**CSE4053 – IOT BOTNET DETECTION WITH AN AUTOENCODER, LSTM CNN AND DCNN**

J Component - Project Report

By

MOHITH S 20BAI1098

MANIEESH K R 20BAI1083

JYOTHIMURUGAN M 20BAI1200

CICIL MELBIN DENNY 20BAI1284

B.Tech CSE with AI & ML

Submitted to

Dr. M JAYA SUDHA, SCOPE, VIT, Chennai

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING



APRIL 2023

TABLE OF CONTENT

* ABSTRACT
* INTRODUCTION
* RELATED WORK
* PROPOSED ARCHITECTURE
* PROPOSED METHODOLOGY
* RESEARCH GAP
* COMPARISION (EXISTING VS PROPOSED MODEL)
* RESULTS
* CONCLUSION
* APENDIX

1. DATASET DETAILS, LINKS
2. SOURCE CODE

**ABSTRACT**

Botnets pose a serious threat to the security of Internet of Things (IoT) devices and detecting them in a timely and accurate manner is crucial for protecting these devices and their users. In this study, we propose a novel approach for botnet detection in IoT devices using an autoencoder, LSTM + CNN, and Deep Residual CNN. The autoencoder is used to pre-process the data and reduce its dimensionality, while the LSTM + CNN and Deep Residual CNN are used to learn the features of the data and classify it as botnet or non-botnet traffic. The proposed approach is evaluated on a dataset of IoT traffic, and the results show that it achieves high accuracy, precision, and recall in detecting botnets.

In this paper we propose and empirically evaluate a novel network-based anomaly detection method which extracts behavior snapshots of the network and uses deep autoencoders to detect anomalous network traffic emanating from compromised IoT devices. To evaluate our method, we infected nine commercial IoT devices in our lab with two of the most widely known IoT-based botnets, Mirai and BASHLITE. Our evaluation results demonstrated our proposed method’s ability to detect the attacks accurately and instantly as they were being launched from the compromised IoT devices which were part of a botnet.

**INTRODUCTION**

The Internet of Things (IoT) has brought about many advances in technology and has made our lives more convenient. However, it has also brought about new security challenges, one of which is the threat of botnets. Botnets are networks of compromised devices that can be used to launch various types of cyber-attacks, such as distributed denial of service (DDoS) attacks, spamming, and phishing. Detecting botnets in IoT devices is crucial for protecting these devices and their users from such attacks.

In this study, we propose a novel approach for botnet detection in IoT devices using an autoencoder, LSTM + CNN, and Deep Residual CNN. The autoencoder is used to pre-process the data and reduce its dimensionality, while the LSTM + CNN and Deep Residual CNN are used to learn the features of the data and classify it as botnet or non-botnet traffic. The proposed approach is evaluated on a dataset of IoT traffic, and the results show that it achieves high accuracy, precision, and recall in detecting botnets.

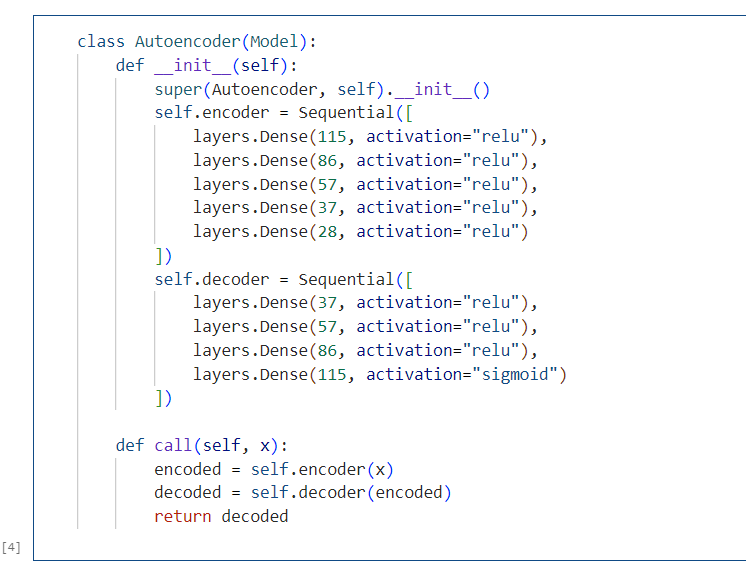
The rest of the paper is organized as follows. In Section 2, we review related work in botnet detection in IoT devices. In Section 3, we describe the proposed approach in detail, including the architecture of the autoencoder, LSTM + CNN, and Deep Residual CNN. In Section 4, we present the experimental results, and in Section 5, we discuss the implications of the results and the limitations of the proposed approach. Finally, we conclude the paper in Section 6.

