Data Science Toolbox Assessed Coursework 4

Kishalay Banerjee, Shanglin Zou, Katarzyna Staniewicz

March 2019

Abstract

In this report, we use the 'Detection of Botnet Attacks' dataset from the UCI Machine Learning Repository. A single Neural Networks Model (for Anomaly Detection), is developed, using Tensorflow. The main objective of this model is to differentiate between benign and malicious traffic, with a high level of precision. Three variations of this scenario are considered - based on the training - testing data split. Finally, the three versions of the model are compared to obtain an understanding of how well the model works under different scenarios.

1 Equity

We have chosen the following **equity split**:

- Kishalay Banerjee- 35%
- Shanglin Zou - 35%
- Katarzyna Staniewicz - 30%

2 Introduction

In this assignment, our objective is to carry out Deep Learning in a Cybersecurity context. After some brainstorming sessions, we decided to keep our inference goals simple, but interesting.

Technology has now become an integral part of people's lives. There are increasing number of instances of people and commercial enterprises opting for Home Automation. The system is based upon a central processor, such as a microprocessor-based computer, and is connected by means of a data bus to control various products and subsystems within a home or commercial building, such as lighting systems, security systems, various sensors, multiple external terminals, as well as to allow for the input of commands by a variety of means such as touchscreens, voice recognition systems, telephones, custom switches or any device capable of providing an input to a computer system. [4]

Thus, as more people opt for this technology, the chances of cyber attacks on the devices in the network also increase exponentially. Thus, it is extremely important to have a security system in place, which can monitor the feed from the various digital devices in the network, and be able to flag any sort of incoming malicious traffic. Deep learning is an appropriate method for implementing such a model.

Deep Learning is a a part of a broader family of machine learning methods which are based on learning data representations, as opposed to task based algorithms. It can be supervised, semi-supervised, or unsupervised.

This technique allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. It discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer.

This is carried out by composing simple but non linear modules that each transform the representation at one level into a representation at a higher and more abstract level. With enough such transformations, complex functions can be learned. For classification problems, higher layers of representation amplify aspects of the input that are important to discriminate, and suppress irrelevant variations. The key aspect of deep learning is that the layers of features are not designed by human engineers, rather, they are learned from data using a general purpose learning procedure. [5]

3 Using BlueCrystal

One of the suggestions for this assignment was that we utilise the resources and computing power of BlueCrystal Phase 4, the supercomputing facility available at the University. It is primarily intended for projects requiring large amounts of parallel computing.

Keeping this suggestion in mind, we deliberately chose a large data set. Computations on this data set would not be possible on our laptops alone. So, to debug our code, we usually used a very small subset of our data, and ran it on our laptop. The main analysis of the entire data was done on BlueCrystal, by submitting jobs to the GPU Head Node.

4 Dataset

The data set we have used is the 'Detection of Botnet Attacks' from the UCI Machine Learning Repository. It consists of data from commercial IoT (Internet of Things) devices which were infected with 2 common botnets - Mirai and Bashlite. This data set was created to facilitate research in new methods to detect attacks launched from compromised IoT devices. [6]

In our project, we have used data from 4 IoT devices - Doorbell, Thermostat, Webcam and Baby Monitor. Corresponding to each device, there were 115 columns of numeric data. This included a large number of features, whose abbreviated forms were used as column names. An explanation of the column names is provided below -
Stream aggregation:

- H: Stats summarizing the recent traffic from this packet's host (IP)
- HH: Stats summarizing the recent traffic going from this packet's host (IP) to the packet's destination host.
- HpHp: Stats summarizing the recent traffic going from this packet's host+port (IP) to the packet's destination host+port.
- HHjit: Stats summarizing the jitter of the traffic going from this packet's host (IP) to the packet's destination host.

Time-frame (The decay factor Lambda used in the damped window): How much recent history of the stream is capture in these statistics L5, L3, L1, ...

