

IOT BOTNET ATTACK DETECTION AND CLASSIFICATION USING MACHINE LEARNING

ABSTRACT

The growing adoption of Internet-of-Things devices brings with it the increased participation of said devices in botnet attacks, and as such novel methods for IoT botnet attack detection are needed. This work demonstrates that deep learning models can be used to detect and classify IoT botnet attacks based on network data in a device agnostic way and that it can be more accurate than some more traditional machine learning methods, especially without feature selection. Furthermore, this work shows that the opaqueness of deep learning models can be mitigated to some degree with Local Interpretable Model-Agnostic Explanations technique.

TABLE OF CONTENT

Content

1. Introduction.....	1
1.1. Overview.....1
2. Software Requirement.....	2-12
2.1. Problem	
Definition.....	2
2.2. Modules.....2
2.2.1 Test	
Bed.....	...2
2.2.1.1 Lab	
Setup.....2
2.2.1.2 Port	
mirroring.....3
2.2.2 Tool.....4
2.2.2.1 Wireshark.....4
2.2.3 Dataset.....5
2.2.3.1 Data	
Description.....	

.....	5-6
2.2.4 Anomaly Detection.....	7
2.2.5 Deep Auto-encoder.....	7-9
2.3. Attack Classification.....	9-11
2.3.1 Deep Neural Network.....	9-10
2.4. Evaluation metrics.....	11-12
3. Software Design.....	13
3.1. Control Flow Diagram.....	13
4. Requirements.....	13-14
4.1. Software.....	13
4.2. Hardware.....	14
5. Coding.....	14-12
5.1. Anomaly Detection.....	14-18
5.1.1 Train Model	
5.1.1.1 Libraries Required	

5.1.1.2 Train Function	
5.1.1.3 Create_model function	
5.1.2 Test	
5.1.2.1 Libraries Required	
5.1.2.2 load_mal_data()	
5.1.2.3 test	
5.2. Classification.....	19-23
5.2.1 Train model	
5.2.1.1 Libraries Required	
5.2.1.2 load_data	
5.2.1.3 create_model()	
5.2.1.4 Train_with_data()	
5.2.2 Test	
5.2.2.1 Libraries Required	
5.2.2.2 test_with_data	
5.3. Result.....	...24
5.3.1 Feature Scoring.....	24
5.3.2 Accuracy.....	24

5.3.3 Result.....	24
6. Testing.....	25-26
6.1. Black Box Testing	
7. Output Screens.....	26-39
7.1. Anomaly Detection	
7.1.1 models ouput console with 15 features	
7.1.2 models ouput console with 115 features	
7.2. Classification	
7.2.1 Models output console	
8. Conclusions.....	40
9. Further Enhancements.....	40
10. References.....	41
Appendix 1 – Fisher’s score calculation procedure	42
Appendix 2 – Auto encoder model creation function	43
Appendix 3–Deep Neural Network model creation function.....	44

LIST OF FIGURES

2.1 Lab setup for detecting IoT botnet attack

2.2	Port mirroring	
2.3	Data Description	
2.4	Deep Autoencoder	
2.5	Deep Neural Network	
3.1	Potential attack detection, classification and explanation pipeline	
5.1	Libraries	
5.2	Train Function	
5.3	Create model function	
5.4	Libraries for test	
5.5	Test_with_data()	
5.6	Libraries	
5.7	Load Data	
5.8	Create_model Function	
5.9	train function	
5.10	Libraries	
5.11	Test	
6.1	Black Box Testing	
6.2	Black Box Test	
7.1	train output screen with top 15 features	
7.2	test output screen with top 15 features	1
7.3	Test output screen with top 15 features	2
7.4	Test output screen with top 15 features	3

- 7.5 train output screen with top 15 features
- 7.6 test output screen with 115 features 1
- 7.7 test output screen with 115 features 2
- 7.8 test output screen with 115 features 3
- 7.9 Train output screen with top 5 feature 1
- 7.10 Train output screen with top 5 feature 2
- 7.11 Train output screen with top 5 feature 3
- 7.12 Test output screen with top 5 feature 1
- 7.13 Test output screen with top 5 feature 2
- 7.14 Test output screen with top 5 feature 3
- 7.15 Test output screen with top 5 feature 4 List

of Tables

Table 2.1 Confusion matrix structure

Table 5.1 Result for top 5 features

Table 5.2 Confusion matrix output

1. Introduction

1.1 Overview

In recent years the IoT (Internet-of-Things) market has experienced a dramatic growth, with the number of IoT devices connected to the network reaching 7 billion in 2018. This creates a motivation for the malware industry to infect such devices and use them for malicious purposes. In 2017 the number of DDoS (Distributed Denial of Service) attacks has increased by 91% thanks to IoT botnets. Such attacks are carried out by a group of infected machines (bots), forming a botnet under the control of the attacker through a C&C (Command and Control) server, against some other entity in the network, such as a corporation or a government. For example, in 2016 one such botnet under the name of Mirai has attacked multiple internet companies like Krebs on Security, OVH and Dyn. A typical botnet such as Mirai usually operates in multiple steps. Botnets spread starting with a scan phase in which they scan available networks for vulnerable devices. Access to these devices is then brute forced using some common credentials. In case of a successful access, the details of the compromised machine are sent to the attacker's server and the load phase begins, where malware is loaded and installed on a device. At this point the device becomes a part of the botnet network and will be operated through a C&C server .

IoT itself means extending internet connection to devices and appliances that in the past operated offline. Connecting them to the internet allows for remote control and monitoring. IoT devices find their use in many different domains, both in consumer and enterprise markets. For example, the concept of Smart Home heavily relies on IoT by connecting thermometers, light bulbs, security cameras and other devices to the network. This sheer variety of IoT devices and manufacturers contributes to the security challenges faced by the industry.

2. Software Requirement

2.1 Problem Definition

Traditional malware detection methods require a lot of manpower in order to analyze the threats and come up with detection rules, especially as the malware industry continuously adapts and evades new countermeasures. In this regard machine learning gives a lot of promise for the task. One possible approach to the problem of IoT security against botnets is based on the application of machine learning to the network traffic data.

2.2 Modules

For the purpose of our tasks we will employ two different architectures of neural networks: an Auto encoder for anomaly detection and a Classifier for classification.

On the technical side, code will be written in Python 3 with the use of modern frameworks for data handling and machine learning, such as Pandas, scikit-learn and Keras.

2.2.1 Test Bed

2.2.1.1 Lab Setup

To replicate a typical organizational data flow, we collected the traffic data from IoT devices that were connected via Wi-Fi to several access points, wire connected to a central switch which also connects to a router. For sniffing the network traffic, we performed port mirroring on the switch, and recorded the data using Wireshark. To evaluate our detection method as realistically as possible, we also deployed all of the components of two botnets in our isolated lab and used them to infect nine commercial IoT devices [1].

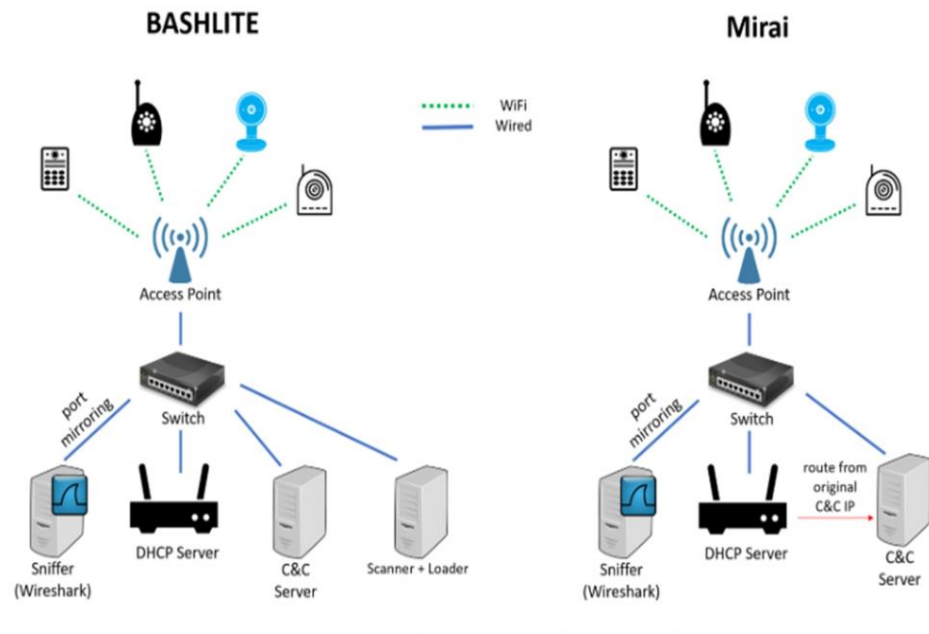


Fig.1.1 Lab setup for detecting IoT botnet attack

2.2.1.2 Port mirroring

Port Mirroring, also known as SPAN (Switched Port Analyzer), is a method of monitoring network traffic. With port mirroring enabled, the switch sends a copy of all network packets seen on one port (or an entire VLAN) to another port, where the packet can be analyzed.

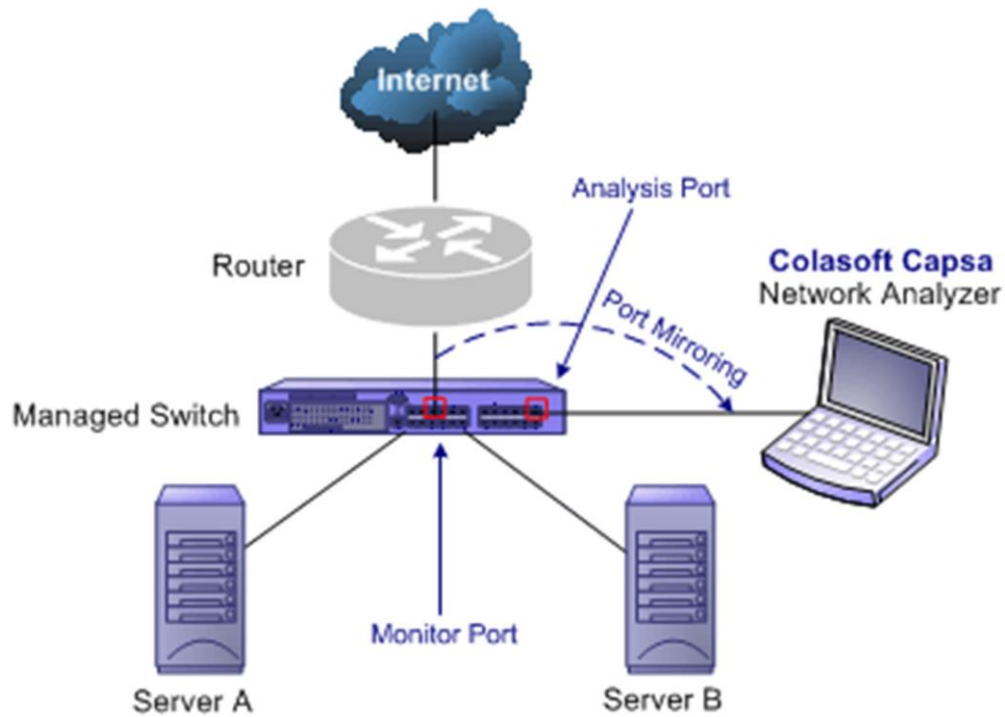


Fig.2.2: Port mirroring

2.2.2 Tool

2.2.2.1 Wireshark

Wireshark is the world's foremost and widely-used network protocol analyzer. It lets you see what's happening on your network at a microscopic level and is the de facto (and often de jure) standard across many commercial and non-profit enterprises, government agencies, and educational institutions. Wireshark development thrives thanks to the volunteer contributions of networking experts around the globe and is the continuation of a project started by Gerald Combs in 1998.

