normal(0, 1, (batch_size, input_dim))
        gen_imgs = generator.predict(noise)

        d_loss_real = discriminator.train_on_batch(real_imgs, real)
        d_loss_fake = discriminator.train_on_batch(gen_imgs, fake)
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

        # Training Generator
        noise = np.random.normal(0, 1, (batch_size, input_dim))
        g_loss = gan.train_on_batch(noise, real)
```

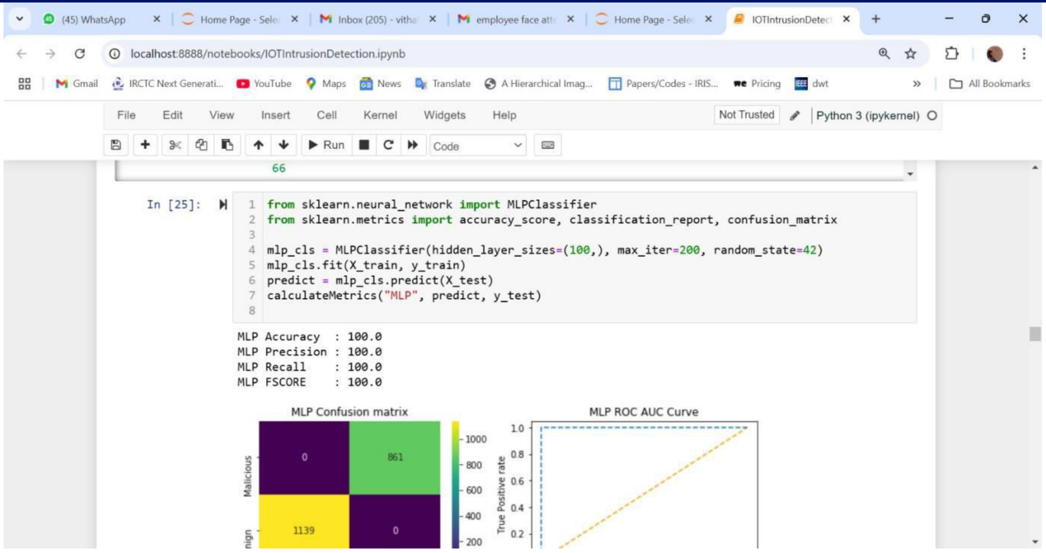
Fig 3. In above screen looping and reading all images from dataset folder and then resizing and adding to training X and Y array and then in blue colour text displaying total number of images loaded



```
print(predict)

# Loop to print detailed prediction results
for i in range(len(predict)):
    y_pred = predict[i]  # Get the predicted class label
    print(f"Test Data = {temp[i, 0:15]} Predicted As ====> {labels[y_pred]}")
    print()
```

```
Shape before prediction: (12, 30)
[1 0 0 1 1 0 0 0 1 1 0 1]
Test Data = [-1.13810603e+00  2.00402249e+00 -6.94809478e-01 -1.60512905e-01
 -5.12992162e-03 -3.54895771e-01  9.54974516e-01  3.26100621e-01
  2.35986285e+00 -3.45359233e-01  1.08270892e+00  6.62648006e+00
  1.33997303e+00  1.59860049e+00  1.51220789e+00] Predicted As ====> Malicious

Test Data = [-1.48951338  2.12530464 -0.69480948 -0.18941314 -0.19604338 -0.35489577
  0.98201152  0.35643346  2.48151773  2.81770169  2.67897506 -0.17603888
  1.43101628  1.69599532  1.60963568] Predicted As ====> Benign

Test Data = [-1.42404189 -0.85141536 -0.69480948  1.07310763 -0.38695685  0.35488887
 -0.32297446 -0.37559894 -0.45442005 -0.41227872 -0.44875617 -0.17603888
 -0.76616094 -0.6544666  -0.7416217 ] Predicted As ====> Benign