The statistics extracted from the packet stream:

- weight: The weight of the stream (can be viewed as the number of items observed in recent history)
- mean: The mean of the stream
- std: The standard deviation of the stream
- radius: The root squared sum of the two streams' variances
- magnitude: The root squared sum of the two streams' means
- cov: an approximated covariance between two streams
- pcc: an approximated covariance between two streams [13]

Each of the devices contained a data file for benign traffic, and 5 files corresponding to 5 different attacks which were named - Combo, Scan, Junk, TCP, UDP. These attack names, along with the device, were part of the file names. Thus, when the data files were read into Python, we created two additional columns which contained the traffic label and device label for each observation. The data files for each of the devices were then merged to create a single large data set.

5 Neural Networks

We have chosen to create a Neural Networks model, using Tensorflow. Tensorflow is an end-to-end open source platform used for machine learning. Furthermore, we have used Keras, which is a high level API used for building and training models in Tensorflow.

A neural network can be defined to be a composite function. The following parts make up the architecture of a basic neural network -

Units/Neurons - Neurons are functions which contain weights and biases and perform certain functions on the input data. They also restrict the data to a range, using an activation function.

Hyper-parameters - These are certain values which are set by the programmer. Based on the output returned by the network, these values may be changed to fine tune its performance.

Activation Functions - These are also known as Mapping Functions. They take some input on the x-axis, and usually output a value in a restricted range. They are used to convert large outputs from the neurons into a smaller value, and promote non-linearity in the neural network. There are various types of activation functions available, namely, Relu, Sigmoid, Softmax etc.

Layers - Layers refer to the number of 'sets' of neurons which make up the neural network. The first and last layers are the Input and Output Layers of the network. [9]

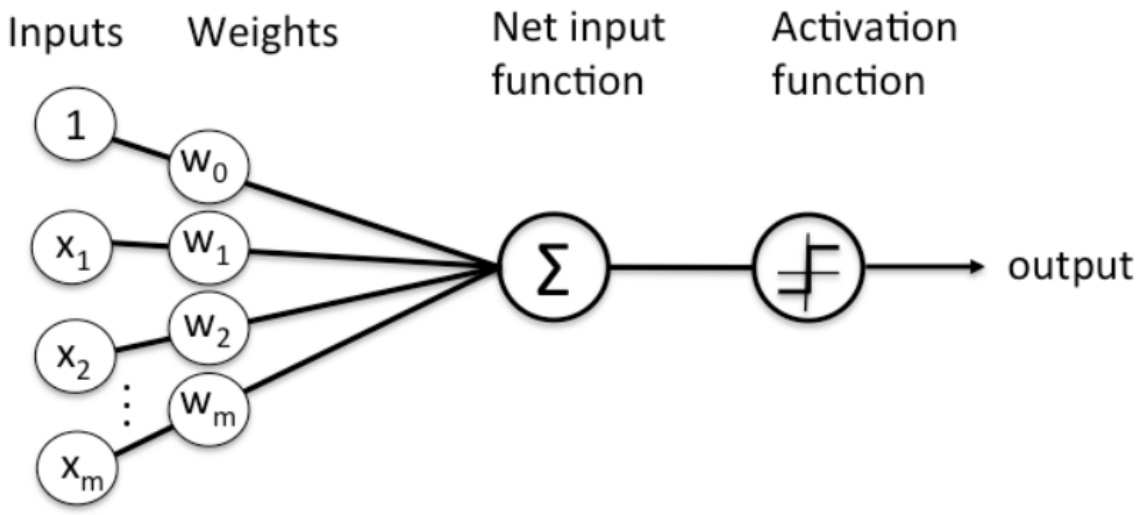

Figure 1: A Typical Neural Network

A neural network learns features of the data by the backpropagation algorithm. This algorithm repeatedly adjusts the weight of the connections in the network so as to minimise the measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to identify new, useful features is a distinguishing characteristic of the Backpropagation Algorithm from other earlier methods. [8]

6 Our Model

In our project, we have considered 3 variants of a Neural Networks model.