2.2.3 Dataset












The data was obtained experimentally in a laboratory environment by the research team at Ben-Gurion University of Negev. In this environment different IoT devices were connected to an isolated network using both WiFi and wired connections. Additionally, botnet components were installed in the network, such as C&C server. Then, using port mirroring at the internet switch, data was gathered using Wireshark for both normal traffic, when none of the devices were infected, and as well for malicious traffic, when devices were infected [3].

2.2.3.1 Data Description

The data comes from 9 IoT devices belonging to smart-home and security domains:

- Doorbells,
- Security cameras,
- Thermostat,
- Baby monitor.

Index of /ml/machine-learning-databases/00442

Name	Last modified	Size	Description
 Parent Directory		-	
 Danmini Doorbell/	2019-01-08 23:48	-	
 Ecobee Thermostat/	2019-01-08 23:53	-	
 Ennio Doorbell/	2019-01-08 23:54	-	
 N BaIoT dataset desc.>	2018-04-01 21:41	4.5K	
 Philips B120N10 Baby.>	2019-01-08 23:55	-	
 Provision PT 737E Se.>	2019-01-09 00:18	-	
 Provision PT 838 Sec.>	2019-01-10 23:47	-	
 Samsung SNH 1011 N W.>	2019-01-09 00:20	-	
 SimpleHome XCS7 1002.>	2019-01-09 00:29	-	
 SimpleHome XCS7 1003.>	2019-01-09 00:31	-	
 demonstrate structur.>	2018-03-19 07:08	1.7K	

Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips SVN/1.7.14 Phusion_Passenger/4.0.53 mod_perl/2.0.11 Perl/v5.16.3 Server at archive.ics.uci.edu Port 443

Fig. 2.3: Data Description

For this work all devices are combined into one dataset.

In total data consists of 115 features which are related to different information about the packets, aggregated in different ways.

The network statistics data that is included consists of: weight, or packet count; mean of packet size; variance of packet size; standard deviation of packet size; radius as root squared sum of two stream's variances; magnitude of two stream's means; covariance and Pearson's correlation coefficient.

Data is aggregated by:

- Source Host IP (H – feature name part in dataset file)
- Source MAC-IP (MI)
- Source and destination host IP (HH)
- Source and destination host and port (HpHp)
- Source and destination host traffic jitter (HH_jit)

Two types of botnet were installed in the laboratory environment: BASHLITE and Mirai.

These two are the most notable botnet families in the world of Linux based IoT. Further, Mirai data consists of 5 attack classes: SYN, ACK, UDP, UDPPplain, SCAN.

The total dataset consists of more than 5 million points.

Data set size –

- Normal - 5,55,932
- Mirai - 36,68,402
- BASHLITE - 10,32,056

2.2.4 Anomaly Detection

The task of detecting an attack can be framed as an anomaly or outlier detection problem. This is based on the assumption that the normal usage network traffic data and malware network data will differ in some dimensions, so that an algorithm will be able to make a distinction. There are different methods for anomaly detection, some of them, which were used previously in other works for the same task, are Local Outlier Factor, Isolation Forest and One class SVM.

2.2.5 Deep Auto-encoder

An autoencoder is an unsupervised learning technique for neural networks that learns efficient data representations (encoding) by training the network to ignore signal “noise.”

The autoencoder network has three layers: the input, a hidden layer for encoding, and the output decoding layer. Using back propagation, the unsupervised algorithm continuously trains itself by setting the target output values to equal the inputs. This forces the smaller hidden encoding layer to use dimensional reduction to eliminate noise and reconstruct the inputs.

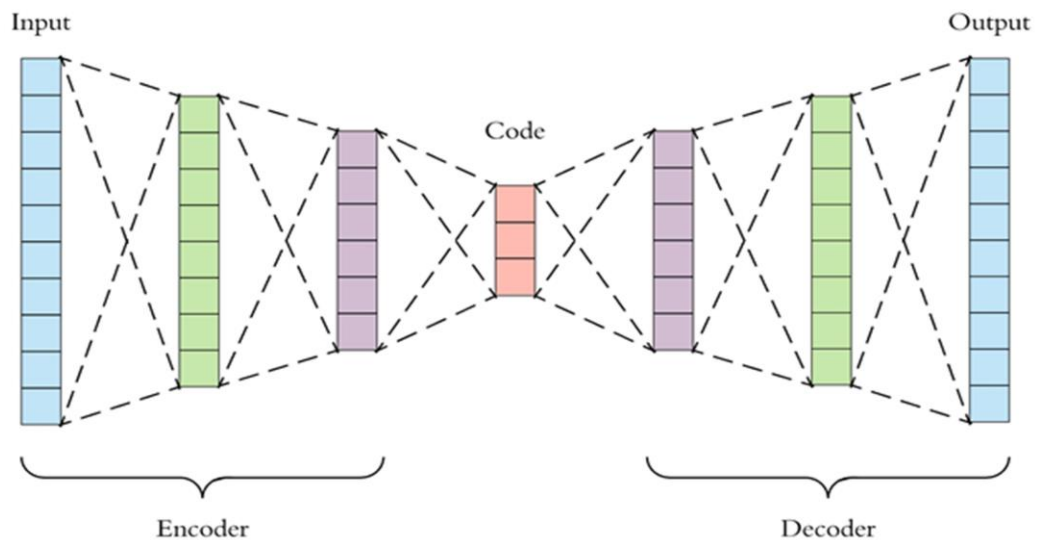


Fig. 2.5: Deep Autoencoder

This kind of network is composed of two parts :

1. Encoder

This is the part of the network that compresses the input into a latent-space representation. It can be represented by an encoding function $z=f(x)$.

2. Decoder

This part aims to reconstruct the input from the latent space representation. It can be represented by a decoding function $x'=g(z)$

$$Z = e(X)$$

$$X' = d(Z)$$

$$X \approx X'$$

Function Loss-

- a. Mean Square Error (mse) – Used generally used in continuous data.
- b. Binary Error- Used when data is in binary i.e. 0 and 1.

We would be using mse in our model. The code for model creation is given in Appendix 2.

2.3 Attack Classification

Having solved the problem of attack detection using autoencoders we then would like to inspect the data to find out what type of attack is it exactly. As the dataset consists of two botnet types: Mirai and BASHLITE, there will be these two classes, plus, in order to compare our performance with earlier work using traditional ML methods, we will consider normal data as a third class. Also, we will try to classify different Mirai attack types: ACK, SCAN, SYN, UDP, UDPPPlain.

In general, the task of classification appears to be the most popular application of machine learning and consequently there are lots of algorithms to solve it: SVMs, decision trees, random forests, k-nearest neighbors and others.

Apart from the neural network that will be applied in this work, of interest to briefly introduce decision trees and k-nearest neighbors, as this is what we will compare our results to.

Decision Tree algorithm learns the partitions of feature values for given classes. Main advantage of the algorithm is that the results are easily interpretable. Sometimes however, decision tree learners can produce overly complicated and biased solutions. k-Nearest Neighbors algorithm works on principle that the class of the data point is decided by the classes of k nearest neighbors to that point.

2.3.1 Deep Neural Network

We have attempted to solve the problem of classification with a deep neural network with two hidden layers each with 8 neurons. Unfortunately, there are no theoretical rules yet that would guide the choice of a neural network size for particular tasks. Mostly the approach seems to be to just

increase the number of layers or neurons until good accuracy is attained. Preliminary experiments showed that for our task of classification with 115 features and 3 classes and 5 classes, the proposed architecture should suffice.

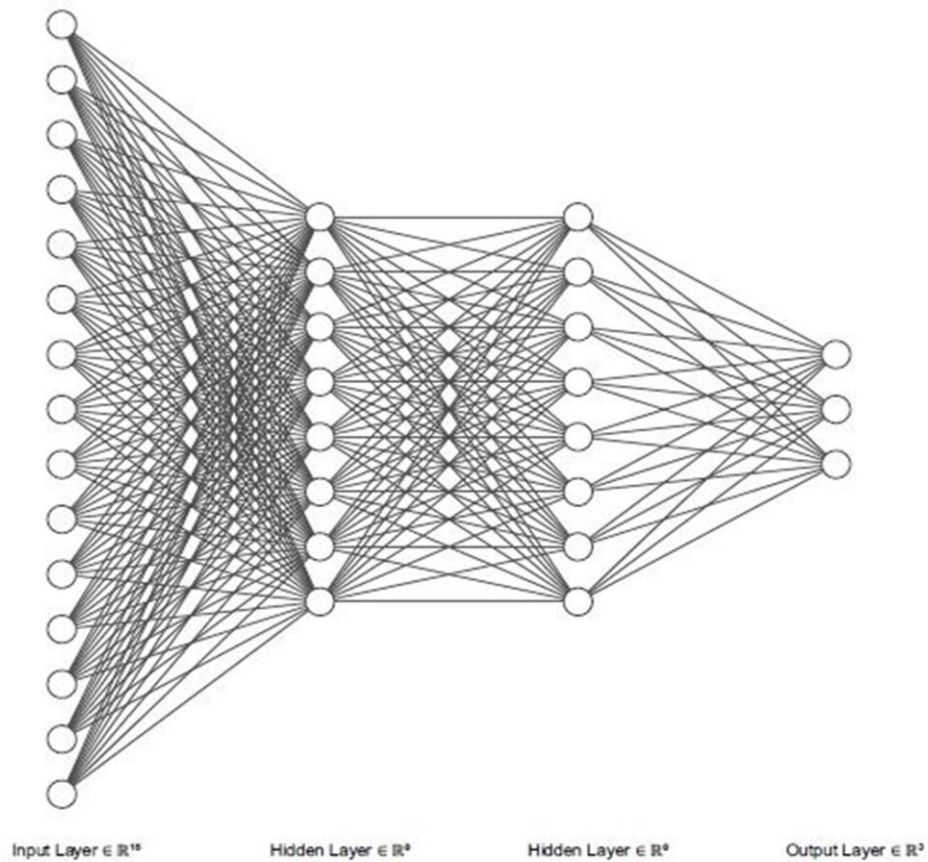


Fig 2.2: Deep Neural Network

Apart from the number of layers and neurons other important details to describe for proposed neural network are:

- Hyperbolic tangent activation function for hidden neurons
- Softmax function applied to the last layer
- Categorical cross-entropy as a loss function

Hyperbolic tangent activation function definition was already given previously.