Test Data = [-1.13810603  2.00402249 -0.69480948 -0.17356462 -0.38695685  0.35488887
```

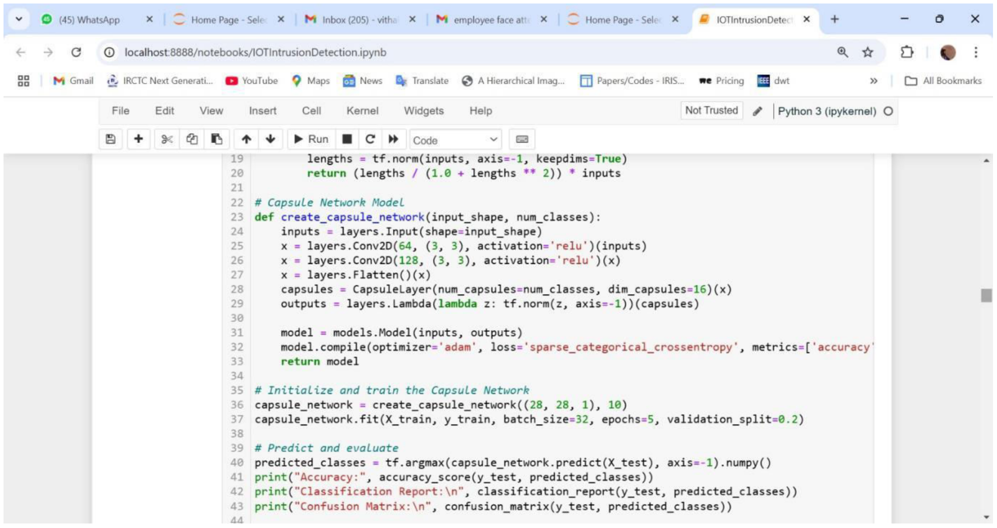
Fig 4. In above screen displaying predictions of the algorithm



```
In [25]:
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

mlp_cls = MLPClassifier(hidden_layer_sizes=(100,), max_iter=200, random_state=42)
mlp_cls.fit(X_train, y_train)
predict = mlp_cls.predict(X_test)
calculateMetrics("MLP", predict, y_test)
```

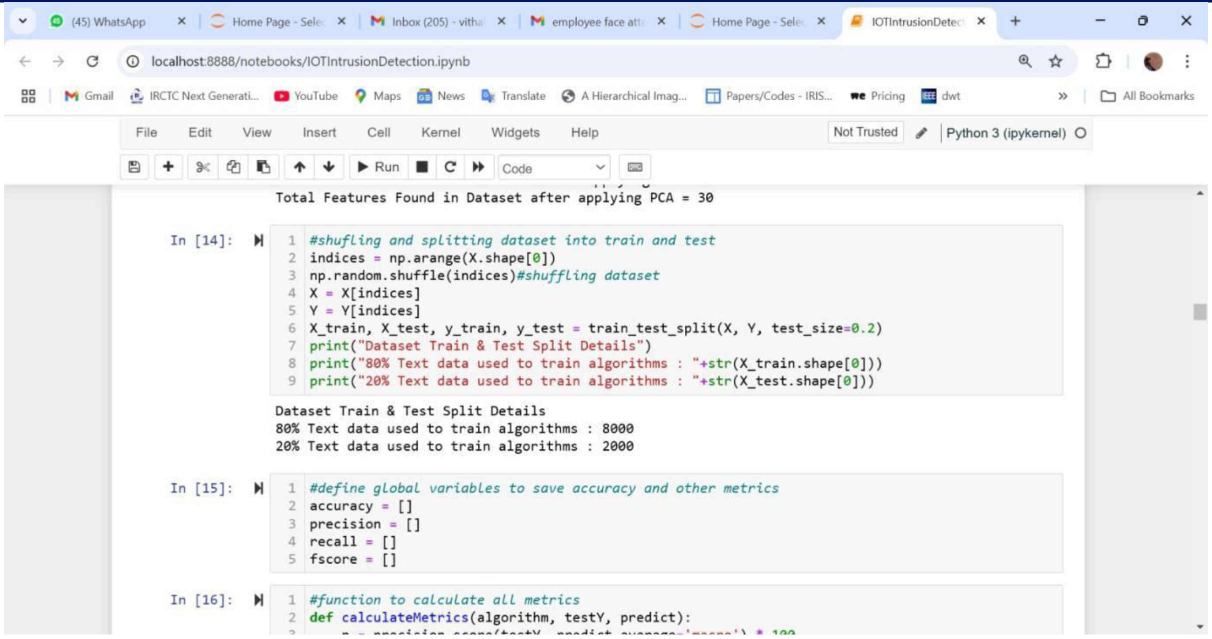
```
MLP Accuracy  : 100.0
MLP Precision : 100.0
MLP Recall    : 100.0
MLP FSCORE    : 100.0
```

Figure 5: The results shown in the figure indicate that the GAN model combined with MLP and Softmax achieved an impressive 100% accuracy. Additional metrics, such as precision and recall, can be observed through the confusion matrix. In the confusion matrix, the x-axis represents the predicted labels, and the y-axis represents the true labels. The diagonal color boxes reflect the number of correct predictions, while the blue boxes off the diagonal represent incorrect predictions, which are very few in this case. In the ROC curve, the x-axis represents the False Positive Rate, while the y-axis represents the True Positive Rate. If the blue line falls below the orange line, it indicates that most predictions are incorrect or false. Conversely, if the blue line is above the orange line, it suggests that the predictions are largely correct or true. In the shown ROC graph, only a few predictions fall below the orange line.



