Intrusion Detection System – Using Neural Network

Sai Tarun Sathyan (ss4005)

I. Executive Summary

Network Security is a major expense for organizations trying to protect their data. The intrusion detection system is one part of that, which helps identify and track abnormal network activity and helps track malicious signatures. Since an Intrusion detection system plays such a huge role in network/system security it has to be constantly updated. Our project focuses on building an IDS that is trained with a dataset from the knowledge discovery in databases also known as KDD. It works with raw network data containing different types of packets such as TCP, UDP and ICMP packets. Our neural network is trained to identify 10 different attack types.

I developed a neural network model with 79 neurons in the input layer and 12 neurons in the output layer. In total the model runs through 6 million packets to classify each one. I first perform data clean-up and then compile the cleaned data into one folder. For the data clean-up I just remove all packets that have Nan or Inf or empty variables, since those packets affect the model's learning. The data clean and compilation is done in two different ways, for anomaly-based csv I mapped all normal packets to label 0 and all attack packets to label 1. For signature based compiled csv I labelled all attacks appropriately. After data clean-up and compilation, I am left with two big folders for anomaly based and signature-based testing/training.

Next, I create our neural network model and train it on this cleaned and compiled csv file with around 6 million packets in it. The training is done using Adam optimiser and for 10 epochs and batch size of 10 . The neural network trains and tests on two different datasets, one for anomaly-based and another for signature-based detection. The neural network took me around 1 hour to train and the accuracy rating is as follows:

Signature based: 95.43%

Anomaly based: 97.94%

The false positive and false negative ratios are as follows:

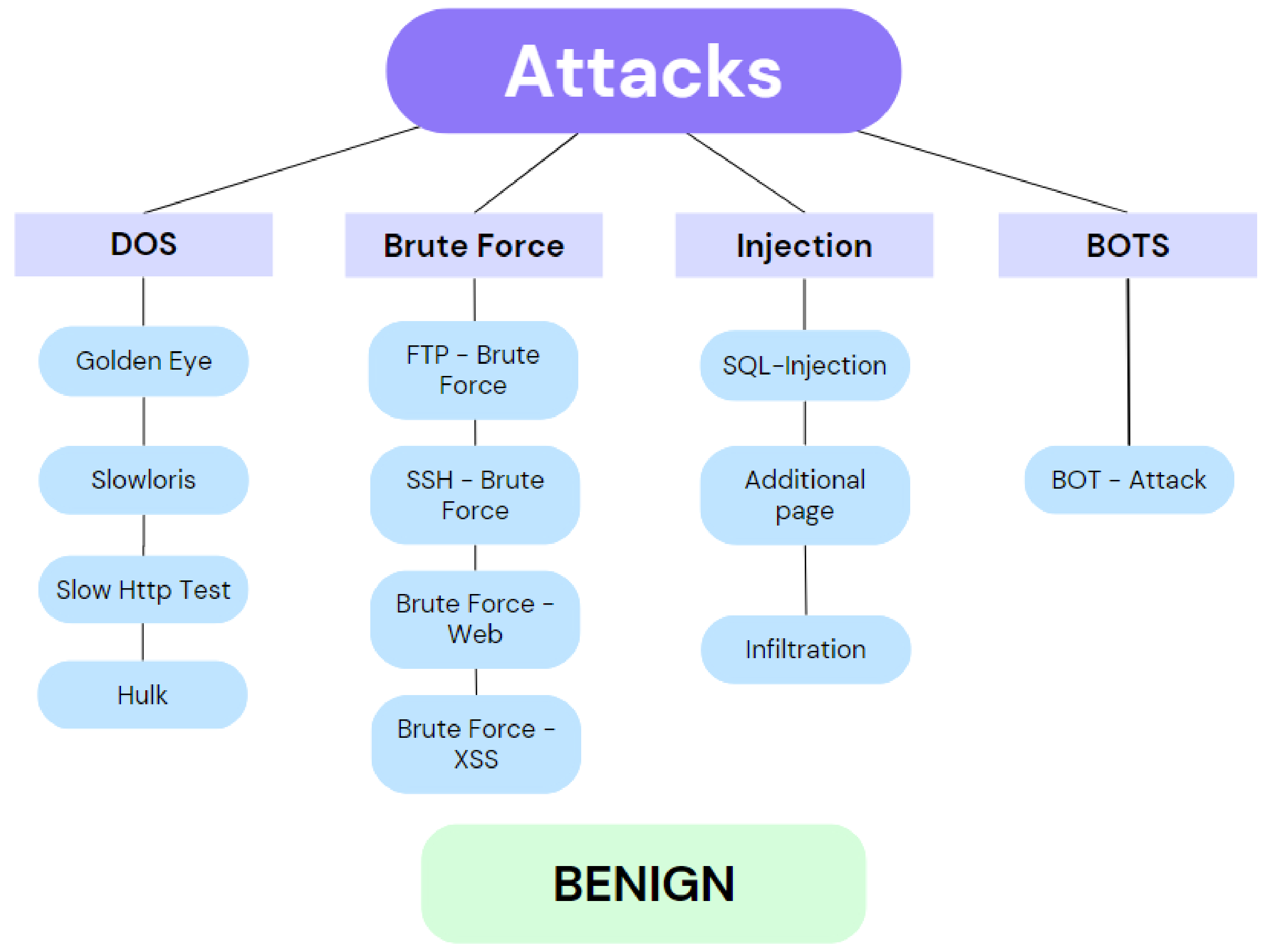
False Positive: 0.8%

False Negative: 1.4%

II. Specification

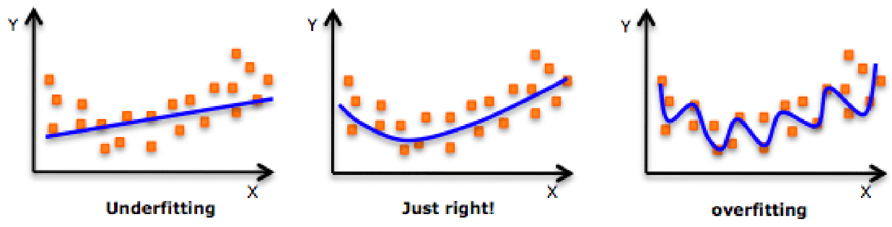
Our goal was to create a program to clean up a huge dataset and then create a sequential neural network model that is capable of identifying signature-based and anomaly-based attacks from this cleaned dataset. Our neural network model consists of 3 dense layers with the 1^{st} two layers having a "***ReLU***" activation function and the final output layer having a "***Softmax***" activation function. Major benefits of ReLUs are sparsity and a reduced likelihood of vanishing gradient. ReLu is faster to compute than the sigmoid function, and its derivative is faster to compute. The Softmax activation function is able to handle multiple classes. It normalizes the outputs for each class between 0 and 1 and divides by their sum. Hence forming a probability distribution. Therefore, giving a clear probability of input belonging to any particular class. The neural network model uses Adam optimizer simply because it's the best optimizer and has the fastest computation time.

The different types of packets I encountered are as follows:



I used a the most recent CIC dataset to train our packets. The data set had 7GB worth of packet data in it. Each packet had 80 attributes describing the packet.

As I trained our model, I also needed to make sure that I were not **over-fitting** the model or over training it and also wanted to avoid selection bias. In Overfitting, the model tries to learn too many details in the training data along with the noise from the training data. As a result, the model performance is very poor on unseen or test datasets.

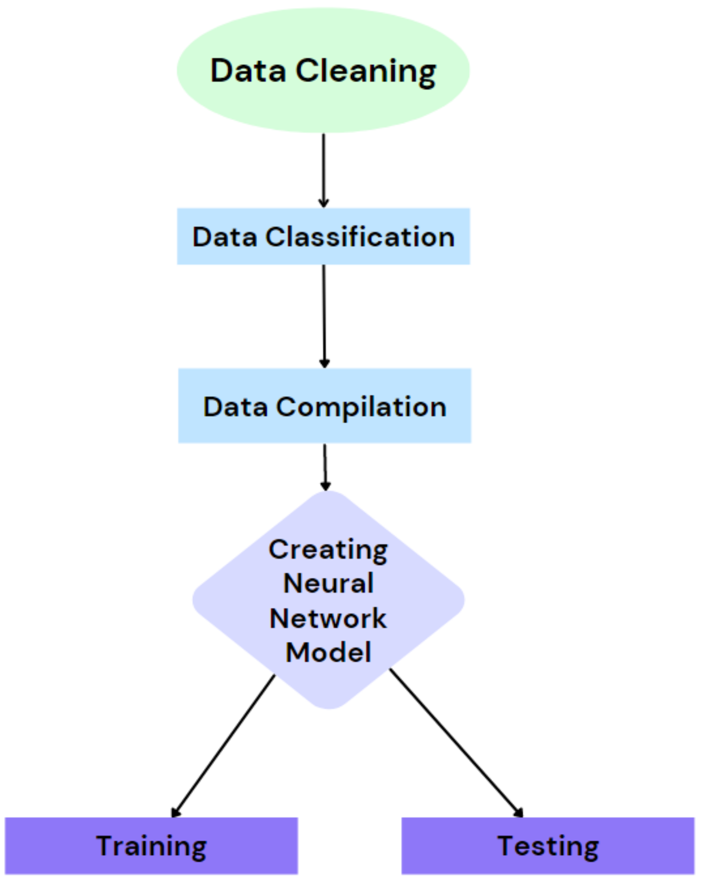


I also wanted to avoid **selection bias** in our model. Since selection bias occurs if a data set's examples are chosen in a way that is not reflective of their real-world distribution.