- The first model takes in the entire data as input. It then performs the usual functions of splitting it into a training and a test set, learning features of the data based on the training set, and predicting the labels of the test data set. The efficacy of the model is then calculated by using a suitable metric of performance.
- The second model is also written along the same lines. However, in this instance, the observations corresponding to certain specific attacks are removed from the training data. However, the test

data set remains the same. The objective is to investigate if the model can recognise a certain attack which it hasn't seen before.

- The third model is similar to the second model, but instead of removing a certain attack, we remove all the observations corresponding to a particular device. Then, the analysis is the same as the second model. Our objective here is to investigate if the model can classify traffic from a particular device, without having come across data from that device before.

In all three models, we have used the same training and test data split to facilitate sensible comparison.

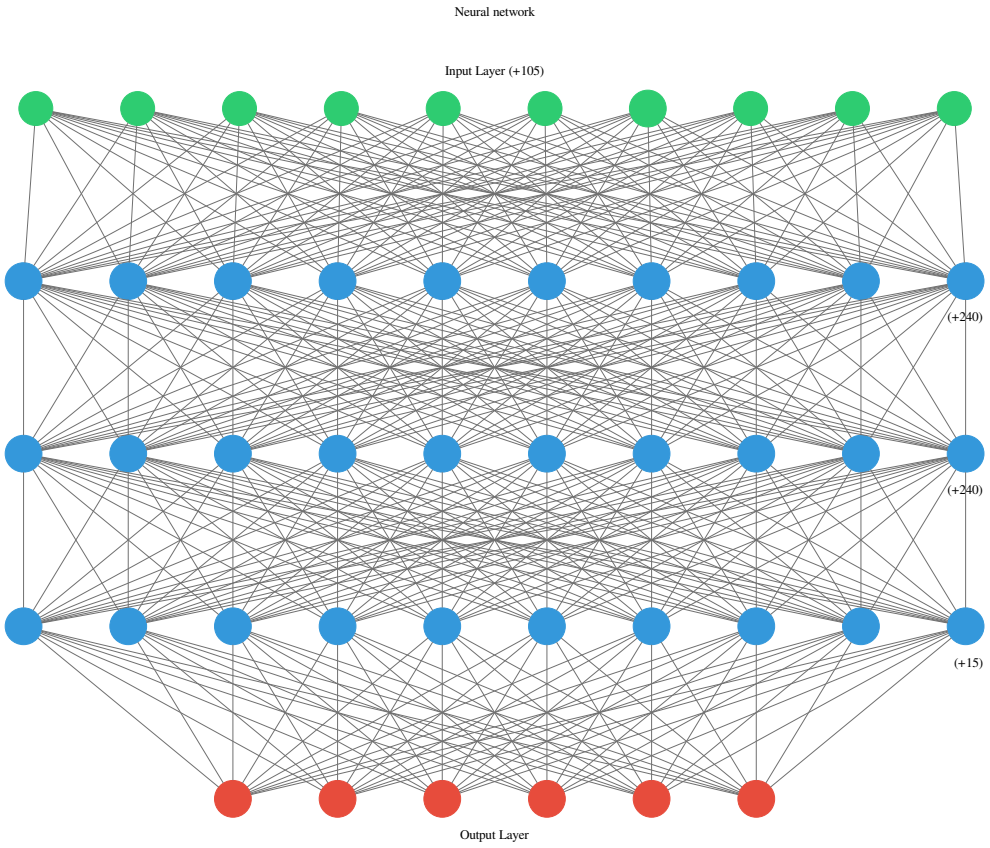

Figure 2: A Neural Network visualisation generated using ann-viz package[10] in Python. Green points represent the features of our data. Blue points are the 3 layers of the neural network. Red points represents the types of the attacks.

6.1 Data Preprocessing

It is always mentioned that the data provided to a neural network as the input needs to be normalised and scaled. According to Stack Overflow [14], this needs to be done because -

- This makes training faster and allows a faster approach to global minima at error surface.
- Weight decay and Bayesian Estimation can be done more conveniently with standardised inputs.