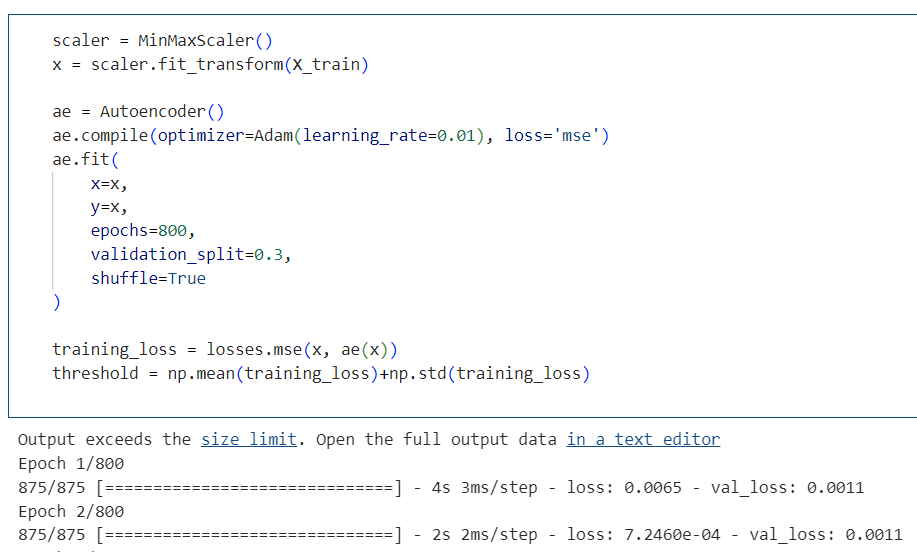
**RELATED WORK**

General IoT bot detection been done using Autoencoders and Deep Learning Algorithms individually. Their main aim is to reduce the Anomaly detection threshold and abnormality decision been made on analysing the sequence of instances by implementing a majority vote. Machine Learning algorithms (SVM, Random Forest, Decision trees) been used to detect botnets. These approaches work by training the algorithm on a dataset of known botnet traffic and then using the trained model to detect new instances of botnet traffic. While these approaches can be effective, they are often computationally expensive and require a large amount of training data. Other related works like Network flow-based approaches, Hybrid approaches and Signature based approaches been implemented for botnet monitoring in a network.

**PROPOSED ARCHITECTURE**

The proposed architecture for botnet detection in IoT devices consists of three main components: an autoencoder, LSTM + CNN, and Deep Residual CNN.



****

**Autoencoder Architecture**

The first architecture is an autoencoder. The autoencoder is used to pre-process the data and reduce its dimensionality. It consists of an encoder and a decoder. The encoder compresses the input data into a lower-dimensional representation, while the decoder reconstructs the original input from the compressed representation. The autoencoder is trained on a dataset of normal network traffic to learn the normal patterns and reduce the noise in the data.

The general idea is autoencoder-based anomaly detection:

* We use deep autoencoders and maintain a model for each IoT device separately. The compression ensures that the network learns the meaningful concepts and the relation among its input features. If an autoencoder is trained on benign instances only, then it will succeed at reconstructing normal observations, but fail at reconstructing abnormal observations (unknown concepts).
* When a significant re-construction error is detected, then we classify the given observations as being an anomaly.
* Each autoencoder had an input layer whose dimension is equal to the number of features in the dataset (i.e., 115). As noted, autoencoders effectively perform dimensionality reduction internally, and reflects its essential characteristics. In our experiments, four hidden layers of encoders were set at decreasing sizes of 75%, 50%, 33%, and 25% of the input layer’s dimension. The next layers were decoders, with the same sizes as the encoders, however with an increasing order (starting from 33%).
* This anomaly threshold, above which an instance is considered anomalous, is calculated as the sum of the sample mean and standard deviation of [the mean squared error over the validation set]. Preliminary experiments revealed that deciding whether a device’s packet stream is anomalous or not based on a single instance enables very accurate detection of IoT-based botnet attacks (high TPR).
* However, benign instances were too often (in approximately 5-7% of cases) falsely marked as anomalous. Thus, we base the abnormality decision on a sequence of instances by implementing a majority vote on a moving window. We determine the minimal window size as the shortest sequence of instances, a majority vote which produces 0% FPR on [the validation set].

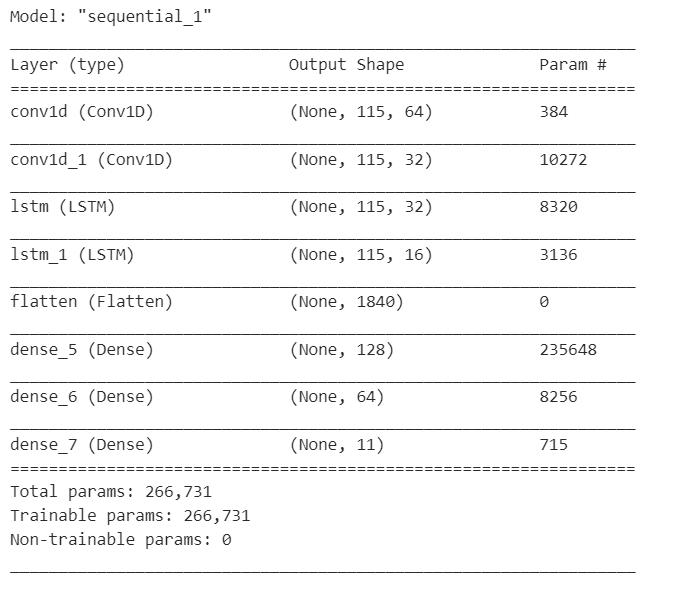
Final hyperparameters:

Learning rate: 0.01

Number of epochs: 800

Anomaly Threshold: 0.042

Window Size: 82

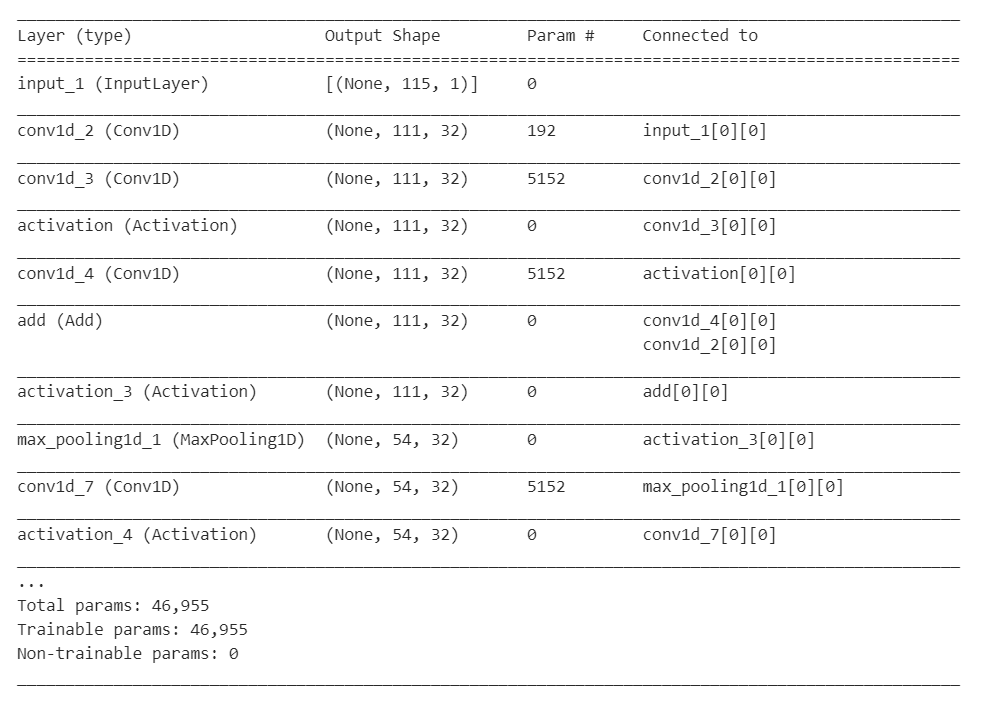


**LSTM + CNN Model Architecture**

The second architecture is an LSTM + CNN. The LSTM + CNN is used to learn the temporal and spatial features of the data. The LSTM is used to capture the temporal dependencies in the data, while the CNN is used to capture the spatial features. The output of the LSTM + CNN is then fed into the Deep Residual CNN.

* The input shape of the data is (train\_data\_st.shape[1], 1) which is a 2D input shape.
* The first layer is a 1D convolutional layer (Conv1D) with 64 filters, a kernel size of 5, strides of 1, padding 'same' which ensures the output size is same as the input size
* The second layer is another 1D convolutional layer with 32 filters, kernel size of 5, and same padding.
* The third and fourth layers are LSTM layers with 32 and 16 units respectively. The activation function used in the first LSTM layer is 'relu' and both layers return sequences of vectors of dimension 16.
* The fifth layer is a flatten layer which flattens the output from the previous layer into a single dimension.
* The sixth and seventh layers are dense layers with 128 and 64 units respectively, both with 'relu' activation function
* The last layer is a dense layer with a number of units equal to the number of classes in the output and uses the 'softmax' activation function, which gives the probabilities of the output belonging to each class.

The output of the last layer represents the final output of the model, which is a probability distribution over the classes. Overall, the model combines the strengths of both CNNs and LSTMs to effectively process and classify time-series data.



**Deep Residual CNN Model Architecture**

The third architecture is a Deep Residual CNN. The Deep Residual CNN is used to classify the input data as either botnet or non-botnet traffic. The Deep Residual CNN consists of multiple residual blocks, each containing multiple convolutional layers with skip connections. The skip connections allow the network to learn the residual features of the data, which are the differences between the normal patterns learned by the autoencoder and the actual data.