III. Methods, Techniques & Implementation

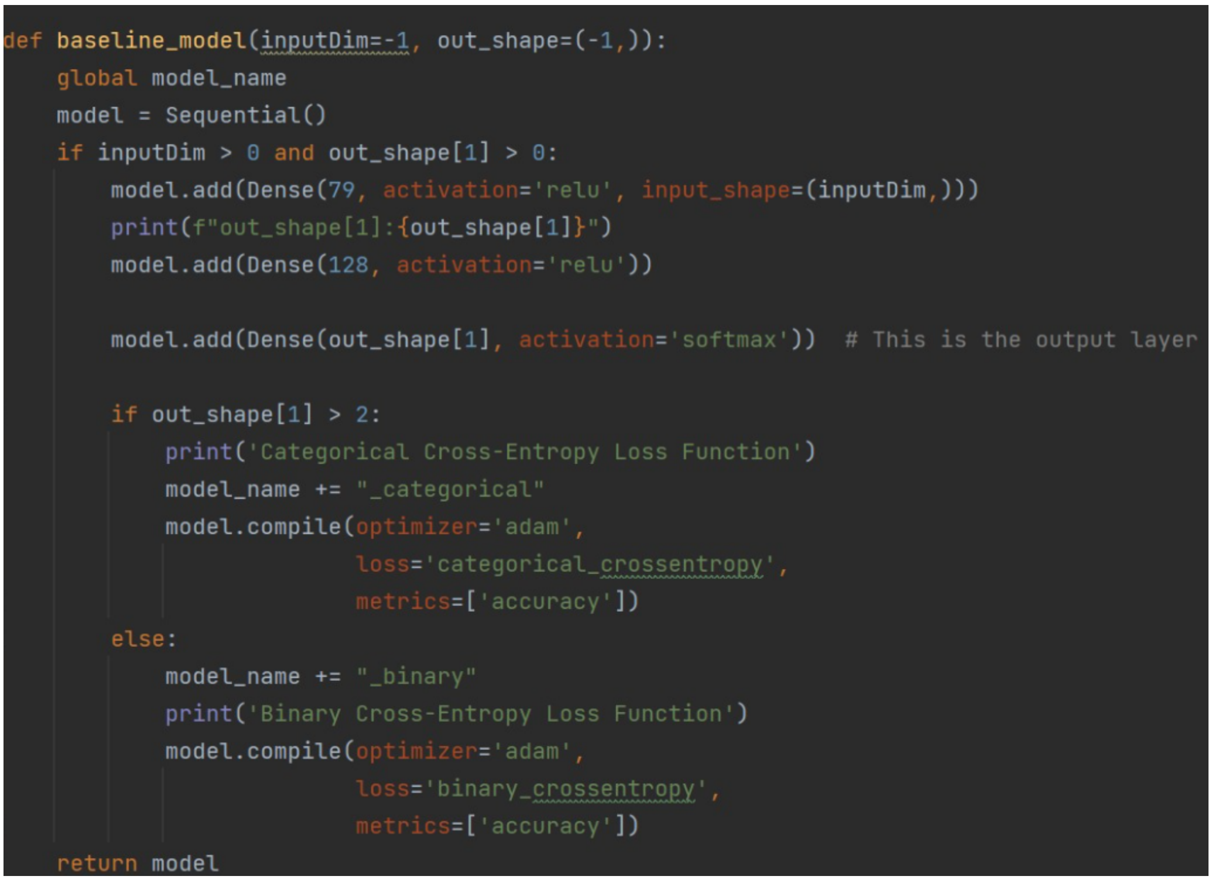
I used a neural network with Adam optimizer with 10 epochs, 10 batch size and a learning rate of 0.001. The neural network had 3 dense layers.

Our project workflow can be split into six steps:



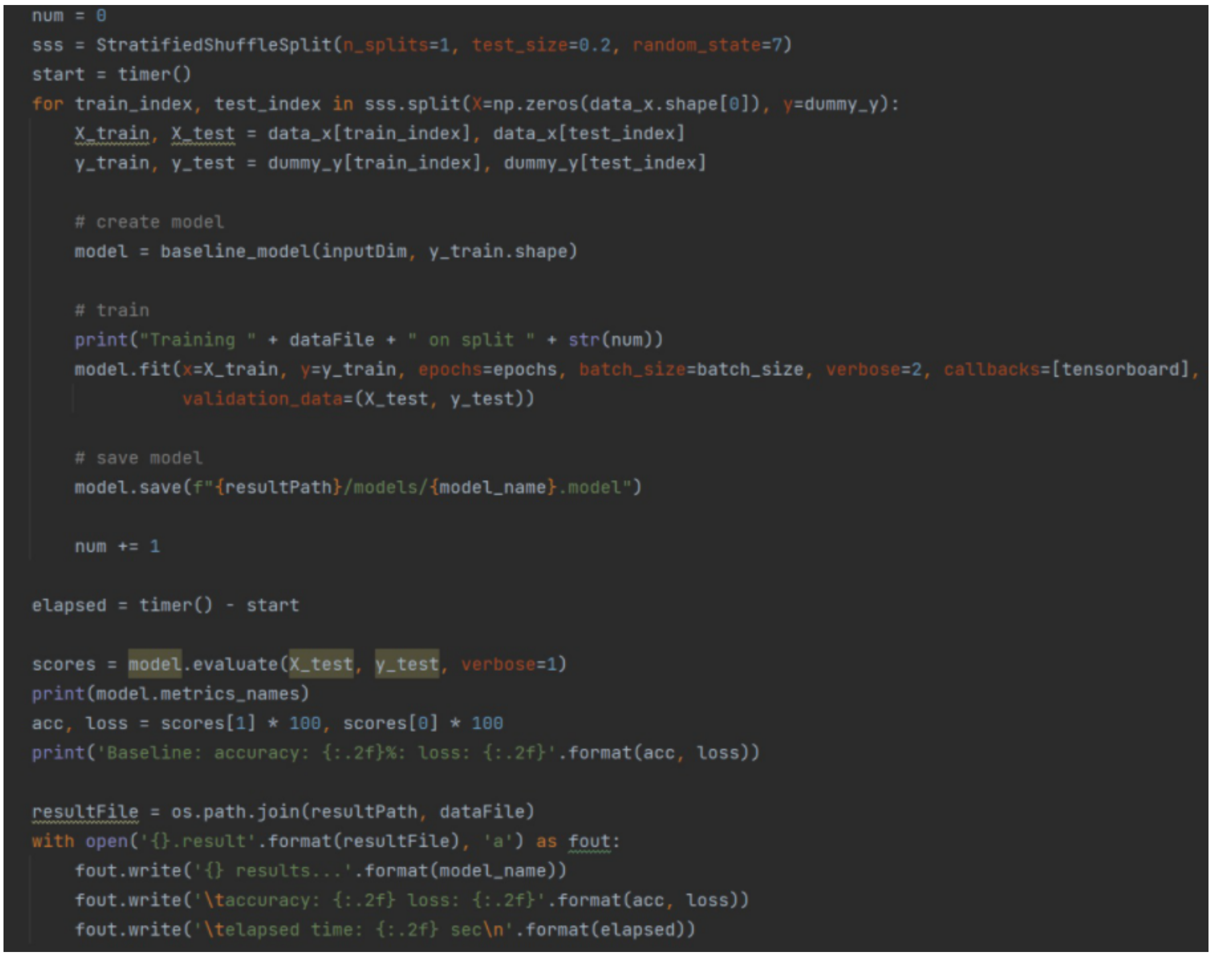
I did data cleaning in one program and data classification/compilation in another separate program. For data cleaning I iterated through each file and removed all “Nan”, “Inf” and empty rows. For classification and compilation, I first joined all CSV files containing multiple packets with different attack types and classified into a signature-based and anomaly-based csv files. For signature based all the packets will be labelled appropriately whereas in the anomaly based one the benign packets are labelled 0 and the attack type packets are labelled as 1.

After data processing has been completed; I created, trained and tested our model in a separate program. The model creation function looks like this:



```
def baseline_model(inputDim=-1, out_shape=(-1,)):
    global model_name
    model = Sequential()
    if inputDim > 0 and out_shape[1] > 0:
        model.add(Dense(79, activation='relu', input_shape=(inputDim,)))
        print(f"out_shape[1]:{out_shape[1]}")
        model.add(Dense(128, activation='relu'))

        model.add(Dense(out_shape[1], activation='softmax'))  # This is the output layer

        if out_shape[1] > 2:
            print('Categorical Cross-Entropy Loss Function')
            model_name += "_categorical"
            model.compile(optimizer='adam',
                          loss='categorical_crossentropy',
                          metrics=['accuracy'])
        else:
            model_name += "_binary"
            print('Binary Cross-Entropy Loss Function')
            model.compile(optimizer='adam',
                          loss='binary_crossentropy',
                          metrics=['accuracy'])
    return model
```

Our model was tested and trained twice. Once for anomaly-based dataset and once for signature-based dataset. I split the training and testing data into an 80:20 split. It ran for 10 epochs and performed cross validation on each epoch. The model training and testing code looks like this:



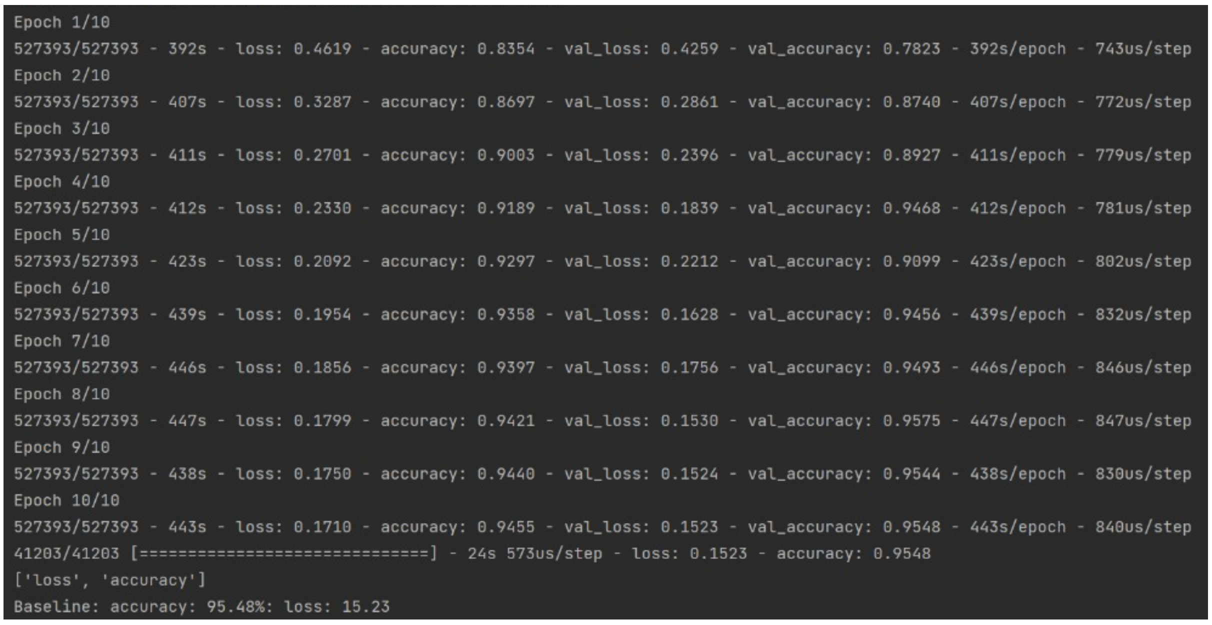
```python
num = 0
sss = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=7)
start = timer()
for train_index, test_index in sss.split(X=np.zeros(data_x.shape[0]), y=dummy_y):
    X_train, X_test = data_x[train_index], data_x[test_index]
    y_train, y_test = dummy_y[train_index], dummy_y[test_index]

    # create model
    model = baseline_model(inputDim, y_train.shape)

    # train
    print("Training " + dataFile + " on split " + str(num))
    model.fit(x=X_train, y=y_train, epochs=epochs, batch_size=batch_size, verbose=2, callbacks=[tensorboard],
              validation_data=(X_test, y_test))

    # save model
    model.save(f"{resultPath}/models/{model_name}.model")

    num += 1

elapsed = timer() - start

scores = model.evaluate(X_test, y_test, verbose=1)
print(model.metrics_names)
acc, loss = scores[1] * 100, scores[0] * 100
print('Baseline: accuracy: {:.2f}%: loss: {:.2f}'.format(acc, loss))

resultFile = os.path.join(resultPath, dataFile)
with open('{}.result'.format(resultFile), 'a') as fout:
    fout.write('{} results...'.format(model_name))
    fout.write('\taccuracy: {:.2f} loss: {:.2f}'.format(acc, loss))
    fout.write('\telapsed time: {:.2f} sec\n'.format(elapsed))
```

In order to prevent **over-fitting** and avoid **selection bias,** I used 10-fold cross validation techniques to validate the model. With this method I have one data set which I divide randomly into 10 parts. I use 9 of those parts for training and reserve one tenth for testing. I repeat this procedure 10 times each time reserving a different tenth for testing. The purpose of cross–validation is to test the ability of a machine learning model to predict new data. It is also used to flag problems like overfitting or selection bias and gives insights on how the model will generalize to an independent dataset.

After training and testing I saved this model for future testing purposes. The program I developed is on python and I used the PyCharm IDE to develop this Intrusion Detection System. The Main libraries I used are CSV, Keras, TensorFlow and NumPy. The limitation of our program is that it takes a really long time to train the model due to the sheer size of the dataset I am feeding it. I had an intel i7 processor and a GTX 3070 graphics card and training took almost 1 hour for each dataset (signature/anomaly). But this ensures it learns well and produces an accuracy of 98% for signature-based classification and 95% for anomaly-based classification.