For the purpose of standardisation, we made use of the StandardScaler feature. This worked quite well, and we were able to carry out data preprocessing on the entire data set. Our data set had a mixture of numeric and text columns (attack labels and device labels). So, before data preprocessing, we had to ensure that we extracted only the numeric columns from the data set.
We did not drop any values because we ran a function that told us that there were no columns which had a single unique value.

6.2 Merging of Two Attacks

In our data, we originally had 6 types of traffic - benign and 5 different attacks. These were - Scan, Junk, Combo, TCP and UDP. When we ran our model, we noticed that TCP and UDP were consistently getting misclassified with each other. We tried tuning the parameters of the model, but the problem remained unresolved.

As a check, we ran the model in a binary classification scenario, consisting only of TCP and UDP data. The results from that model were also unsatisfactory, with an accuracy of about 0.5.

Then, we considered the hypothesis that the data corresponding to TCP and UDP were so similar, that the model could not distinguish between them satisfactorily. To test this mathematically, we decided to opt for a non-parametric test (since the distribution of the data was unknown).

The test we applied was the Kruskal-Wallis test. The Kruskal-Wallis test is a non-parametric method for testing if samples originate from the same distribution. It is used for comparing two or more independent samples. After applying this test on each pair of rows (paired by index number) of the TCP and UDP data sets, we obtained a set of p values. To obtain some valid ineferences from this multiple testing scenario, we applied the Bonferroni correction to this list of p values. The Bonferroni correction rejects the null hypothesis for each i such that,

$$p_i \leq \frac{\alpha}{m}.$$

Here, m is the total number of tests and α is our desired level of significance. We have chosen $\alpha = 0.05$. Based on the result of the Bonferroni correction, we claimed that our hypothesis was justified, and merged the data sets for TCP and UDP.

6.3 One Hot Encoding

In our data, we have a column of attack labels. Thus, this falls under categorical data. However, this creates a problem because neural networks cannot work with label data directly. It needs all input and output variables to be numeric. Thus, we had to convert our label data into numeric. The two ways to do that are -

- Integer Encoding - In this technique, each unique category value is assigned an integer value. For example, if we used integer encoding in our model, we might have mapped benign-0, scan-1, combo-2 etc. However, when the mapping is done in this manner, the natural ordering of the integers is often taken into account by the machine learning algorithm, and exploited appropriately. This would be a problem in our case since, the integers would represent different attacks, which were not related to each other originally.

- One Hot Encoding - This method is suggested for categorical variables where ordinal relationships do not exist. In this technique, the integer variable is removed and a new binary variable is added for each unique integer value. These binary variables are often termed as 'dummy variables'. [1]

6.4 Choice of Model Parameters

The choice of the model parameters often play a critical role in determining the effectiveness of a neural network. We tried a variety of combinations of batch sizes, loss functions and activation functions. To obtain our results within a reasonable computation time, we decided to use 5 epochs for each model.

- Batch Sizes - Keeping the value of the epochs constant, we tried out a variety of batch sizes to see which one gave the best accuracy. The batch size is defined as the number of samples that would be propagated through the network in one go. According to literature we found on Stack Exchange [2], the network trains faster when batches of smaller size are used, because the weights in the network are updated after each propagation, which lead to more accurate feature identification. Since we have around 1.3 million data points, we ran the model with various batch sizes (for 50 epochs each), to identify the batch size which gave maximum accuracy. The graph of Batch Sizes vs Accuracy is provided below -