```
        lengths = tf.norm(inputs, axis=-1, keepdims=True)
        return (lengths / (1.0 + lengths ** 2)) * inputs

# Capsule Network Model
def create_capsule_network(input_shape, num_classes):
    inputs = layers.Input(shape=input_shape)
    x = layers.Conv2D(64, (3, 3), activation='relu')(inputs)
    x = layers.Conv2D(128, (3, 3), activation='relu')(x)
    x = layers.Flatten()(x)
    capsules = CapsuleLayer(num_capsules=num_classes, dim_capsules=16)(x)
    outputs = layers.Lambda(lambda z: tf.norm(z, axis=-1))(capsules)

    model = models.Model(inputs, outputs)
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'
    return model

# Initialize and train the Capsule Network
capsule_network = create_capsule_network((28, 28, 1), 10)
capsule_network.fit(X_train, y_train, batch_size=32, epochs=5, validation_split=0.2)

# Predict and evaluate
predicted_classes = tf.argmax(capsule_network.predict(X_test), axis=-1).numpy()
print("Accuracy:", accuracy_score(y_test, predicted_classes))
print("Classification Report:\n", classification_report(y_test, predicted_classes))
print("Confusion Matrix:\n", confusion_matrix(y_test, predicted_classes))
```

Fig 6 In above screen defining deep learning models GAN and MLP layer for training and you can see LSTM layer at '======dashed' lines so we are combining both GAN and MLP as hybrid algorithm and after executing above block will get below output


localhost:8888/notebooks/IOTIntrusionDetection.ipynb
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code

```
Total Features Found in Dataset after applying PCA = 30
```

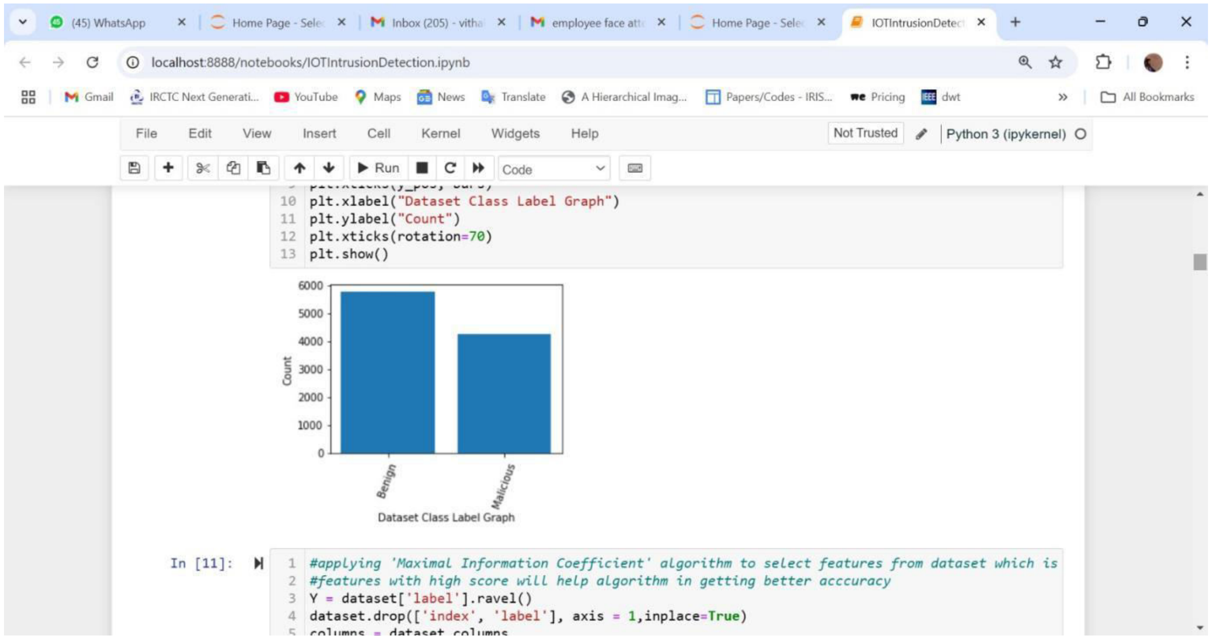
```
In [14]:
#shufling and splitting dataset into train and test
indices = np.arange(X.shape[0])
np.random.shuffle(indices)#shuffling dataset
X = X[indices]
Y = Y[indices]
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
print("Dataset Train & Test Split Details")
print("80% Text data used to train algorithms : "+str(X_train.shape[0]))
print("20% Text data used to train algorithms : "+str(X_test.shape[0]))
```