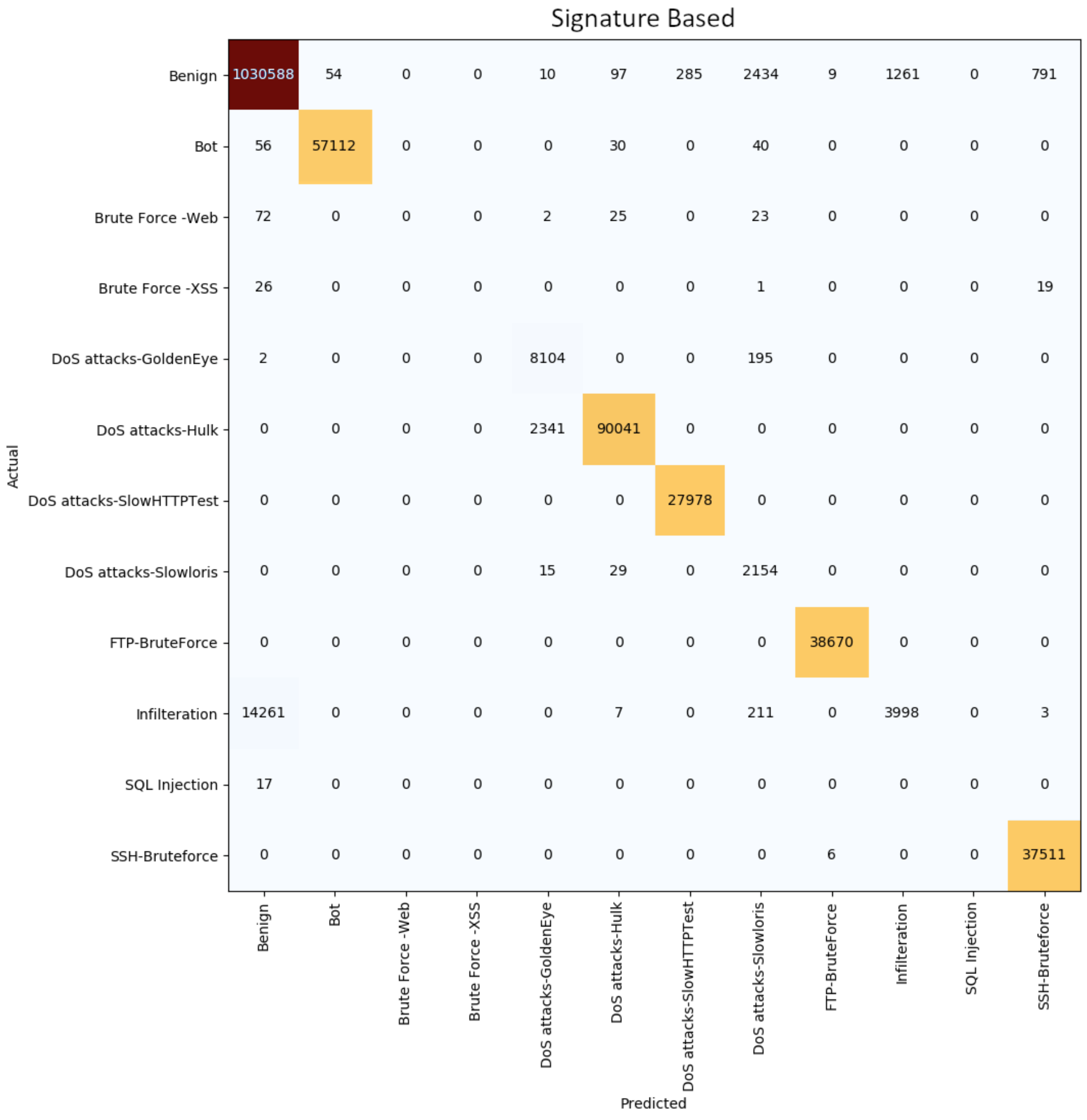
IV. Tests & Results

The Training and testing results for the signature-based model looks like this:



```
Epoch 1/10
527393/527393 - 392s - loss: 0.4619 - accuracy: 0.8354 - val_loss: 0.4259 - val_accuracy: 0.7823 - 392s/epoch - 743us/step
Epoch 2/10
527393/527393 - 407s - loss: 0.3287 - accuracy: 0.8697 - val_loss: 0.2861 - val_accuracy: 0.8740 - 407s/epoch - 772us/step
Epoch 3/10
527393/527393 - 411s - loss: 0.2701 - accuracy: 0.9003 - val_loss: 0.2396 - val_accuracy: 0.8927 - 411s/epoch - 779us/step
Epoch 4/10
527393/527393 - 412s - loss: 0.2330 - accuracy: 0.9189 - val_loss: 0.1839 - val_accuracy: 0.9468 - 412s/epoch - 781us/step
Epoch 5/10
527393/527393 - 423s - loss: 0.2092 - accuracy: 0.9297 - val_loss: 0.2212 - val_accuracy: 0.9099 - 423s/epoch - 802us/step
Epoch 6/10
527393/527393 - 439s - loss: 0.1954 - accuracy: 0.9358 - val_loss: 0.1628 - val_accuracy: 0.9456 - 439s/epoch - 832us/step
Epoch 7/10
527393/527393 - 446s - loss: 0.1856 - accuracy: 0.9397 - val_loss: 0.1756 - val_accuracy: 0.9493 - 446s/epoch - 846us/step
Epoch 8/10
527393/527393 - 447s - loss: 0.1799 - accuracy: 0.9421 - val_loss: 0.1530 - val_accuracy: 0.9575 - 447s/epoch - 847us/step
Epoch 9/10
527393/527393 - 438s - loss: 0.1750 - accuracy: 0.9440 - val_loss: 0.1524 - val_accuracy: 0.9544 - 438s/epoch - 830us/step
Epoch 10/10
527393/527393 - 443s - loss: 0.1710 - accuracy: 0.9455 - val_loss: 0.1523 - val_accuracy: 0.9548 - 443s/epoch - 840us/step
41203/41203 [==============================] - 24s 573us/step - loss: 0.1523 - accuracy: 0.9548
['loss', 'accuracy']
Baseline: accuracy: 95.48%: loss: 15.23
```