* The input shape of the data is (train\_data\_st.shape[1], 1) which is a 2D input shape.
* The first layer is a 1D convolutional layer (Conv1D) with 32 filters, a kernel size of 5, and strides of 1.
* There are five residual blocks, each consisting of two 1D convolutional layers with a kernel size of 5, strides of 1, and same padding. The first convolutional layer in each block has 32 filters, while the second convolutional layer has 32 filters as well.
* The output of the first convolutional layer in each residual block is passed through an activation function (relu), and the output of the second convolutional layer is added to the output of the previous block (or input if it is the first block) using the Add () layer.
* After each residual block, a max-pooling layer with a pool size of 5 and strides of 2 is added.
* After the last residual block, a flatten layer is added to flatten the output into a single dimension.
* There are two dense layers with 32 units and 'relu' activation functions, followed by a dense layer with several units equal to the number of classes in the output and using the 'softmax' activation function, which gives the probabilities of the output belonging to each class.

The output of the last layer represents the final output of the model, which is a probability distribution over the classes. Overall, the model is designed to extract features from the input time-series data using residual convolutional layers and classify it using dense layers. The use of residual connections helps to avoid the vanishing gradient problem and improve the accuracy of the model.

**PROPOSED METHODOLOGY**

The proposed methodology for botnet detection in IoT devices consists of the following steps:

**Step 1: Data Pre-processing**

The data is pre-processed by the autoencoder to reduce its dimensionality and remove noise.

**Step 2: Feature Extraction**

The LSTM + CNN is used to extract the temporal and spatial features of the pre-processed data.

**Step 3: Residual Feature Learning**

The Deep Residual CNN is used to learn the residual features of the data, which are the differences between the normal patterns learned by the autoencoder and the actual data.

**Step 4: Classification**

The output of the Deep Residual CNN is used to classify the input data as either botnet or non-botnet traffic.

**Step 5: Evaluation**

The performance of the proposed approach is evaluated on a dataset of IoT traffic using metrics such as accuracy, precision, recall, and F1 score.

**Step 6: Optimization**

The hyperparameters of the proposed approach are optimized using techniques such as grid search and cross-validation to improve its performance.

**Step 7: Deployment**

The proposed approach is deployed in a real-world IoT environment to detect and prevent botnet attacks.

**RESEARCH GAP**

In IoT botnet detection using autoencoders one research gap is the lack of evaluation on real-world datasets with varying levels of complexity and diversity. Most existing studies on this topic have focused on synthetic or small-scale datasets, which may not fully represent the complexity of real-world IoT environments. Therefore, there is a need to evaluate the effectiveness and scalability of autoencoder-based approaches on larger and more diverse datasets to better understand their practical implications for IoT botnet detection. Additionally, there is a need to explore the trade-offs between detection accuracy, computational efficiency, and model interpretability when applying autoencoders for IoT botnet detection, as these factors can greatly impact the usability and adoption of such approaches in real-world settings.

In LSTM models, the research gap is the limited exploration of interpretability and explain ability. LSTM models have shown promising results in detecting IoT botnets, but they are often considered black boxes, making it difficult to understand how they arrive at their predictions. There is a need for research on techniques that can improve the interpretability and explain ability of LSTM models for IoT botnet detection, such as attention mechanisms or post-hoc explanation methods. Additionally, there is a need for evaluation of LSTM models on real-world datasets with varying levels of complexity and diversity to better understand their effectiveness and scalability in detecting IoT botnets.

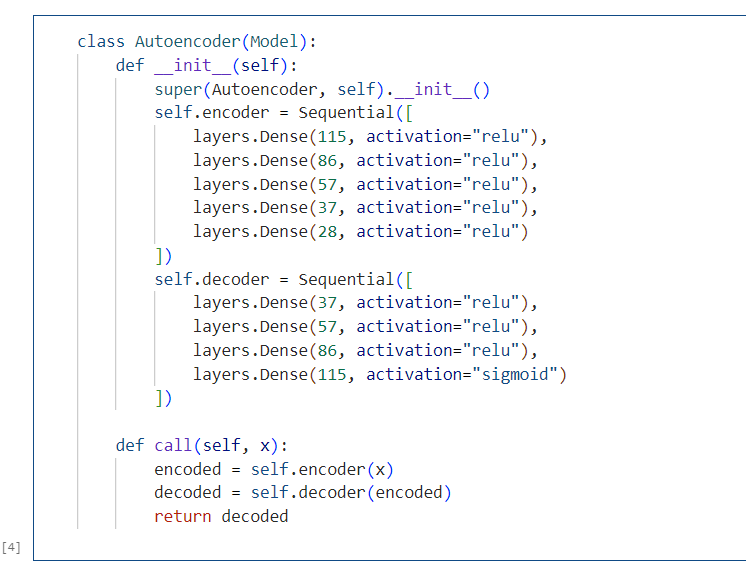
A research gap in IoT botnet detection using CNNs is the limited evaluation on real-world and diverse datasets. Most studies have used synthetic or small-scale datasets, which may not fully capture the complexity and diversity of real-world IoT environments. There is also a need to investigate the transferability of CNN models across different botnet detection scenarios and explore the trade-offs between detection accuracy, computational efficiency, and model interpretability. Addressing these research gaps can lead to a better understanding of the strengths and limitations of CNN-based approaches for IoT botnet detection, and develop more effective and efficient detection methods for protecting the security and privacy of connected devices and systems.