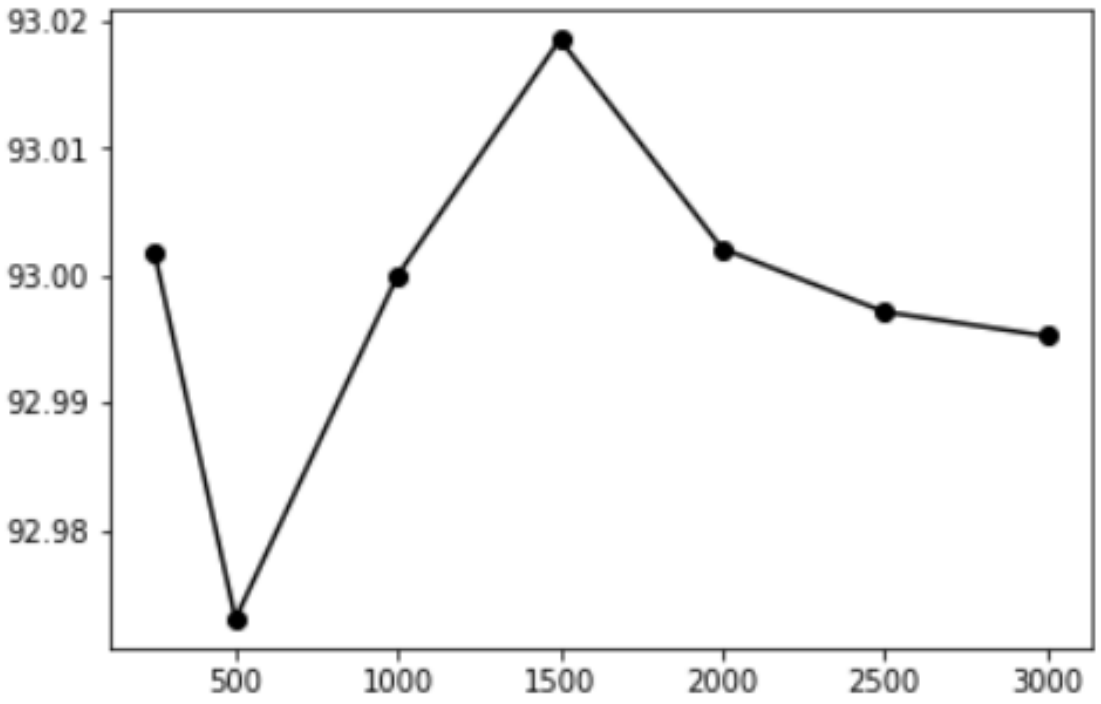

Figure 3: Batch Size vs Accuracy

From the graph, it is evident that the accuracy is maximised at a batch size of 1500. So, for all our subsequent models, we have proceeded with a batch size of 1500.

- Loss Functions - In a neural network, the function used to evaluate a particular solution is termed as a Loss Function. Improvements in the model are obtained by minimising the value returned by the loss function.

  In the final layer of our neural network, we are predicting the attack label (or device label) from multiple classes. In this type of a model, 'Cross-Entropy' was suggested as the Loss Function [7]. However, through trial and error, we found that using Binary Cross-Entropy gave much better results than simply using 'Cross-Entropy'. So, we persisted with Binary Cross-Entropy in our model.

- Activation Functions - The function which is used to obtain the output of a node in a neural network is called the activation function. Activation Functions can be linear or non-linear. However, non-linear activation functions are preferred because they make it easy for the model to generalise or adapt with a variety of data, and to differentiate between the output. It was suggested that the Softmax function was ideal for multiclass classification [11], and so, we tried out a lot of combinations of functions with Softmax as the one in the Output layer. The combination which gave us the best results has been used in our model.

- Optimizers - Optimizers are algorithms which are used to minimise the value of the Loss Function in the model. Optimization algorithms can be first order or second order (using Gradient Descent or Hessian for minimisation) respectively. We have used the Adam optimizer, which is a first order algorithm, since it uses the gradient descent technique for optimization. It is also straightforward to implement, computationally efficient, invariant to diagonal rescaling of the gradients, and well suited for problems with large amounts of data and parameters [3]. So, it was an ideal fit for our model.