The confusion matrix for true/false positives is as follows:

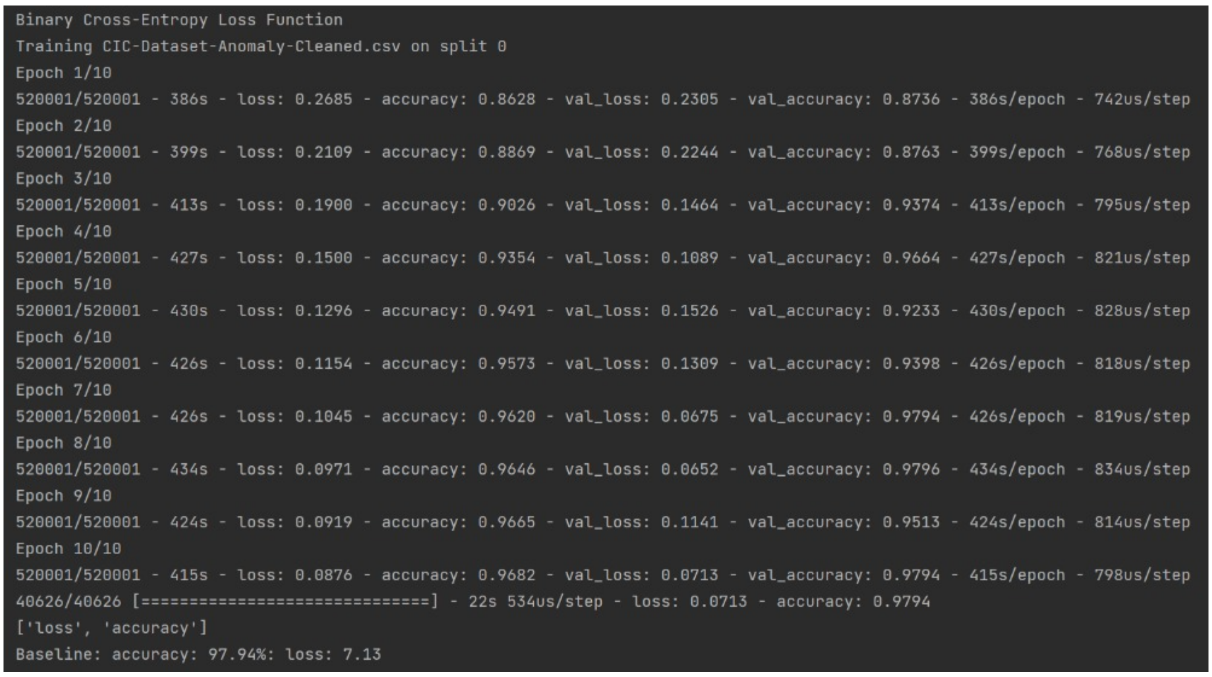


Signature Based

	Predicted											
Actual	Benign	Bot	Brute Force -Web	Brute Force -XSS	DoS attacks-GoldenEye	DoS attacks-Hulk	DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce	Infilteration	SQL Injection	SSH-Bruteforce
Benign	1030588	54	0	0	10	97	285	2434	9	1261	0	791
Bot	56	57112	0	0	0	30	0	40	0	0	0	0
Brute Force -Web	72	0	0	0	2	25	0	23	0	0	0	0
Brute Force -XSS	26	0	0	0	0	0	0	1	0	0	0	19
DoS attacks-GoldenEye	2	0	0	0	8104	0	0	195	0	0	0	0
DoS attacks-Hulk	0	0	0	0	2341	90041	0	0	0	0	0	0
DoS attacks-SlowHTTPTest	0	0	0	0	0	0	27978	0	0	0	0	0
DoS attacks-Slowloris	0	0	0	0	15	29	0	2154	0	0	0	0
FTP-BruteForce	0	0	0	0	0	0	0	0	38670	0	0	0
Infilteration	14261	0	0	0	0	7	0	211	0	3998	0	3
SQL Injection	17	0	0	0	0	0	0	0	0	0	0	0
SSH-Bruteforce	0	0	0	0	0	0	0	0	6	0	0	37511