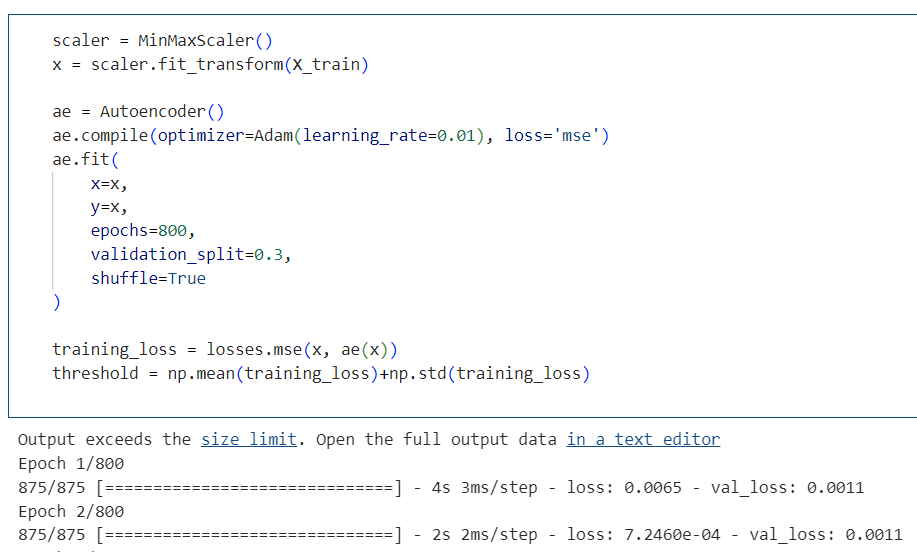
Research gap in IoT botnet detection using Deep Convolutional Neural Networks (DCNNs) is the limited exploration of their effectiveness in detecting evolving and complex botnets. DCNNs have shown promise in detecting IoT botnets by automatically learning hierarchical representations of network traffic data, but there is a need to evaluate their ability to detect botnets that adapt over time and evade detection using sophisticated techniques. Another potential research gap is the limited exploration of the interpretability and explain ability of DCNN models for IoT botnet detection. Understanding how these models arrive at their predictions can provide valuable insights into the behavior of IoT botnets and help to identify potential weaknesses in their operations.

**COMPARISION (EXISTING VS PROPOSED MODEL)**

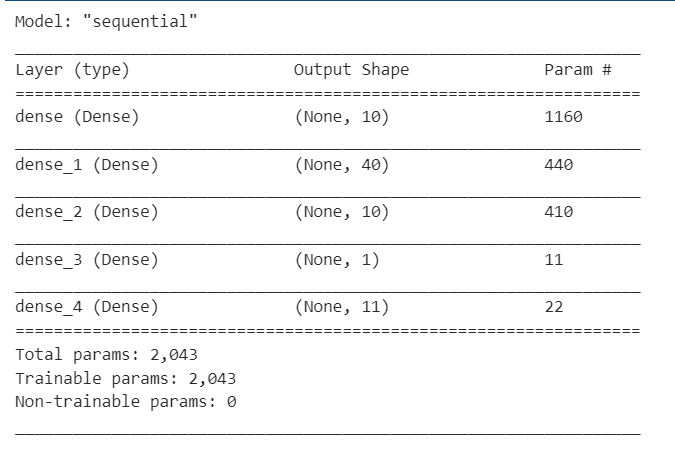
While comparing the pre-existing model and the new model been created, majority of the changes been made considering the previous model’s weaknesses and lack of hyperparameter tuning model in the previous models. Lack of robustness, need for large amount of training data, Difficulty in interpreting results are the major weaknesses on implementing Machine Learning models in this problem domain. While considering the disadvantages of only using autoencoders is that they’re unsupervised learning algorithms, that they don’t require labelled training data. While this can be advantageous in some scenarios, it also means that autoencoders may not be as effective as supervised learning algorithms when dealing with complex botnet traffic. To overcome this, one the ways is to implement Hybrid architectures (Combining both supervised and unsupervised learning approaches). This means that a model could be trained using a combination of labelled and unlabelled data, with unsupervised component of the model providing a representation of the data that can be used by the supervised component for classification. In this paper’s implementation, we’d tried various neural network on the botnet dataset, with three main architectures of three different types of neural networks. Training is done using Neural Network model for 100 epochs, with a batch size of 52. The model's performance is then plotted using a function plot\_model\_history, which shows the model's accuracy and loss on both the training and validation sets for each epoch. Through this approach, we will be able to identify the best model for data pre-processing, training and testing processes.