The need for softmax arises from the fact that we are doing a classification, hence at the last layer we want to receive a probability vector with normalized probabilities for each class, and softmax gives us exactly that .

$$\text{softmax}(x_j) = \frac{e^{x_j}}{\sum_{i=1}^N e^{x_i}}$$

Categorical cross-entropy is a standard choice for a loss function with multiple classes. The code for model creation is given in Appendix 3.

2.4 Evaluation metrics

For the task of anomaly detection, we can apply a two-class confusion matrix to evaluate our results as in Table 1 .

	Predicted normal	Predicted attack
Actual normal	True Negative (TN)	False Positive (FP)
Actual attack	False Negative (FN)	True Positive (TP)

Table 2.1: Confusion Matrix Structure

Accuracy can then be defined as

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

And precision as (10).

$$PR = \frac{TP}{TP + FP}$$

False positive rate will be defined as

$$FPR = \frac{FP}{FP + TN}$$

Similarly, accuracy can be computed for more than 2 classes. Also, for our deep learning models, we can also measure how long does the training take in seconds to achieve good accuracy and how complex the model is, meaning how many parameters does it contain and how much does it weight when saved to disk.

3. Software Design

3.1 Control Flow Diagram

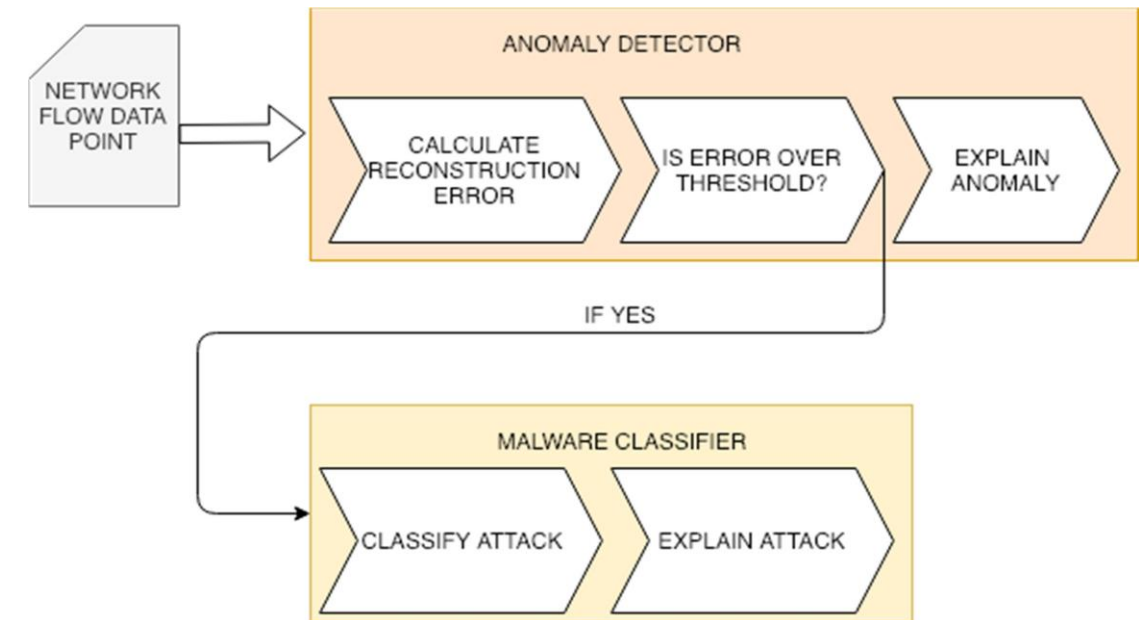


Fig.3.1: Potential attack detection, classification and explanation pipeline

4. Requirements

4.1 Software

- Python 3.6 or higher
- wget
- numpy
- pandas
- sklearn
- tensorflow
- keras
- tensorboard
- lime

4.2 Hardware

- Intel Core i7 CPU
- 16 GB DDR3 RAM

5. Coding 5.1 Anomaly

Detection

5.1.1 Train Model:

5.1.1.1 Libraries Required

```
import sys
import os
import pandas as pd
from glob import glob
import numpy as np
from keras.models import load_model
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from keras.models import Model, Sequential
from keras.layers import Input, Dense
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras.optimizers import SGD
```

Fig. 5.1: Libraries

5.1.1.2 Train function

- Parameters - (Top n features)
- Functionality
- The train function creates a data frame for all the normal traffic data and calculates the top n features using the fisher scoring algorithm.
- The data is then splitted randomly in 3 equal parts i.e. the train, optimized and test data.
- The model is then fitted and the threshold is calculated.

```

def train(top_n_features=10):
    print("Loading combined training data...")
    df = pd.concat((pd.read_csv(f) for f in iglob('../data/**/benign_traffic.csv', recursive=True)), ignore_index=True)

    fisher = pd.read_csv('../fisher.csv')
    features = fisher.iloc[0:int(top_n_features)][['Feature']].values
    df = df[list(features)]
    x_train, x_opt, x_test = np.split(df.sample(frac=1, random_state=17), [int(1/3*len(df)), int(2/3*len(df))])
    scaler = StandardScaler()
    scaler.fit(x_train.append(x_opt))
    x_train = scaler.transform(x_train)
    x_opt = scaler.transform(x_opt)
    x_test = scaler.transform(x_test)

    model = create_model(top_n_features)
    model.compile(loss="mean_squared_error",
                  optimizer="sgd")
    cp = ModelCheckpoint(filepath=f"models/model_{top_n_features}.h5",
                        save_best_only=True,
                        verbose=0)
    tb = TensorBoard(log_dir=f"./logs",
                    histogram_freq=0,
                    write_graph=True,
                    write_images=True)
    print(f"Training model for all data combined")
    model.fit(x_train, x_train,
            epochs=500,
            batch_size=64,
            validation_data=(x_opt, x_opt),
            verbose=1,
            callbacks=[cp, tb])

    print("Calculating threshold")
    x_opt_predictions = model.predict(x_opt)
    print("Calculating MSE on optimization set...")
    mse = np.mean(np.power(x_opt - x_opt_predictions, 2), axis=1)
    print("mean is %.5f" % mse.mean())
    print("min is %.5f" % mse.min())
    print("max is %.5f" % mse.max())
    print("std is %.5f" % mse.std())
    tr = mse.mean() + mse.std()
    with open(f'threshold_{top_n_features}', 'w') as t:
        t.write(str(tr))
    print(f"Calculated threshold is {tr}")

    x_test_predictions = model.predict(x_test)
    print("Calculating MSE on test set...")
    mse_test = np.mean(np.power(x_test - x_test_predictions, 2), axis=1)
    over_tr = mse_test > tr
    false_positives = sum(over_tr)
    test_size = mse_test.shape[0]
    print(f"{false_positives} false positives on dataset without attacks with size {test_size}")

```

Fig. 5.2: Train Function

5.1.1.3 Create_model Function

- Parameters : (Input data)
- Functionality
- It creates the autoencoder sequential model and returns the model to its calling function.

```
def create_model(input_dim):
    autoencoder = Sequential()
    autoencoder.add(Dense(int(0.75 * input_dim), activation="tanh", input_shape=(input_dim,)))
    autoencoder.add(Dense(int(0.5 * input_dim), activation="tanh"))
    autoencoder.add(Dense(int(0.33 * input_dim), activation="tanh"))
    autoencoder.add(Dense(int(0.25 * input_dim), activation="tanh"))
    autoencoder.add(Dense(int(0.33 * input_dim), activation="tanh"))
    autoencoder.add(Dense(int(0.5 * input_dim), activation="tanh"))
    autoencoder.add(Dense(int(0.75 * input_dim), activation="tanh"))
    autoencoder.add(Dense(input_dim))
    return autoencoder

if __name__ == '__main__':
    train(*sys.argv[1:])
```

Fig. 5.3: Create model function

5.1.2 Test

5.1.2.1 Libraries Required

```
import sys
import pandas as pd
import numpy as np
import random
from keras.models import load_model
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from glob import glob
from sklearn.metrics import recall_score, accuracy_score, precision_score, confusion_matrix
import lime
import lime.lime_tabular
```

Fig. 5.4: Libraries for test

5.1.2.2 load_mal_data()

- Parameters - none
- Creates a data frame for all the malicious packets together in a single data frame.

5.1.2.3 test

- Parameters - top_n_features (default value = 115)
- Functionality - calls test_with_data function

5.1.2.3 test_with_data

- Parameters - top_n_features and data frame for malicious packets.
- Functionality -
 - Creates a data frame for normal packets and test the saved trained model.
 - Creates the accuracy, recall, and Precision and later explains using lime.

```

def test_with_data(top_n_features, df_malicious):
    print("Testing")
    df = pd.concat((pd.read_csv(f) for f in iglob('../data/**/benign_traffic.csv', recursive=True)), ignore_index=True)
    fisher = pd.read_csv('../fisher.csv')
    features = fisher.iloc[0:int(top_n_features)][['Feature']].values
    df = df[list(features)]
    x_train, x_opt, x_test = np.split(df.sample(frac=1, random_state=17), [int(1/3*len(df)), int(2/3*len(df))])
    scaler = StandardScaler()
    scaler.fit(x_train.append(x_opt))

    print(f"Loading model")
    saved_model = load_model(f'models/model_{top_n_features}.h5')
    with open(f'threshold_{top_n_features}') as t:
        tr = np.float64(t.read())
    print(f"Calculated threshold is {tr}")
    df_benign = pd.DataFrame(x_test, columns=df.columns)
    df_benign['malicious'] = 0
    df_malicious = df_malicious.sample(n=df_benign.shape[0], random_state=17)[list(features)]
    df_malicious['malicious'] = 1
    df = df_benign.append(df_malicious)
    X_test = df.drop(columns=['malicious']).values
    X_test_scaled = scaler.transform(X_test)
    Y_test = df['malicious']
    Y_pred = model.predict(X_test_scaled)
    print('Accuracy')
    print(accuracy_score(Y_test, Y_pred))
    print('Recall')
    print(recall_score(Y_test, Y_pred))
    print('Precision')
    print(precision_score(Y_test, Y_pred))
    print(confusion_matrix(Y_test, Y_pred))

class AnomalyModel:
    def __init__(self, model, threshold, scaler):
        self.model = model
        self.threshold = threshold
        self.scaler = scaler

    def predict(self, x):
        x_pred = self.model.predict(x)
        mse = np.mean(np.power(x - x_pred, 2), axis=1)
        y_pred = mse > self.threshold
        return y_pred.astype(int)

    def scale_predict_classes(self, x):
        x = self.scaler.transform(x)
        y_pred = self.predict(x)
        classes_arr = []
        for e in y_pred:
            el = [0,0]
            el[e] = 1
            classes_arr.append(el)

        return np.array(classes_arr)

if __name__ == '__main__':
    test(*sys.argv[1:])