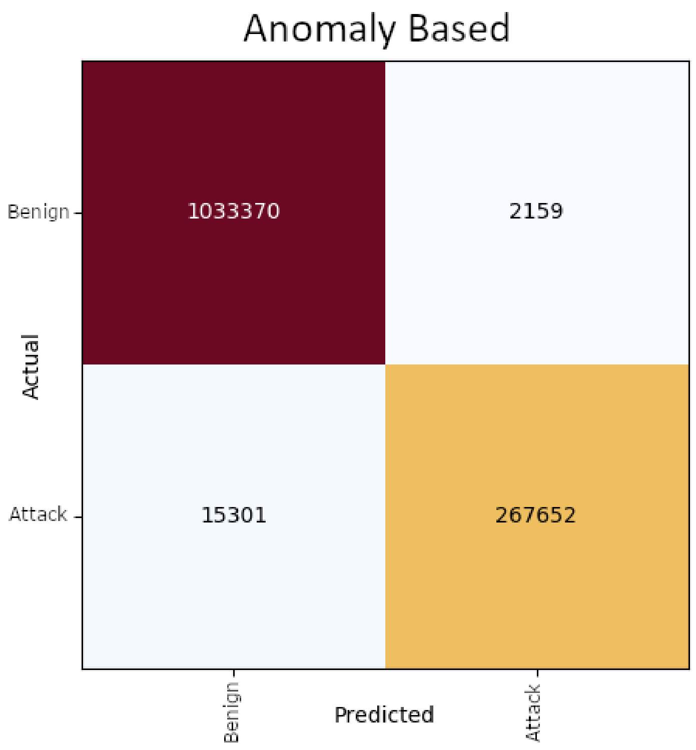
Here I can see the false positive and false negative rates for each attack type

The Training and testing results for the anomaly-based model looks like this:



```
Binary Cross-Entropy Loss Function
Training CIC-Dataset-Anomaly-Cleaned.csv on split 0
Epoch 1/10
520001/520001 - 386s - loss: 0.2685 - accuracy: 0.8628 - val_loss: 0.2305 - val_accuracy: 0.8736 - 386s/epoch - 742us/step
Epoch 2/10
520001/520001 - 399s - loss: 0.2109 - accuracy: 0.8869 - val_loss: 0.2244 - val_accuracy: 0.8763 - 399s/epoch - 768us/step
Epoch 3/10
520001/520001 - 413s - loss: 0.1900 - accuracy: 0.9026 - val_loss: 0.1464 - val_accuracy: 0.9374 - 413s/epoch - 795us/step
Epoch 4/10
520001/520001 - 427s - loss: 0.1500 - accuracy: 0.9354 - val_loss: 0.1089 - val_accuracy: 0.9664 - 427s/epoch - 821us/step
Epoch 5/10
520001/520001 - 430s - loss: 0.1296 - accuracy: 0.9491 - val_loss: 0.1526 - val_accuracy: 0.9233 - 430s/epoch - 828us/step
Epoch 6/10
520001/520001 - 426s - loss: 0.1154 - accuracy: 0.9573 - val_loss: 0.1309 - val_accuracy: 0.9398 - 426s/epoch - 818us/step
Epoch 7/10
520001/520001 - 426s - loss: 0.1045 - accuracy: 0.9620 - val_loss: 0.0675 - val_accuracy: 0.9794 - 426s/epoch - 819us/step
Epoch 8/10
520001/520001 - 434s - loss: 0.0971 - accuracy: 0.9646 - val_loss: 0.0652 - val_accuracy: 0.9796 - 434s/epoch - 834us/step
Epoch 9/10
520001/520001 - 424s - loss: 0.0919 - accuracy: 0.9665 - val_loss: 0.1141 - val_accuracy: 0.9513 - 424s/epoch - 814us/step
Epoch 10/10
520001/520001 - 415s - loss: 0.0876 - accuracy: 0.9682 - val_loss: 0.0713 - val_accuracy: 0.9794 - 415s/epoch - 798us/step
40626/40626 [==============================] - 22s 534us/step - loss: 0.0713 - accuracy: 0.9794
['loss', 'accuracy']
Baseline: accuracy: 97.94%: loss: 7.13
```

The confusion matrix for true/false positives is as follows:



Here I can see the false positive and false negative ratio for true attacks and benign packets

VI. Conclusion & Future Scope

This intrusion detection system model has been trained to detect 12 different types of attacks and also manages to perform signature-based and anomaly-based detection. In future I could try to add more hidden layers and improve the speed of training for the model and also train this model on other attack types to broaden its detection scope. The current neural network model is a sequential one. If I get a dataset with a similar format but different attack types, I could further train it and increase its detection range. The current false positive rate is 0.8% and false negative rate is 1.4%. I could try and improve this further .