```
Dataset Train & Test Split Details
80% Text data used to train algorithms : 8000
20% Text data used to train algorithms : 2000
```

```
In [15]:
#define global variables to save accuracy and other metrics
accuracy = []
precision = []
recall = []
fscore = []
```

```
In [16]:
#function to calculate all metrics
def calculateMetrics(algorithm, testY, predict):
```

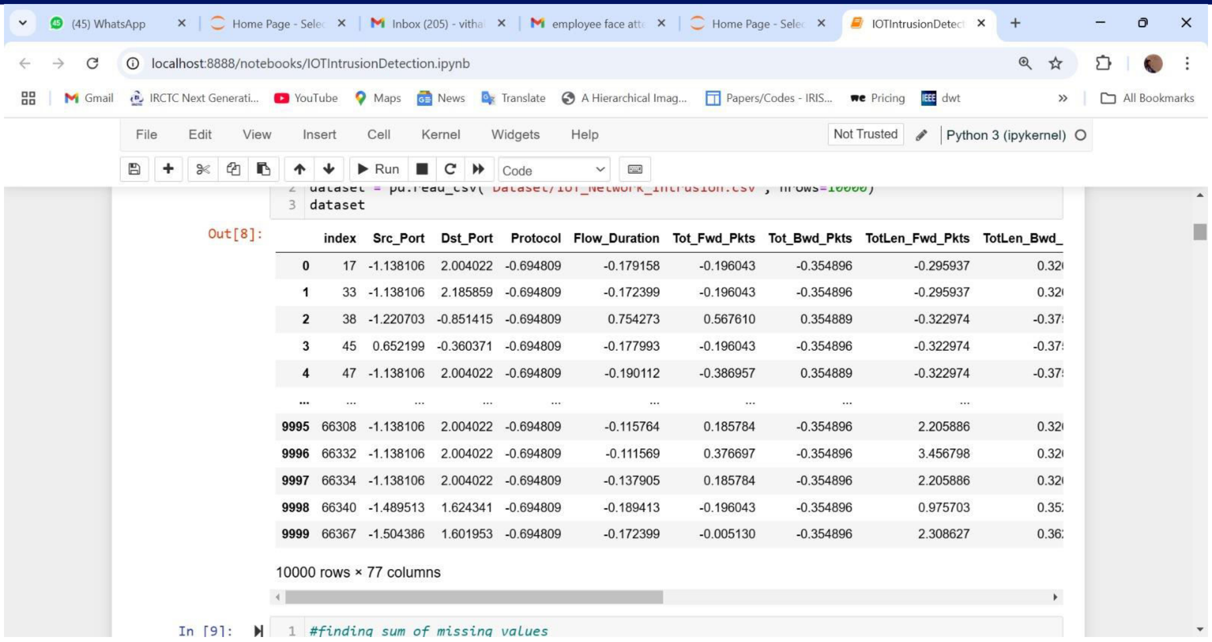
Fig 7. In above screen applying pre-processing techniques like Normalization, shuffling and splitting dataset into train and test where application using 80% dataset for training and 20% for testing and then in blue colour text displaying training and testing size images


localhost:8888/notebooks/IOTIntrusionDetection.ipynb
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
6000
5000
4000
3000
2000
1000
0
Count
Benign
Malicious
Dataset Class Label Graph

```
plt.xlabel("Dataset Class Label Graph")
plt.ylabel("Count")
plt.xticks(rotation=70)
plt.show()
```

```
In [11]:
#applying 'Maximal Information Coefficient' algorithm to select features from dataset which is
#features with high score will help algorithm in getting better acccuracy
Y = dataset['label'].ravel()
dataset.drop(['index', 'label'], axis = 1,inplace=True)
```

Fig 8. In above screen displaying graphs of different labels and number of images found in that label