```

Fig. 5.5: Test_with_data

5.2 Classification

5.2.1 Train Model

5.2.1.1 Libraries Required

```
import sys
import os
from glob import iglob
import pandas as pd
import numpy as np
import tensorflow as tf
import pickle
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from keras.models import model_from_yaml
from keras.models import Model, Sequential
from keras.layers import Input, Dense, Activation
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras.optimizers import Adam
from sklearn.model_selection import train_test_split
```

Fig. 5.6: Libraries

5.2.1.2 load_data

- Parameters - none
- Functionality -
- Creates a labelled data frame for attacks packets.

```
def load_data():
    print('Loading data')
    print('Loading gafgyt data')
    df_gafgyt = pd.concat((pd.read_csv(f) for f in iglob('C:/Users/ritesh/Desktop/botnet-traffic-analysis-master/data/Danmini_Doorbell/gafgyt_attacks/*.csv', recursive=True)), ignore_index=True)
    print('Loaded, shape: ')
    print(df_gafgyt.shape)
    df_gafgyt['class'] = 'gafgyt'
    print('Loading mirai data')
    df_mirai = pd.concat((pd.read_csv(f) for f in iglob('C:/Users/ritesh/Desktop/botnet-traffic-analysis-master/data/Danmini_Doorbell/mirai_attacks/*.csv', recursive=True)), ignore_index=True)
    print('Loaded, shape: ')
    print(df_mirai.shape)
    df_mirai['class'] = 'mirai'
    print('Loading benign data')
    df_benign = pd.concat((pd.read_csv(f) for f in iglob('C:/Users/ritesh/Desktop/botnet-traffic-analysis-master/data/Danmini_Doorbell/benign_traffic.csv', recursive=True)), ignore_index=True)
    print('Loaded, shape: ')
    print(df_benign.shape)
    df_benign['class'] = 'benign'
    df = df_benign.append(df_gafgyt.sample(n=df_benign.shape[0], random_state=17)).append(df_mirai.sample(n=df_benign.shape[0], random_state=17))
    return df
```

Fig. 5.7: Load Data

5.2.1.3 create_model()

- Parameters - input_dim, add_hidden_layers, hidden_layer_size
- Functionality
 - Creates a model by adding the specified hidden layers of specified layer size.

```
def create_model(input_dim, add_hidden_layers, hidden_layer_size):  
    model = Sequential()  
    model.add(Dense(hidden_layer_size, activation="tanh", input_shape=(input_dim,)))  
    for i in range(add_hidden_layers):  
        model.add(Dense(hidden_layer_size, activation="tanh"))  
    model.add(Dense(3))  
    model.add(Activation('softmax'))  
    return model
```

Fig. 5.8: Create_model Function

5.2.1.4 train_with_data()

- Parameters - top n features and data_frame
- Functionality
 - Calls the create_model function to create the function.
 - Defines the epochs for training.
 - Fits the model.

```

def train(top_n_features = None):
    df = load_data()
    train_with_data(top_n_features, df)
def train_with_data(top_n_features = None, df = None):
    X = df.drop(columns=['class'])
    if top_n_features is not None:
        fisher = pd.read_csv('C:/Users/ritesh/Desktop/botnet-traffic-analysis-master/data/fisherscore.csv')
        features = fisher.iloc[0:int(top_n_features)][['Feature']].values
        X = X[list(features)]
    Y = pd.get_dummies(df['class'])
    print('Splitting data')
    x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
    scaler = StandardScaler()
    print('Transforming data')
    scaler.fit(x_train)
    input_dim = X.shape[1]
    scalerfile = f'./models/scaler_{input_dim}.sav'
    pickle.dump(scaler, open(scalerfile, 'wb'))
    x_train = scaler.transform(x_train)
    x_test = scaler.transform(x_test)
    print('Creating a model')

    model = create_model(input_dim, 1, 128)
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    cp = ModelCheckpoint(filepath=f'./models/model_{input_dim}.h5',
                        save_best_only=True,
                        verbose=0)
    tb = TensorBoard(log_dir=f'./logs',
                    histogram_freq=0,
                    write_graph=True,
                    write_images=True)
    epochs = 25
    model.fit(x_train, y_train,
            epochs=epochs,
            batch_size=256,
            validation_data=(x_test, y_test),
            verbose=1,
            callbacks=[tb, cp])
    print('Model evaluation')
    print('Loss, Accuracy')
    print(model.evaluate(x_test, y_test))
    y_pred_proba = model.predict(x_test)
    y_pred = np.argmax(y_pred_proba, axis=1)
    cnf_matrix = confusion_matrix(np.argmax(y_test.values, axis=1), y_pred)
    print('Confusion matrix')
    print('benign  gafgyt  mirai')
    print(cnf_matrix)

if __name__ == '__main__':
    train(*sys.argv[1:])

```

Fig. 5.9: train function

5.2.2 Test

5.2.2.1 Libraries required

```
import train
import sys
from glob import iglob
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from keras.models import load_model
from sklearn.model_selection import train_test_split
import lime
import lime.lime_tabular
```

Fig. 5.10: Libraries

5.2.2.2 test_with_data

- Parameters - top_n_features, model_name
- Functionality -
 - Test the data after splitting with the saved trained model.
 - Calculates the loss and accuracy.
 - Creates a confusion matrix and print it.
 - Explains some random results which includes the printing of that data and the classification result of that data packet.


```

def test_with_data(num_features, model_name, df):
    X = df.drop(columns=['class'])
    if num_features is not None:
        fisher = pd.read_csv('../fisher.csv')
        features = fisher.iloc[0:int(num_features)][['Feature']].values
        X = X[list(features)]
    Y = pd.get_dummies(df['class'])
    classes = Y.columns.tolist()
    print('Splitting data')
    x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
    scaler = StandardScaler()
    print('Transforming data')
    scaler.fit(x_train)
    x_test_scaled = scaler.transform(x_test)

    model = load_model(f'models/{model_name}')
    wrapper = ModelWrapper(model, scaler)
    print('Model evaluation')
    print('Loss, accuracy')
    print(model.evaluate(x_test_scaled, y_test))
    y_pred_proba = model.predict(x_test_scaled)
    y_pred = np.argmax(y_pred_proba, axis=1)

    cnf_matrix = confusion_matrix(np.argmax(y_test.values, axis=1), y_pred)
    print('Confusion matrix')
    print(classes)

    print(cnf_matrix)

    print('Explaining data using lime')
    explainer = lime.lime_tabular.LimeTabularExplainer(x_train.values, feature_names=X.columns.tolist(), class_names=classes, discretize_continuous=True)
    for j in range(10):
        i = np.random.randint(0, x_test.shape[0])
        print(f'Explaining for record nr {i}')
        exp = explainer.explain_instance(x_test.values[i], wrapper.scale_predict, num_features=int(num_features), top_labels=5)
        exp.save_to_file(f'lime/explanation{j}.html')
        print(y_test.values[i])
        print(exp.as_list())

def test(num_features, model_name):
    df = train.load_data()
    test_with_data(num_features, model_name, df)

class ModelWrapper:
    def __init__(self, model, scaler):
        self.model = model
        self.scaler = scaler

    def scale_predict(self, x):
        x = self.scaler.transform(x)
        return self.model.predict(x)

if __name__ == '__main__':
    test(*sys.argv[1:])

```

Fig. 5.11: test

5.3 Result

5.3.1 Feature scoring

Fisher's score was calculated on training sets for each data subset used respectively for attack detection, botnet type classification and Mirai attack type classification. Results in later sections where only a subset of features is used, are based on these rankings. Code for the score calculation procedure can be found in Appendix 1.

5.3.2 Accuracy

For top 5 feature, trained for 25 epochs

Loss	Accuracy
0.00360615776788811	0.99946033462036

Table 5.1: Result for top 5 features

5.3.3 Confusion Matrix

Benign	gafgyt	mirai
110824	38	0
95	111063	0
24	23	111493

Table 5.2: Confusion matrix

6. Testing

6.1 BLACK BOX TESTING

Black box testing is defined as a testing technique in which functionality of the Application Under Test (AUT) is tested without looking at the internal code structure, implementation details and knowledge of internal paths of the software. This type of testing is based entirely on software requirements

and specifications. In BlackBox Testing we just focus on inputs and output of the software system without bothering about internal knowledge of the software program.



Fig. 6.1: Black Box Testing

The above Black-Box can be any software system you want to test. For Example, an operating system like Windows, a website like Google, a database like Oracle or even your own custom application. Under Black Box Testing, you can test these applications by just focusing on the inputs and outputs without knowing their internal code implementation.