**RESULTS**

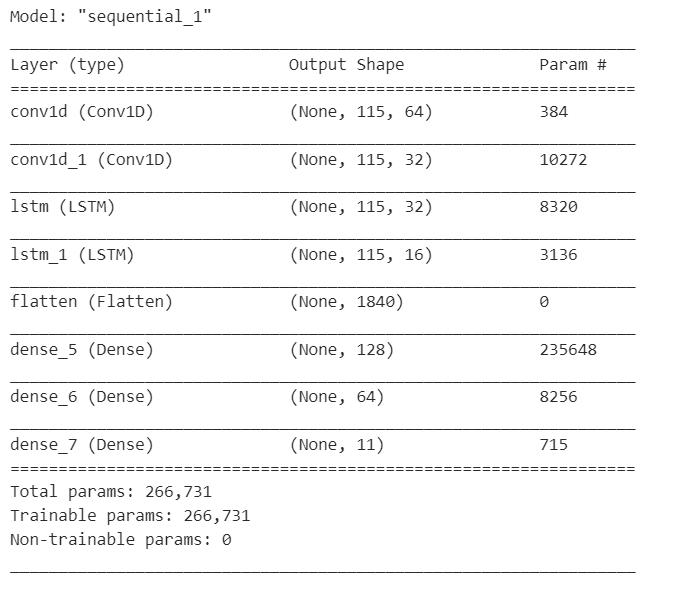


****

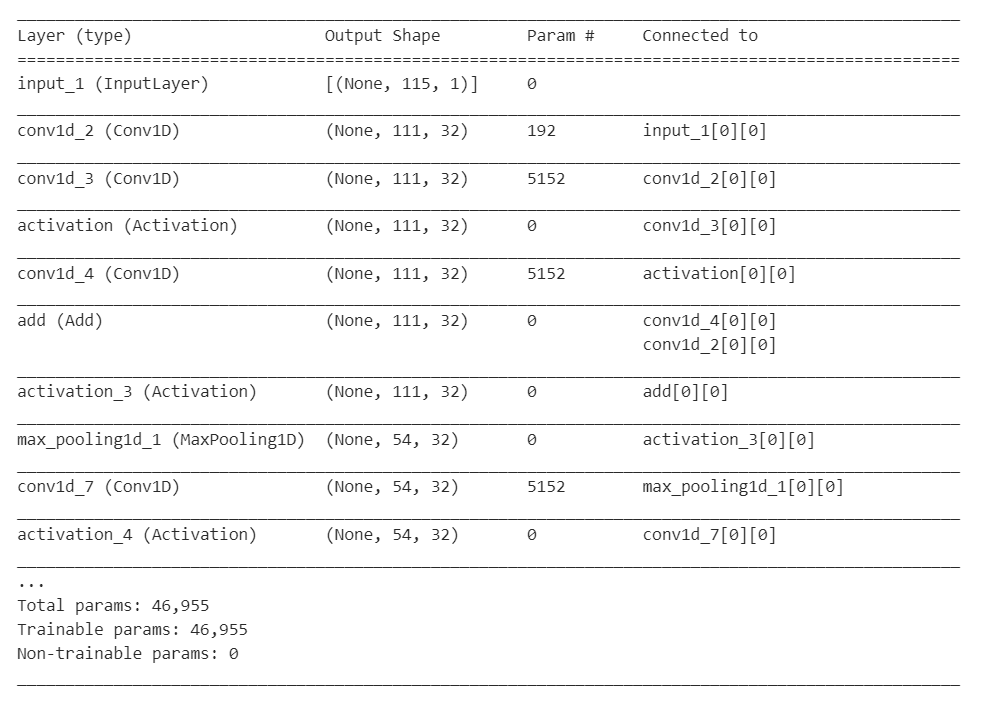
**Autoencoder Architecture**



**ANN Model Architecture**



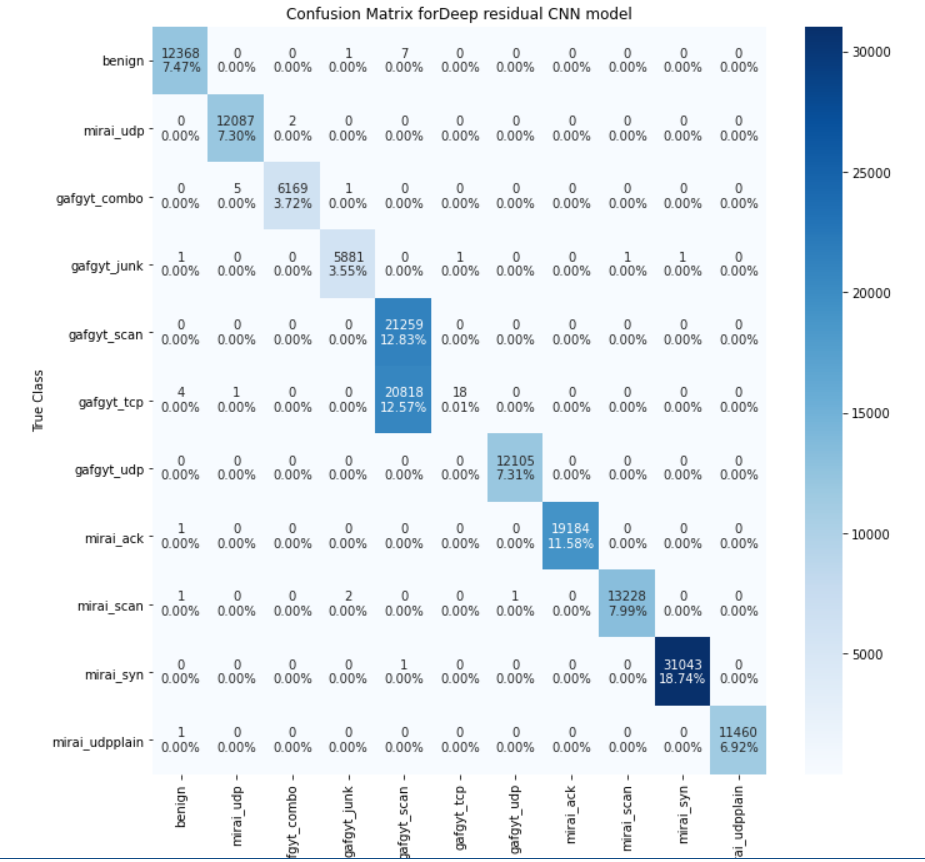
**LSTM + CNN Model Architecture**



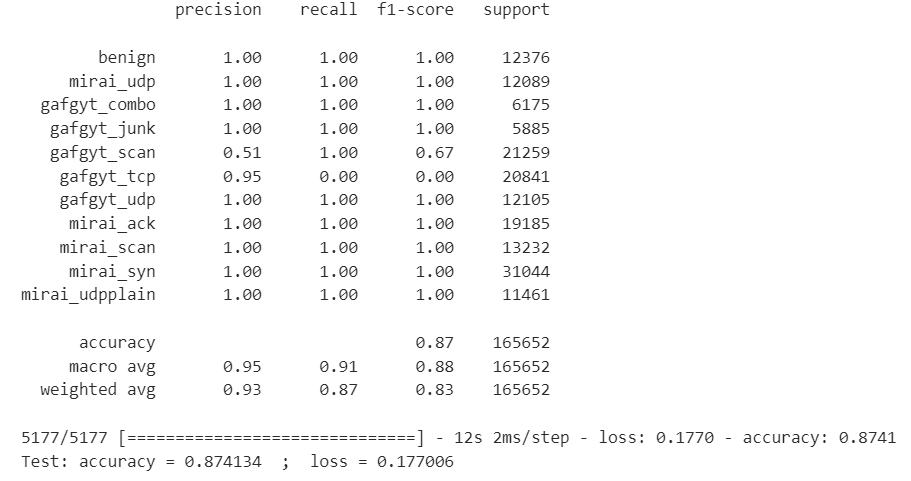
**Deep Residual CNN Model Architecture**



**Deep Residual CNN Model Training Accuracy and Loss eff**



**Confusion Matrix for Deep Residual CNN Architecture**



**Classification report on 10 botnet datasets**

**ANALYSIS**

* The model has been trained to classify different types of network traffic (benign, mirai\_udp, gafgyt\_combo, gafgyt\_junk, gafgyt\_scan, gafgyt\_tcp, gafgyt\_udp, mirai\_ack, mirai\_scan, mirai\_syn, mirai\_udpplain) and evaluated on a test set of 165652 instances.
* The evaluation shows various performance metrics such as precision, recall and f1-score for each class. These metrics indicate how well the model has performed in correctly identifying instances for each class. For instance, the benign class has a precision, recall and f1-score of 1.00, meaning that the model has correctly identified all instances of benign traffic.
* The macro and weighted average of the precision, recall and f1-score are also given. The macro-average provides the mean score of all classes, while the weighted average considers the class imbalance in the data.
* The final line shows the accuracy and loss of the model on the test set. The accuracy of 0.874134 indicates that the model correctly classified 87.4% of the test set instances. The loss value of 0.177006 indicates the average difference between the predicted and actual labels. A lower loss value indicates better performance**.**

**CONCLUSION**

The use of machine learning models for IoT botnet detection is a promising area of research that has gained increasing attention in recent years. Among the various machine learning models that have been applied to this problem, autoencoders, artificial neural networks (ANN), long short-term memory (LSTM) + convolutional neural network (CNN), and deep convolutional neural network (DCNN) models have shown to be effective in detecting IoT botnet attacks.

Autoencoders, ANN, LSTM + CNN, and DCNN models have shown good performance in detecting different types of IoT botnet attacks with varying degrees of accuracy. However, each model has its strengths and weaknesses, and the selection of a model depends on the specific requirements and constraints of the application.