6.5 Choice of Performance Metrics

The choice of the performance metric of a model is crucial, since it allows us to properly interpret the results obtained. In our model, we have variously used the following performance metrics -

- Confusion Matrices - For any classification problem, the most straightforward tool which is used is the Confusion Matrix.
  A confusion matrix is a table that contains information about actual and predicted classification. According to how the data points are (mis)classified, the observations in a confusion matrix can be of four types -

  1. True Positive (TP): Actual positive condition predicted as positive.
  2. True Negative (TN): Actual negative condition predicted as negative.
  3. False Positive (FP) (Type I Error): Actual negative condition predicted as positive.
  4. False Negative (FN) (Type II Error): Actual Positive condition predicted as negative.

  Since we have a multiclass framework, all of the quantities mentioned above have to be calculated separately for each individual class.
  In our context, a 'positive' would refer to the correct prediction of benign traffic as 'benign' or, the correct prediction of each individual attack type.

  The confusion matrix shows the ways in which our classification model is 'confused' when it makes predictions. Using a confusion matrix is an obvious and easy performance metric for model comparison. We can use the information in the confusion matrix to calculate certain statistics like accuracy, sensitivity, specificity etc. Those statistics provide us with a direct insight into the performance of our model.

  Thus, we have chosen to use a confusion matrix as one of our performance tools.

- Sensitivity - Since we have a multiclass framework, sensitivity is calculated separately for each individual class. Sensitivity for each class is calculated as TP/(TP+FN), where each of these quantities have been explained above.
  For a model like ours, which detects malicious traffic, sensitivity is an important metric because it tells us the proportion of malicious traffic that we are able to detect successfully. A high value would indicate that our model is quite useful and conversely a comparatively low value would indicate that the model isn't up to the mark.

- Specificity - Like the previous metrics, specificity is calculated separately for each class in our model. It is represented as TN/(TN+FP).
  Analytically, specificity values for each class would tell us the proportion of that class which has been correctly classified by the model. This is an important metric for our model since it gives us an idea of which attack is being (mostly) correctly classified, and which is not.

Note : We have used 'Accuracy' only while training the model. We have avoided it while gauging the model's predictive power because our data set is not balanced, i.e, the amount of traffic corresponding to each class is not approximately same. This would result in the True Positive (TP) terms of each class to dominate, thus biasing the result.

In our case of detecting malicious traffic, it is especially concerning because malicious traffic within networks should be rare, and so this measure may cause us to optimise away the prediction of malicious traffic.

7 Results

The results of each model variant are summarised below. In the heatmap we present the vertical labels stand for the real type of the traffic and the vertical labels correspond to the predicted type of traffic.

7.1 General Model

In the general model, we are attempting to classify 5 types of malicious traffic along with benign traffic. We generate a heatmap from the confusion matrix, and also obtain values for sensitivity and specificity for each individual class. These metrics are given below -