- Random section of record from inputted data as input to the anomaly detector and classifier.
- Output Matrix- ['benign', 'gafgyt', 'mirai']
- The first record is Detected as a Normal packet.
- The second record is detected as attacked and classified as mirai.

```

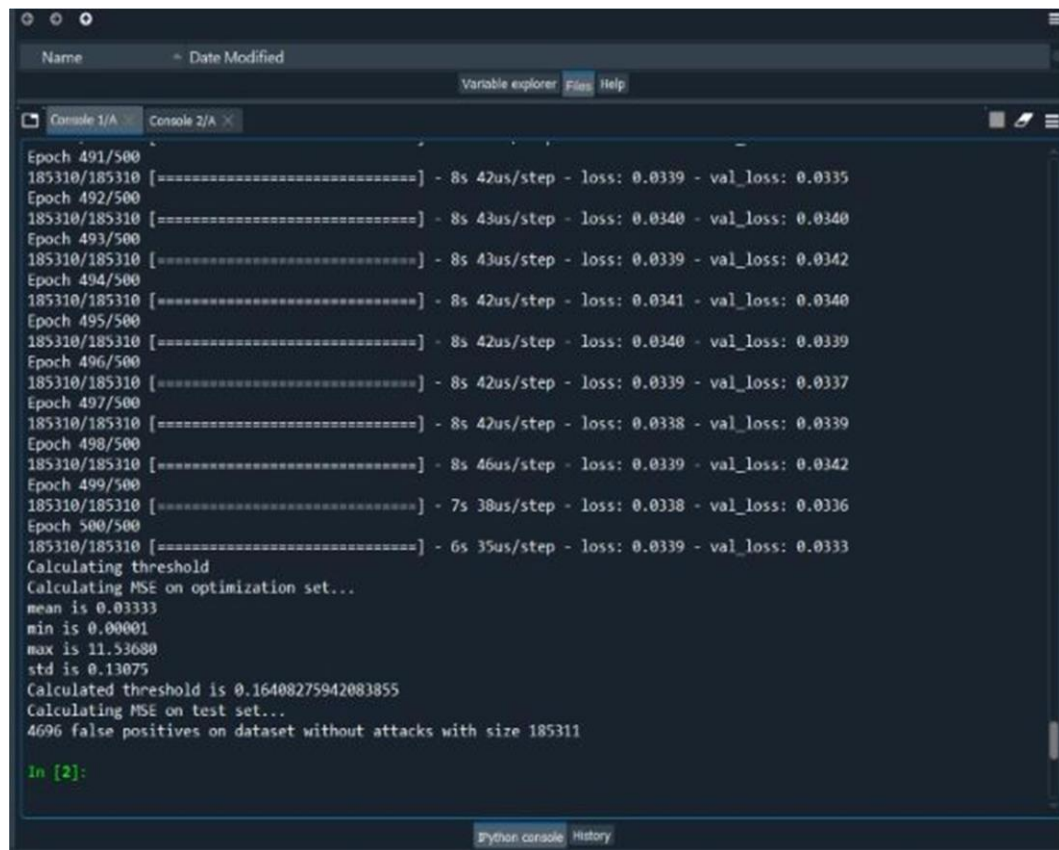
Explaining for record nr 183061
[1 0 0]
[('74.56 < MI_dir_l0.01_mean <= 240.56', 0.16903030472006833), ('74.56 < H_l0.01_mean <= 240.55',
0.14053674532223923), ('74.57 < MI_dir_l0.1_mean <= 280.60', 0.11499177090614666), ('105.87 < H_l0.01_weight <=
24113.25', 0.012527247594878254), ('105.87 < MI_dir_l0.01_weight <= 24113.25', -0.0063705718306930345)]
Explaining for record nr 242156
[0 0 1]
[('H_l0.01_weight > 24113.25', 0.38400434945527934), ('MI_dir_l0.01_weight > 24113.25', 0.36659219069848975),
('H_l0.01_mean > 240.55', -0.24527293148629978), ('MI_dir_l0.01_mean > 240.56', -0.22584127346799565),
('MI_dir_l0.1_mean > 280.60', -0.0905168106571494)]
  
```

Fig. 6.2: Black Box Test

7. Output Screens

7.1 Anomaly Detection

7.1.1 Models output console with 15 features



```
Name      Date Modified
Variable explorer Files Help

Console 1/A Console 2/A

Epoch 491/500
185310/185310 [=====] - 8s 42us/step - loss: 0.0339 - val_loss: 0.0335
Epoch 492/500
185310/185310 [=====] - 8s 43us/step - loss: 0.0340 - val_loss: 0.0340
Epoch 493/500
185310/185310 [=====] - 8s 43us/step - loss: 0.0339 - val_loss: 0.0342
Epoch 494/500
185310/185310 [=====] - 8s 42us/step - loss: 0.0341 - val_loss: 0.0340
Epoch 495/500
185310/185310 [=====] - 8s 42us/step - loss: 0.0340 - val_loss: 0.0339
Epoch 496/500
185310/185310 [=====] - 8s 42us/step - loss: 0.0339 - val_loss: 0.0337
Epoch 497/500
185310/185310 [=====] - 8s 42us/step - loss: 0.0338 - val_loss: 0.0339
Epoch 498/500
185310/185310 [=====] - 8s 46us/step - loss: 0.0339 - val_loss: 0.0342
Epoch 499/500
185310/185310 [=====] - 7s 38us/step - loss: 0.0338 - val_loss: 0.0336
Epoch 500/500
185310/185310 [=====] - 6s 35us/step - loss: 0.0339 - val_loss: 0.0333
Calculating threshold
Calculating MSE on optimization set...
mean is 0.03333
min is 0.00001
max is 11.53680
std is 0.13075
Calculated threshold is 0.16408275942083855
Calculating MSE on test set...
4696 false positives on dataset without attacks with size 185311

In [2]:
```

Fig. 7.1: train output screen with top 15 features

```

In [2]: runfile('C:/Users/Sanjay/Documents/VISUAL STUDIO/botnet-traffic-analysis-master/anomaly-detection/test.py',
wdir='C:/Users/Sanjay/Documents/VISUAL STUDIO/botnet-traffic-analysis-master/anomaly-detection')
Testing
Loading model
Calculated threshold is 0.16408275942083855
Accuracy
0.8567947936172164
Recall
0.7389307704345668
Precision
0.9668427147174288
[[180615  4696]
 [ 48379 136932]]
explaining with LIME
Explaining for record nr 279576
[('MI_dir_L0.01_weight > 100.18', 0.04077913617199219), ('H_L0.01_weight > 100.18', 0.03649428348971734), ('66.04 <
H_L1_mean <= 82.65', -0.022658888506444763), ('MI_dir_L0.1_mean <= 72.31', -0.01778015865682279), ('66.04 <
MI_dir_L1_mean <= 82.65', -0.01629493885745757), ('H_L0.1_mean <= 72.31', -0.014770972491050229), ('MI_dir_L0.01_mean
<= 73.59', -0.011777770698442186), ('H_L0.01_variance <= 354.13', -0.00884983332466872), ('H_L0.01_mean <= 73.59',
-0.008266607657583787), ('MI_dir_L0.01_variance <= 354.08', -0.0017564513448467543)]
Actual class
1974625  1
Name: malicious, dtype: int64
Explaining for record nr 84594
[('H_L0.01_variance > 23367.24', 0.04407384397918242), ('MI_dir_L0.01_variance > 23367.24', 0.04102209250882233),
('72.31 < MI_dir_L0.1_mean <= 86.55', -0.0214142685900306), ('66.04 < MI_dir_L1_mean <= 82.65',
-0.018218521895836604), ('72.31 < H_L0.1_mean <= 86.55', -0.01531230202916082), ('38.31 < H_L0.01_weight <= 100.18',
-0.012671193869003637), ('91.75 < MI_dir_L0.01_mean <= 149.58', -0.011509253807102236), ('91.75 < H_L0.01_mean <=
149.58', -0.010800854809423116), ('66.04 < H_L1_mean <= 82.65', -0.0087555829604082), ('38.31 < MI_dir_L0.01_weight
<= 100.18', -0.0088857298523056996)]
Actual class
299176  0

```

Fig. 7.2: test output screen with top 15 features 1

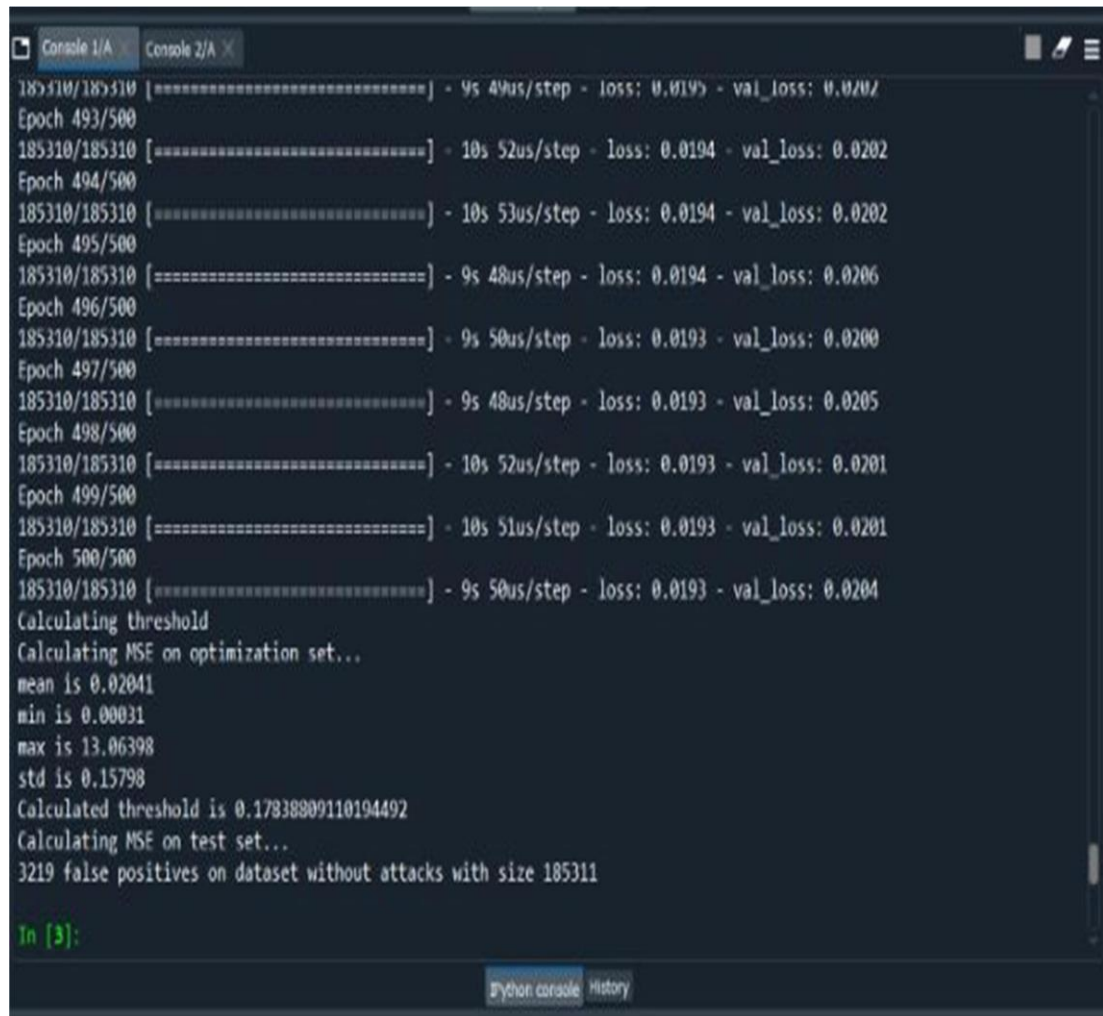
```
explaining with LIME
Explaining for record nr 279576
[('MI_dir_L0.01_weight > 100.18', 0.04077913617199219), ('H_L0.01_weight > 100.18', 0.03649428348971734), ('66.04 <
H_L1_mean <= 82.65', -0.022658889506444763), ('MI_dir_L0.1_mean <= 72.31', -0.01778015865682279), ('66.04 <
MI_dir_L1_mean <= 82.65', -0.01629493885745757), ('H_L0.1_mean <= 72.31', -0.014770972491050229), ('MI_dir_L0.01_mean
<= 73.59', -0.011777770698442186), ('H_L0.01_variance <= 354.13', -0.00884983332466872), ('H_L0.01_mean <= 73.59',
-0.008266607657583787), ('MI_dir_L0.01_variance <= 354.08', -0.0017564513448467543)]
Actual class
1974625 1
Name: malicious, dtype: int64
Explaining for record nr 84594
[('H_L0.01_variance > 23367.24', 0.04407384397918242), ('MI_dir_L0.01_variance > 23367.24', 0.04102209250882233),
('72.31 < MI_dir_L0.1_mean <= 86.55', -0.0214142685900306), ('66.04 < MI_dir_L1_mean <= 82.65',
-0.018218521895836604), ('72.31 < H_L0.1_mean <= 86.55', -0.01531230202916082), ('38.31 < H_L0.01_weight <= 100.18',
-0.012671193869003637), ('91.75 < MI_dir_L0.01_mean <= 149.58', -0.011509253807102236), ('91.75 < H_L0.01_mean <=
149.58', -0.010800854809423116), ('66.04 < H_L1_mean <= 82.65', -0.0087555829604082), ('38.31 < MI_dir_L0.01_weight
<= 100.18', -0.0008857298523056996)]
Actual class
299176 0
Name: malicious, dtype: int64
Explaining for record nr 208735
[('H_L0.01_weight <= 28.27', -0.024756330484134153), ('H_L0.1_mean <= 72.31', -0.02011723360433408), ('H_L1_mean <=
66.04', -0.017736326815645978), ('MI_dir_L0.01_variance <= 354.08', -0.016014704848145035), ('MI_dir_L0.01_mean <=
73.59', -0.014024977494643973), ('MI_dir_L0.01_weight <= 28.27', -0.012142602907848849), ('H_L0.01_variance <=
354.13', -0.010459434973824401), ('MI_dir_L1_mean <= 66.04', -0.008625810949835493), ('H_L0.01_mean <= 73.59',
-0.0031830159480030202), ('MI_dir_L0.1_mean <= 72.31', -0.000550883310009241)]
Actual class
1428513 1
Name: malicious, dtype: int64
Explaining for record nr 257252
[('MI_dir_L0.01_weight > 100.18', 0.044393672934943375), ('H_L0.01_weight > 100.18', 0.041381019079519465), ('66.04 <
H_L1_mean <= 82.65', -0.02592026634703219), ('MI_dir_L0.01_variance <= 354.08', -0.02424305305653438), ('H_L0.01_mean
<= 73.59', -0.02286900739701992), ('MI_dir_L0.1_mean <= 72.31', -0.013588732659144502), ('H_L0.01_variance <=
```