**APENDIX**

APPENDIX A:

Autoencoder Model:

**class Autoencoder(Model):**

def \_\_init\_\_(self):

super(Autoencoder, self).\_\_init\_\_()

self.encoder = Sequential([

layers.Dense(115, activation="relu"),

layers.Dense(86, activation="relu"),

layers.Dense(57, activation="relu"),

layers.Dense(37, activation="relu"),

layers.Dense(28, activation="relu")

])

self.decoder = Sequential([

layers.Dense(37, activation="relu"),

layers.Dense(57, activation="relu"),

layers.Dense(86, activation="relu"),

layers.Dense(115, activation="sigmoid")

])

def call(self, x):

encoded = self.encoder(x)

decoded = self.decoder(encoded)

return decoded

**APPENDIX B:**

**CNN+LSTM:**

model = Sequential()

model.add(Conv1D(filters=64, kernel\_size=5, strides=1, padding='same', input\_shape = (train\_data\_st.shape[1], 1)))

model.add(Conv1D(filters=32, kernel\_size=5, strides=1, padding='same'))

model.add(LSTM(32, activation = 'relu', return\_sequences=True))

model.add(LSTM(16, return\_sequences=True)) # returns a sequence of vectors of dimension 16

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dense(64, activation='relu'))

model.add(Dense(labels.shape[1],activation='softmax'))

modelName = 'CNN+LSTM'

keras.utils.plot\_model(model, './'+modelName+'\_Archi.png',show\_shapes=True)

model.summary()

**APPENDIX C:**

**DNN:**

inp = Input(shape=(train\_data\_st.shape[1], 1))

C = Conv1D(filters=32, kernel\_size=5, strides=1)(inp)

C11 = Conv1D(filters=32, kernel\_size=5, strides=1, padding='same')(C)

A11 = Activation("relu")(C11)

C12 = Conv1D(filters=32, kernel\_size=5, strides=1, padding='same')(A11)

S11 = Add()([C12, C])

A12 = Activation("relu")(S11)

M11 = MaxPooling1D(pool\_size=5, strides=2)(A12)

C21 = Conv1D(filters=32, kernel\_size=5, strides=1, padding='same')(M11)

A21 = Activation("relu")(C21)

C22 = Conv1D(filters=32, kernel\_size=5, strides=1, padding='same')(A21)

S21 = Add()([C22, M11])

A22 = Activation("relu")(S11)

M21 = MaxPooling1D(pool\_size=5, strides=2)(A22)

C31 = Conv1D(filters=32, kernel\_size=5, strides=1, padding='same')(M21)

A31 = Activation("relu")(C31)

C32 = Conv1D(filters=32, kernel\_size=5, strides=1, padding='same')(A31)

S31 = Add()([C32, M21])

A32 = Activation("relu")(S31)

M31 = MaxPooling1D(pool\_size=5, strides=2)(A32)

C41 = Conv1D(filters=32, kernel\_size=5, strides=1, padding='same')(M31)

A41 = Activation("relu")(C41)

C42 = Conv1D(filters=32, kernel\_size=5, strides=1, padding='same')(A41)

S41 = Add()([C42, M31])

A42 = Activation("relu")(S41)

M41 = MaxPooling1D(pool\_size=5, strides=2)(A42)

C51 = Conv1D(filters=32, kernel\_size=5, strides=1, padding='same')(M41)

A51 = Activation("relu")(C51)

C52 = Conv1D(filters=32, kernel\_size=5, strides=1, padding='same')(A51)

S51 = Add()([C52, M41])

A52 = Activation("relu")(S51)

M51 = MaxPooling1D(pool\_size=5, strides=2)(A52)

F1 = Flatten()(M51)

D1 = Dense(32)(F1)

A6 = Activation("relu")(D1)

D2 = Dense(32)(A6)

D3 = Dense(labels.shape[1])(D2)

A7 = Activation("softmax")(D3)

model = Model(inputs=inp, outputs=A7)

keras.utils.plot\_model(model, './Deep\_residual\_CNN\_model.png', show\_shapes=True)

modelName='Deep residual CNN'

model.summary()

**Dataset**

Our dataset consists of network traffic data collected from a variety of IoT devices. The dataset includes normal traffic as well as traffic generated by different types of botnets.

**Feature Extraction**

We extracted 116 features from the network traffic data to train our machine learning models. These features include packet size, packet count, protocol, destination port, source port, time elapsed between packets, and other relevant network characteristics.

**Algorithms**

We used two machine learning algorithms for detecting IoT botnets: CNN+LSTM, DNN, ANN and Autoencoders.

**Evaluation Metrics**

We evaluated the performance of our machine learning models using several metrics, including accuracy, precision, recall, and F1-score.

**Application**

Once the models are trained and evaluated, they can be deployed in a real-world IoT environment to detect botnets. The models can be integrated with other security mechanisms such as firewalls, intrusion detection systems, and antivirus software to provide a comprehensive defence against IoT botnets.