```
dataset = pd.read_csv('Dataset/IoT_Network_Intrusion.csv', nrows=10000)
dataset
```

Out[8]:

	index	Src_Port	Dst_Port	Protocol	Flow_Duration	Tot_Fwd_Pkts	Tot_Bwd_Pkts	TotLen_Fwd_Pkts	TotLen_Bwd_
0	17	-1.138106	2.004022	-0.694809	-0.179158	-0.196043	-0.354896	-0.295937	0.32
1	33	-1.138106	2.185859	-0.694809	-0.172399	-0.196043	-0.354896	-0.295937	0.32
2	38	-1.220703	-0.851415	-0.694809	0.754273	0.567610	0.354889	-0.322974	-0.37
3	45	0.652199	-0.360371	-0.694809	-0.177993	-0.196043	-0.354896	-0.322974	-0.37
4	47	-1.138106	2.004022	-0.694809	-0.190112	-0.386957	0.354889	-0.322974	-0.37
...	...	...	...	...	...	...	...	...	
9995	66308	-1.138106	2.004022	-0.694809	-0.115764	0.185784	-0.354896	2.205886	0.32
9996	66332	-1.138106	2.004022	-0.694809	-0.111569	0.376697	-0.354896	3.456798	0.32
9997	66334	-1.138106	2.004022	-0.694809	-0.137905	0.185784	-0.354896	2.205886	0.32
9998	66340	-1.489513	1.624341	-0.694809	-0.189413	-0.196043	-0.354896	0.975703	0.35
9999	66367	-1.504386	1.601953	-0.694809	-0.172399	-0.005130	-0.354896	2.308627	0.36

10000 rows × 77 columns

```
In [9]: #finding sum of missing values
```

Fig 9. In above screen defining function to find and display different labels found in dataset

CONCLSUION

This study presents a deep learning-based framework for detecting cyberattacks in IoT environments, leveraging Generative Adversarial Networks (GANs), Capsule Networks, and Multi-Layer Perceptrons (MLPs). GANs address the challenge of data imbalance by generating synthetic attack data, enabling the model to better recognize malicious patterns. Capsule Networks enhance the model's ability to capture complex relationships within the data, facilitating the detection of sophisticated and evolving attack strategies. Finally, MLPs serve as the classifier, accurately distinguishing between normal and malicious activities. Experimental results on benchmark IoT datasets show that the proposed framework significantly improves detection accuracy, offering a scalable and effective solution for protecting IoT networks from emerging cyber threats. This approach underscores the promising potential of deep learning techniques in providing dynamic and adaptive security for IoT ecosystems.

REFERENCES

1. Parra, Gonzalo De La Torre, Paul Rad, Kim-Kwang Raymond Choo, and Nicole Beebe. "Distributed Deep Learning for Detecting IoT Attacks." *Journal of Network and Computer Applications*, vol. 163, 2020, p. 102662.
2. Abu Al-Haija, Qasem, and Saleh Zein-Sabatto. "A Deep Learning-Based Detection and Classification System for Cyber-Attacks in IoT Networks." *Electronics*, vol. 9, no. 12, 2020, p. 2152.

3. Praveenchandar, J., D. Vetrithangam, S. Kaliappan, M. Karthick, Naresh Kumar Pegada, Pravin P. Patil, S. Govinda Rao, and Syed Umar. "IoT-Based Monitoring of Harmful Toxic Gases and Fault Detection Using Deep Learning on Sensor Data." *Scientific Programming*, vol. 2022, no. 1, 2022, p. 7516328.
4. Rajesh, M., Rajiv Vincent, Sakshi Kathuria, Bhavana Jamalpur, Thirupathi Durgam, and Tarun Jaiswal. "Designing Deep Learning Models for Identifying Harmful Attack Activities in Industrial IoT." In *Proceedings of the 2023 IEEE Uttar Pradesh Section International Conference on Electrical, Electronics, and Computer Engineering (UPCON)*, vol. 10, pp. 1765-1771. IEEE, 2023.
5. Liang, Fan, William Grant Hatcher, Weixian Liao, Weichao Gao, and Wei Yu. "Machine Learning for IoT Security: Opportunities, Challenges, and Risks." *IEEE Access*, vol. 7, 2019, pp. 158126-158147.
6. Shafiq, Muhammad, Zhihong Tian, Yanbin Sun, Xiaojiang Du, and Mohsen Guizani. "Effective Machine Learning Algorithms for Bot-IoT Attack Traffic Identification in Smart Cities." *Future Generation Computer Systems*, vol. 107, 2020, pp. 433-442.
7. Saheed, Y. K., and M. O. Arowolo. "Efficient Cyber-Attack Detection in the Internet of Medical Things: A Deep Recurrent Neural Network and Machine Learning Approach." *IEEE Access*, vol. 9, 2021, pp. 161546-161554, doi: 10.1109/ACCESS.2021.3128837.
8. Ullah, Farhan, Hamad Naeem, Sohail Jabbar, Shehzad Khalid, Muhammad Ahsan Latif, Fadi Al-Turjman, and Leonardo Mostarda. "Detection of Cybersecurity Threats in IoT Using Deep Learning Techniques." *IEEE Access*, vol. 7, 2019, pp. 124379-124389.