Fig. 7.3: Test output screen with top 15 features 2

```
Console 1/A Console 2/A
[('H_L0.01_weight <= 28.27', -0.024756330484134153), ('H_L0.1_mean <= 72.31', -0.02011723360433408), ('H_L1_mean <= 66.04', -0.017736326815645078), ('MI_dir_L0.01_variance <= 354.08', -0.016014704848145035), ('MI_dir_L0.01_mean <= 73.59', -0.014024977494643973), ('MI_dir_L0.01_weight <= 28.27', -0.012142602907848849), ('H_L0.01_variance <= 354.13', -0.010459434973824401), ('MI_dir_L1_mean <= 66.04', -0.008625810949835493), ('H_L0.01_mean <= 73.59', -0.0031830159480030202), ('MI_dir_L0.1_mean <= 72.31', -0.000550883310009241)]
Actual class
1428513 1
Name: malicious, dtype: int64
Explaining for record nr 257252
[('MI_dir_L0.01_weight > 100.18', 0.044393672934943375), ('H_L0.01_weight > 100.18', 0.041381019079519465), ('66.04 < H_L1_mean <= 82.65', -0.02592026634703219), ('MI_dir_L0.01_variance <= 354.08', -0.02424305305653438), ('H_L0.01_mean <= 73.59', -0.02286900739701992), ('MI_dir_L0.1_mean <= 72.31', -0.013588732659144502), ('H_L0.01_variance <= 354.13', -0.013262023081592918), ('66.04 < MI_dir_L1_mean <= 82.65', -0.012417147687334564), ('MI_dir_L0.01_mean <= 73.59', -0.010100898803909044), ('H_L0.1_mean <= 72.31', -0.007203953056646888)]
Actual class
2385184 1
Name: malicious, dtype: int64
Explaining for record nr 324746
[('MI_dir_L0.1_mean <= 72.31', -0.03620507534999927), ('H_L0.1_mean <= 72.31', -0.0352874182100805), ('H_L0.01_variance <= 354.13', -0.022856175259161643), ('H_L0.01_mean <= 73.59', -0.022046088581055435), ('MI_dir_L0.01_weight <= 28.27', -0.021383415867324642), ('H_L0.01_weight <= 28.27', -0.017995203373015693), ('H_L1_mean <= 66.04', -0.01706834193650359), ('MI_dir_L0.01_variance <= 354.08', -0.016781169593639244), ('MI_dir_L1_mean <= 66.04', -0.015026565279186985), ('MI_dir_L0.01_mean <= 73.59', -0.002023475221841689)]
Actual class
2453901 1
Name: malicious, dtype: int64
-----
In [3]: |
```

Fig. 7.4: Test output screen with top 15 features 3

7.1.2 Models output console with 115 features



```
185310/185310 [=====] - 9s 49us/step - loss: 0.0193 - val_loss: 0.0202
Epoch 493/500
185310/185310 [=====] - 10s 52us/step - loss: 0.0194 - val_loss: 0.0202
Epoch 494/500
185310/185310 [=====] - 10s 53us/step - loss: 0.0194 - val_loss: 0.0202
Epoch 495/500
185310/185310 [=====] - 9s 48us/step - loss: 0.0194 - val_loss: 0.0206
Epoch 496/500
185310/185310 [=====] - 9s 50us/step - loss: 0.0193 - val_loss: 0.0200
Epoch 497/500
185310/185310 [=====] - 9s 48us/step - loss: 0.0193 - val_loss: 0.0205
Epoch 498/500
185310/185310 [=====] - 10s 52us/step - loss: 0.0193 - val_loss: 0.0201
Epoch 499/500
185310/185310 [=====] - 10s 51us/step - loss: 0.0193 - val_loss: 0.0201
Epoch 500/500
185310/185310 [=====] - 9s 50us/step - loss: 0.0193 - val_loss: 0.0204
Calculating threshold
Calculating MSE on optimization set...
mean is 0.02041
min is 0.00031
max is 13.06398
std is 0.15798
Calculated threshold is 0.17838809110194492
Calculating MSE on test set...
3219 false positives on dataset without attacks with size 185311

In [3]:
```

Fig. 7.5: train output screen with top 15 features

```

In [4]: runfile('C:/Users/Sanjay/Documents/VISUAL STUDIO/botnet-traffic-analysis-master/anomaly-detection/test.py',
wdir='C:/Users/Sanjay/Documents/VISUAL STUDIO/botnet-traffic-analysis-master/anomaly-detection')
Testing
Loading model
Calculated threshold is 0.17838809110194492
Accuracy
0.8610309155959441
Recall
0.7392707394596112
Precision
0.977251326827598
[[182122  3189]
 [ 48316 136995]]
explaining with LIME
Explaining for record nr 137784
[('HH_L5_weight > 1.91', -0.001654111133133366), ('HpHp_L5_mean > 102.00', -0.0016496455495152204), ('2.98 <
HH_jit_L0.1_variance <= 12.79', -0.0015961447333026132), ('HH_L0.01_weight > 74.96', -0.00157726041287236),
('H_L1_weight > 3.07', -0.0015494649241681996), ('HpHp_L0.01_magnitude > 144.25', -0.0015351942332751316),
('HH_L1_std > 8.93', -0.001468170731700056), ('HH_L0.1_mean > 110.00', -0.0014608084810853869), ('0.00 <
HH_jit_L5_variance <= 1.96', -0.0014557430717413245), ('HH_jit_L3_weight > 1.99', -0.0014512111098801891)]
Actual class
328298  0
Name: malicious, dtype: int64
Explaining for record nr 144662
[('0.00 < HH_L1_std <= 0.02', -0.0017439004875220415), ('HH_L0.01_mean > 128.85', -0.0016645751208718604),
('HH_L5_magnitude > 144.25', -0.0015690131686439007), ('12.79 < HH_jit_L0.1_variance <= 106.72',
Python console History

```

Fig. 7.6: test output screen with 115 features 1

```

HH_jit_L5_variance <= 1.96', -0.0014557430717413245), ('HH_jit_L3_weight > 1.99', -0.0014512111098801891)]
Actual class
328298  0
Name: malicious, dtype: int64
Explaining for record nr 144662
[('0.00 < HH_L1_std <= 0.02', -0.0017439004875220415), ('HH_L0.01_mean > 128.85', -0.0016645751208718604),
('HpHp_L5_magnitude > 144.25', -0.0015690131686439007), ('12.79 < HH_jit_L0.1_variance <= 106.72',
-0.0015600585147251197), ('HpHp_L0.1_radius > 250.99', -0.0015374752876848002), ('0.00 < HH_L1_radius <= 5.73',
-0.001536662463703661), ('HH_L3_mean > 104.43', -0.001522892618734432), ('HpHp_L3_magnitude > 144.25',
-0.0015118058678116305), ('HH_L1_covariance > 0.00', -0.0014746806829575548), ('H_L3_mean > 106.07',
-0.0014518706285577855)]
Actual class
409306  0
Name: malicious, dtype: int64
Explaining for record nr 209600
[('H_L1_weight <= 1.11', -0.001514229119004902), ('HH_jit_L0.1_variance <= 2.98', -0.0015086292248725128),
('HpHp_L5_magnitude <= 84.85', -0.0015071968351711594), ('H_L0.01_mean <= 73.59', -0.0014932818718681612),
('HH_L1_mean <= 66.00', -0.001450162676270047), ('MI_dir_L1_variance <= 0.08', -0.0014376360804338976),
('HpHp_L0.01_mean <= 66.12', -0.0014346246246385499), ('H_L3_variance <= 0.00', -0.001422004959748578),
('HpHp_L0.1_std <= 0.00', -0.0014130647875969491), ('HH_L0.1_radius <= 0.00', -0.001409044426264009)]
Actual class
2360300  1
Name: malicious, dtype: int64
Explaining for record nr 110896
[('1.00 < HH_L3_weight <= 1.00', -0.0040089012341199635), ('1.00 < HH_jit_L3_weight <= 1.00', -0.003794241496090158),
('0.00 < HH_jit_L5_variance <= 0.00', -0.0036256482043248), ('0.00 < HH_L5_radius <= 0.00', -0.003422037283308494),
('0.00 < HpHp_L5_std <= 0.00', -0.003335089577167535), ('0.00 < HH_jit_L3_variance <= 0.00', -0.0030185442465308857),
Python console History

```

Fig. 7.7: test output screen with 115 features 2

```
2360300 1
Name: malicious, dtype: int64
Explaining for record nr 110896
[('1.00 < HH_L3_weight <= 1.00', -0.0040089012341199635), ('1.00 < HH_jit_L3_weight <= 1.00', -0.003794241496090158),
('0.00 < HH_jit_L5_variance <= 0.00', -0.0036256482043248), ('0.00 < HH_L5_radius <= 0.00', -0.003422037283308494),
('0.00 < HpHp_L5_std <= 0.00', -0.003335089577167535), ('0.00 < HH_jit_L3_variance <= 0.00', -0.0030185442465308857),
('0.00 < HpHp_L3_std <= 0.00', -0.0024297749250211517), ('0.00 < HH_L3_radius <= 0.00', -0.0022397373002041084),
('0.00 < HpHp_L0.1_std <= 0.00', -0.0018671517095471628), ('2.98 < HH_jit_L0.1_variance <= 12.79',
-0.0015500472725177294)]
Actual class
217889 0
Name: malicious, dtype: int64
Explaining for record nr 172932
[('0.00 < HpHp_L1_covariance <= 0.00', -0.0041425303532310375), ('0.00 < HpHp_L0.1_covariance <= 0.00',
-0.0040608236043739926), ('122.33 < HpHp_L3_magnitude <= 144.25', -0.003279182199810908), ('0.00 < HH_L1_covariance
<= 0.00', -0.002770035412446734), ('122.33 < HpHp_L5_magnitude <= 144.25', -0.0026972753533336617), ('0.00 <
HpHp_L0.01_std <= 1.96', -0.0018367246852047255), ('0.00 < HpHp_L0.01_radius <= 74.29', -0.001645066199189938),
('H_L1_variance <= 0.08', -0.0015292388252748383), ('HH_jit_L0.01_mean > 37.85', -0.0014853048645543637),
('HI_dir_L3_weight > 2.97', -0.001455549340021263)]
Actual class
43382 0
Name: malicious, dtype: int64
.....
In [5]: |
```

Fig. 7.8: test output screen with 115 features 3

7.2 Classification

7.2.1 Models output console with top 5 features

```
array(['MI_dir_L0.01_weight', 'H_L0.01_weight', 'MI_dir_L0.01_mean',  
      'H_L0.01_mean', 'MI_dir_L0.1_mean', 'H_L0.1_mean',  
      'MI_dir_L0.01_variance', 'H_L0.01_variance', 'MI_dir_L1_mean',  
      'H_L1_mean'], dtype=object)  
  
In [12]: runfile('C:/Users/Sanjay/Documents/VISUAL STUDIO/botnet-traffic-analysis-master/classification/train.py',  
              wdir='C:/Users/Sanjay/Documents/VISUAL STUDIO/botnet-traffic-analysis-master/classification')  
Loading data  
Loading gafgyt data  
Loaded, shape:  
(2521622, 115)  
Loading mirai data  
Loaded, shape:  
(3668402, 115)  
Loading benign data  
Loaded, shape:  
(555932, 115)  
Splitting data  
Transforming data  
Creating a model  
Train on 1334236 samples, validate on 333560 samples  
Epoch 1/25  
16128/1334236 [.....] - ETA: 15:26 - loss: 0.5170 - accuracy: 0.8482C:\Users\Sanjay  
Anaconda3\envs\tensorflow_gpu\lib\site-packages\keras\callbacks\callbacks.py:95: RuntimeWarning: Method  
(on_train_batch_end) is slow compared to the batch update (1.763951). Check your callbacks.  
% (hook_name, delta_t_median), RuntimeWarning)  
C:\Users\Sanjay\Anaconda3\envs\tensorflow_gpu\lib\site-packages\keras\callbacks\callbacks.py:95: RuntimeWarning:  
Method (on_train_batch_end) is slow compared to the batch update (0.191624). Check your callbacks.  
% (hook_name, delta_t_median), RuntimeWarning)  
C:\Users\Sanjay\Anaconda3\envs\tensorflow_gpu\lib\site-packages\keras\callbacks\callbacks.py:95: RuntimeWarning:  
Method (on_train_batch_end) is slow compared to the batch update (0.119297). Check your callbacks.  
% (hook_name, delta_t_median), RuntimeWarning)  
1334236/1334236 [=====] - 26s 20us/step - loss: 0.0546 - accuracy: 0.9828 - val_loss: 0.0065  
  
Python console History
```

Fig.7.9: Train output screen with top 5 feature 1

```
Console 2/A
1334236/1334236 [=====] - 13s 10us/step - loss: 0.0047 - accuracy: 0.9993 - val_loss: 0.0044
- val_accuracy: 0.9994
Epoch 10/25
1334236/1334236 [=====] - 13s 10us/step - loss: 0.0052 - accuracy: 0.9992 - val_loss: 0.0043
- val_accuracy: 0.9994
Epoch 11/25
1334236/1334236 [=====] - 12s 9us/step - loss: 0.0047 - accuracy: 0.9993 - val_loss: 0.0041
- val_accuracy: 0.9994
Epoch 12/25
1334236/1334236 [=====] - 13s 10us/step - loss: 0.0044 - accuracy: 0.9993 - val_loss: 0.0044
- val_accuracy: 0.9993
Epoch 13/25
1334236/1334236 [=====] - 12s 9us/step - loss: 0.0054 - accuracy: 0.9992 - val_loss: 0.0050
- val_accuracy: 0.9990
Epoch 14/25
1334236/1334236 [=====] - 12s 9us/step - loss: 0.0043 - accuracy: 0.9994 - val_loss: 0.0041
- val_accuracy: 0.9994
Epoch 15/25
1334236/1334236 [=====] - 12s 9us/step - loss: 0.0045 - accuracy: 0.9993 - val_loss: 0.0041
- val_accuracy: 0.9994
Epoch 16/25
1334236/1334236 [=====] - 12s 9us/step - loss: 0.0044 - accuracy: 0.9994 - val_loss: 0.0041
- val_accuracy: 0.9994
Epoch 17/25
1334236/1334236 [=====] - 12s 9us/step - loss: 0.0050 - accuracy: 0.9993 - val_loss: 0.0052
- val_accuracy: 0.9989
Epoch 18/25
1334236/1334236 [=====] - 13s 9us/step - loss: 0.0045 - accuracy: 0.9993 - val_loss: 0.0087
- val_accuracy: 0.9991
Epoch 19/25
1334236/1334236 [=====] - 12s 9us/step - loss: 0.0042 - accuracy: 0.9994 - val_loss: 0.0039
- val_accuracy: 0.9995
Epoch 20/25
```

Fig. 7.10: Train output screen with top 5 feature 2

```
Console 2/A
1334236/1334236 [=====] - 12s 9us/step - loss: 0.0042 - accuracy: 0.9994 - val_loss: 0.0039
- val_accuracy: 0.9995
Epoch 20/25
1334236/1334236 [=====] - 13s 10us/step - loss: 0.0045 - accuracy: 0.9993 - val_loss: 0.0040
- val_accuracy: 0.9994
Epoch 21/25
1334236/1334236 [=====] - 12s 9us/step - loss: 0.0044 - accuracy: 0.9994 - val_loss: 0.0048
- val_accuracy: 0.9993
Epoch 22/25
1334236/1334236 [=====] - 12s 9us/step - loss: 0.0047 - accuracy: 0.9993 - val_loss: 0.0044
- val_accuracy: 0.9993
Epoch 23/25
1334236/1334236 [=====] - 12s 9us/step - loss: 0.0041 - accuracy: 0.9994 - val_loss: 0.0039
- val_accuracy: 0.9994
Epoch 24/25
1334236/1334236 [=====] - 12s 9us/step - loss: 0.0042 - accuracy: 0.9994 - val_loss: 0.0040
- val_accuracy: 0.9995
Epoch 25/25
1334236/1334236 [=====] - 12s 9us/step - loss: 0.0040 - accuracy: 0.9994 - val_loss: 0.0036
- val_accuracy: 0.9995
Model evaluation
Loss, Accuracy
333560/333560 [=====] - 5s 16us/step
[0.003606157767588811, 0.9994603395462036]
Confusion matrix
benign gafgyt mirai
[[110824 38 0]
 [ 95 111063 0]
 [ 24 23 111493]]

In [13]: runfile('C:/Users/Sanjay/Documents/VISUAL STUDIO/botnet-traffic-analysis-master/classification/test.py',
wdir='C:/Users/Sanjay/Documents/VISUAL STUDIO/botnet-traffic-analysis-master/classification')
Loading data
```

Fig. 7.11: Train output screen with top 5 feature 3

```
Console 2/A
[ 24 23 111493]]

In [13]: runfile('C:/Users/Sanjay/Documents/VISUAL STUDIO/botnet-traffic-analysis-master/classification/test.py',
wdir='C:/Users/Sanjay/Documents/VISUAL STUDIO/botnet-traffic-analysis-master/classification')
Loading data
Loading gafgyt data
Loaded, shape:
(2521622, 115)
Loading mirai data
Loaded, shape:
(3668402, 115)
Loading benign data
Loaded, shape:
(555932, 115)
Splitting data
Transforming data
Model evaluation
Loss, accuracy
333560/333560 [=====] - 9s 26us/step
[0.003606157767588811, 0.9994603395462036]
Confusion matrix
['benign', 'gafgyt', 'mirai']
[[110824 38 0]
 [ 95 111063 0]
 [ 24 23 111493]]
Explaining data using lime
Explaining for record nr 258676
[1 0 0]
[('74.56 < H_L0.01_mean <= 240.55', 0.14932128380621415), ('74.56 < MI_dir_L0.01_mean <= 240.56',
0.14548326312086846), ('74.57 < MI_dir_L0.1_mean <= 280.60', 0.10866433615501647), ('105.87 < MI_dir_L0.01_weight <=
24113.25', -0.013368505716950908), ('105.87 < H_L0.01_weight <= 24113.25', -0.0012481933565945253)]
Explaining for record nr 36228
[0 1 0]
```

Fig. 7.12: Test output screen with top 5 feature 1


```
Console 2/A
['benign', 'gafgyt', 'mirai']
[[110824    38     0]
 [   95 111063     0]
 [   24    23 111493]]
Explaining data using lime
Explaining for record nr 258676
[1 0 0]
[('74.56 < H_L0.01_mean <= 240.55', 0.14932128380621415), ('74.56 < MI_dir_L0.01_mean <= 240.56',
0.14548326312086846), ('74.57 < MI_dir_L0.1_mean <= 280.60', 0.10866433615501647), ('105.87 < MI_dir_L0.01_weight <=
24113.25', -0.013368505716950908), ('105.87 < H_L0.01_weight <= 24113.25', -0.0012481933565945253)]
Explaining for record nr 36228
[0 1 0]
[('60.10 < H_L0.01_mean <= 74.56', 0.06501892849497397), ('60.10 < MI_dir_L0.01_mean <= 74.56',
0.029467891450889818), ('60.10 < MI_dir_L0.1_mean <= 74.57', -0.029242143176057674), ('105.87 < H_L0.01_weight <=
24113.25', 0.008015505586977714), ('105.87 < MI_dir_L0.01_weight <= 24113.25', -0.0033966048644436826)]
Explaining for record nr 123039
[0 1 0]
[('74.57 < MI_dir_L0.1_mean <= 280.60', 0.1644069554266108), ('60.10 < MI_dir_L0.01_mean <= 74.56',
0.05128894823909186), ('60.10 < H_L0.01_mean <= 74.56', 0.04639583343497377), ('105.87 < MI_dir_L0.01_weight <=
24113.25', 0.04427028353179028), ('105.87 < H_L0.01_weight <= 24113.25', 0.013122148091714633)]
Explaining for record nr 144456
[0 0 1]
[('H_L0.01_weight > 24113.25', 0.36324049553749965), ('MI_dir_L0.01_weight > 24113.25', 0.3525180528044182),
('H_L0.01_mean > 240.55', -0.24495635915029937), ('MI_dir_L0.01_mean > 240.56', -0.22940060193642503),
('MI_dir_L0.1_mean > 280.60', -0.09680346248554031)]
Explaining for record nr 65721
[0 0 1]
[('H_L0.01_weight > 24113.25', 0.3991794094749245), ('MI_dir_L0.01_weight > 24113.25', 0.34246376901962816), ('60.10
< H_L0.01_mean <= 74.56', 0.049013225061980335), ('60.10 < MI_dir_L0.01_mean <= 74.56', 0.030552920406873987),
('60.10 < MI_dir_L0.1_mean <= 74.57', -0.0292506672335964)]
Explaining for record nr 325902
[0 1 0]
[('60.10 < MI_dir_L0.01_mean <= 74.56', 0.04246657136201448), ('60.10 < H_L0.01_mean <= 74.56', 0.03539290840831713),
```

Fig. 7.13: Test output screen with top 5 feature 2

```
Console 2/A
[0 0 1]
[('H_L0.01_weight > 24113.25', 0.36324049553749965), ('MI_dir_L0.01_weight > 24113.25', 0.3525180528044182),
('H_L0.01_mean > 240.55', -0.24495635915029937), ('MI_dir_L0.01_mean > 240.56', -0.22940060193642503),
('MI_dir_L0.1_mean > 280.60', -0.09680346248554031)]
Explaining for record nr 65721
[0 0 1]
[('H_L0.01_weight > 24113.25', 0.3991794094749245), ('MI_dir_L0.01_weight > 24113.25', 0.34246376901962816), ('60.10
< H_L0.01_mean <= 74.56', 0.049013225861980335), ('60.10 < MI_dir_L0.01_mean <= 74.56', 0.030552920406873987),
('60.10 < MI_dir_L0.1_mean <= 74.57', -0.0292506672335964)]
Explaining for record nr 325902
[0 1 0]
[('60.10 < MI_dir_L0.01_mean <= 74.56', 0.04246657136201448), ('60.10 < H_L0.01_mean <= 74.56', 0.03539290840831713),
('60.10 < MI_dir_L0.1_mean <= 74.57', -0.010403948518549814), ('105.87 < MI_dir_L0.01_weight <= 24113.25',
0.009119963657625916), ('105.87 < H_L0.01_weight <= 24113.25', 0.0028528589579745445)]
Explaining for record nr 183061
[1 0 0]
[('74.56 < MI_dir_L0.01_mean <= 240.56', 0.16903030472006833), ('74.56 < H_L0.01_mean <= 240.55',
0.14053674532223923), ('74.57 < MI_dir_L0.1_mean <= 280.60', 0.11499177090614666), ('105.87 < H_L0.01_weight <=
24113.25', 0.012527247594878254), ('105.87 < MI_dir_L0.01_weight <= 24113.25', -0.0063705718306930345)]
Explaining for record nr 242156
[0 0 1]
[('H_L0.01_weight > 24113.25', 0.38400434945527934), ('MI_dir_L0.01_weight > 24113.25', 0.36659219069848975),
('H_L0.01_mean > 240.55', -0.24527293148629978), ('MI_dir_L0.01_mean > 240.56', -0.22584127346799565),
('MI_dir_L0.1_mean > 280.60', -0.0905168106571494)]
Explaining for record nr 115694
[1 0 0]
[('22.12 < H_L0.01_weight <= 105.87', -0.18262743395432587), ('22.12 < MI_dir_L0.01_weight <= 105.87',
-0.17449340789737514), ('60.10 < MI_dir_L0.01_mean <= 74.56', 0.036293229624582285), ('60.10 < H_L0.01_mean <=
74.56', 0.023739737551869255), ('60.10 < MI_dir_L0.1_mean <= 74.57', -0.021250207976742902)]
Explaining for record nr 209635
[1 0 0]
[('22.12 < H_L0.01_weight <= 105.87', -0.18045119965838566), ('22.12 < MI_dir_L0.01_weight <= 105.87',
-0.17230763698446028), ('74.56 < MI_dir_L0.01_mean <= 240.56', 0.15290102629520888), ('74.56 < H_L0.01_mean <=
```

Fig. 7.14: Test output screen with top 5 feature 3

```
Console 2/A
Explaining for record nr 183061
[1 0 0]
[('74.56 < MI_dir_L0.01_mean <= 240.56', 0.16903030472006833), ('74.56 < H_L0.01_mean <= 240.55',
0.14053674532223923), ('74.57 < MI_dir_L0.1_mean <= 280.60', 0.11499177090614666), ('105.87 < H_L0.01_weight <=
24113.25', 0.012527247594878254), ('105.87 < MI_dir_L0.01_weight <= 24113.25', -0.0063705718306930345)]
Explaining for record nr 242156
[0 0 1]
[('H_L0.01_weight > 24113.25', 0.38400434945527934), ('MI_dir_L0.01_weight > 24113.25', 0.36659219069848975),
('H_L0.01_mean > 240.55', -0.24527293148629978), ('MI_dir_L0.01_mean > 240.56', -0.22584127346799565),
('MI_dir_L0.1_mean > 280.60', -0.0905168106571494)]
Explaining for record nr 115694
[1 0 0]
[('22.12 < H_L0.01_weight <= 105.87', -0.18262743395432587), ('22.12 < MI_dir_L0.01_weight <= 105.87',
-0.17449340789737514), ('60.10 < MI_dir_L0.01_mean <= 74.56', 0.036293229624582285), ('60.10 < H_L0.01_mean <=
74.56', 0.023739737551869255), ('60.10 < MI_dir_L0.1_mean <= 74.57', -0.021250207976742902)]
Explaining for record nr 209635
[1 0 0]
[('22.12 < H_L0.01_weight <= 105.87', -0.18045119965838566), ('22.12 < MI_dir_L0.01_weight <= 105.87',
-0.17230763698446028), ('74.56 < MI_dir_L0.01_mean <= 240.56', 0.15290107629520888), ('74.56 < H_L0.01_mean <=
240.55', 0.14562955673033318), ('60.10 < MI_dir_L0.1_mean <= 74.57', 0.00697668344665853)]

In [14]:
```

Fig. 7.15: Test output screen with top 5 feature 4

8. Conclusions

Deep learning methods demonstrated good IoT botnet attack detection and classification accuracy. Moreover, these methods can work well with varying

number of features and generally their performance does not suffer from extra features, meaning that in real world environment they offer the possibility of using all existing data features.

It was also shown that there may not be a need to create different models for each device in the network, and that one model trained on data from all devices can be just as good.

The lack of interpretability, one of the neural network limitations, can be successfully addressed using LIME, as was demonstrated by producing feature value boundaries which upon closer inspection agreed with common sense.

9. Further Enhancement

- Creating a more robust and accurate model.
- Increasing accuracy of anomaly detector.
- Setting up a real time environment.
- Setting up GUI.
- Alert Generation.
- Data visualization.

9. References

- [1] Koroniotis, Nickolaos, et al. "Towards the Development of Realistic Botnet Dataset in the Internet of Things for Network Forensic Analytics: Bot-IoT Dataset." *Future Generation Computer Systems*, vol. 100, 2019, pp. 779–796.

[2] Meidan, Yair, et al. “N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders.” IEEE Pervasive Computing, vol. 17, no. 3, 2018, pp. 12–22.

[3] Dataset, archive.ics.uci.edu/ml/machine-learning-databases/00442/.

Appendix 1 – Fisher’s score calculation procedure

```
scored = [] indices = {} shps = {} for cl in classes:
```

```
indices[cl] = df_fish['class'] == cl
```

```
shps[cl] =
```

```
df_fish[indices[cl]].shape[0] for col in
```

```
df_fish.columns: if col == 'class':
```

```
continue num = 0 den = 0 m =
```

```
df_fish[col].mean() for cl in classes:
```

```
num += (shps[cl] / df_fish.shape[0]) * (m
```

```
- df_fish[indices[cl]][col].mean())**2 den
```

```
+= (shps[cl] / df_fish.shape[0]) *
```

```
df_fish[indices[cl]][col].var() score =
```

```
{'feature': col, 'score': num / den}
```

```
scored.append(score)
```

Appendix 2 – Autoencoder model creation function def

```
create_model(input_dim):          inp          =
Input(shape=(input_dim,))
encoder = Dense(int(math.ceil(0.75 * input_dim)), activation="tanh")(inp)
encoder = Dense(int(math.ceil(0.5 * input_dim)),
activation="tanh")(encoder)
encoder      =      Dense(int(math.ceil(0.25      *      input_dim)),
activation="tanh")(encoder)
decoder      =      Dense(int(math.ceil(0.5      *      input_dim)),
activation="tanh")(encoder)
decoder = Dense(int(math.ceil(0.75 *
input_dim)), activation="tanh")(decoder)
decoder = Dense(input_dim)(decoder) return
Model(inp, decoder)
```

Appendix 3 – Deep Neural Network model creation function def

```
create_model(input_dim, hidden_layer_size, num_of_classes):  
model = Sequential()  
model.add(Dense(hidden_layer_size, activation="tanh",  
input_shape=(input_dim,)))  
model.add(Dense(hidden_layer_size, activation="tanh"))  
model.add(Dense(num_of_classes))  
model.add(Activation('softmax'))  
  
return model
```