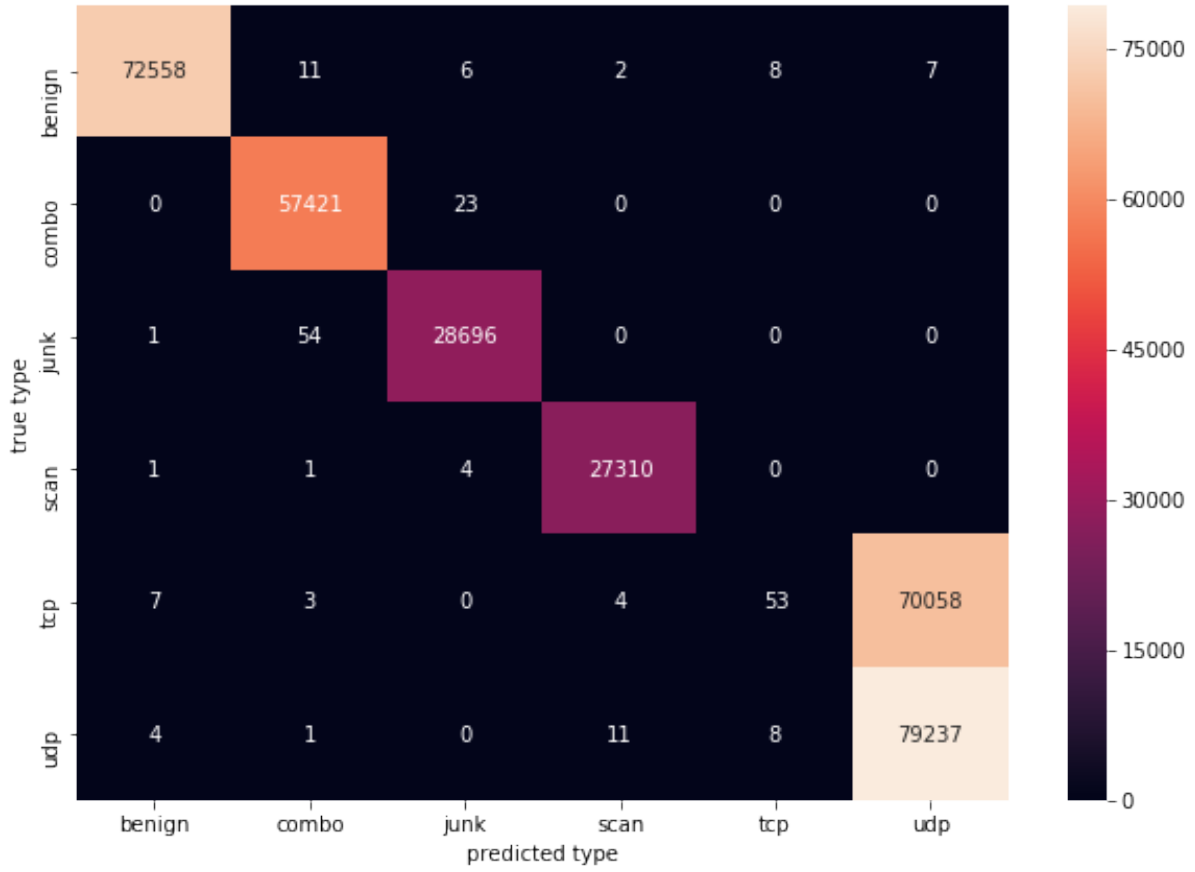

Figure 4: Heatmap for 6 types of traffic

Traffic	Sensitivity	Specificity
Benign	0.99	0.99
Combo	0.99	0.99
Junk	0.99	0.99
Scan	0.99	0.99
TCP	0.0007	0.99
UDP	0.99	0.726

Table 1: Table of Sensitivity and Specificity for Attack Labels

From the heatmap, we observe that a majority of TCP have been misclassified as UDP. This is also reflected in the Sensitivity values where the sensitivity of TCP is nearly 0.
We also note that the other types of traffic are classified quite well.
In order to investigate why TCP was being misclassified to such a great extent, we next performed a binary classification of only TCP and UDP. A snapshot of the results, as obtained from BlueCrystal, is provided below -

```
Binary classification for tcp and udp:

col_0  Residual
row_0
0.0       79083
1.0       69965
```

Figure 5: Binary Classification Table

From this, we observe that 79083 observations were classified correctly, while 69965 were misclassified. Thus, only about 50% observations were correctly classified, which is the sort of precision one would expect from randomly assigning labels to the observations.

Next, we considered the hypothesis that the data for the two attacks were so similar, that they were consistently mistaken for each other. To test this mathematically, we needed a non-parametric test (since the distributions of the two sets of data were unknown). The Kruskal-Wallis Test for comparing distributions was the test of our choice. The null hypothesis for this test is that the data originate from the same distribution. After applying this test, we got a list of p values. We applied the Bonferroni correction to this list and obtained the following result from BlueCrystal -

```
The percentage of rejecting H_0 is :
0.02924740874267688
```

Figure 6: Bonferroni Correction Results

Thus, only 0.02% of the tests (or equivalently, the pvalues) were rejecting the null.
Hence, we merged the two sets of data.

After combining TCP and UDP, we again carried out the same analysis. The only difference was that now, we had 5 labels (classes) instead of six. The confusion matrix and table of sensitivities and specificities obtained this time are given below -

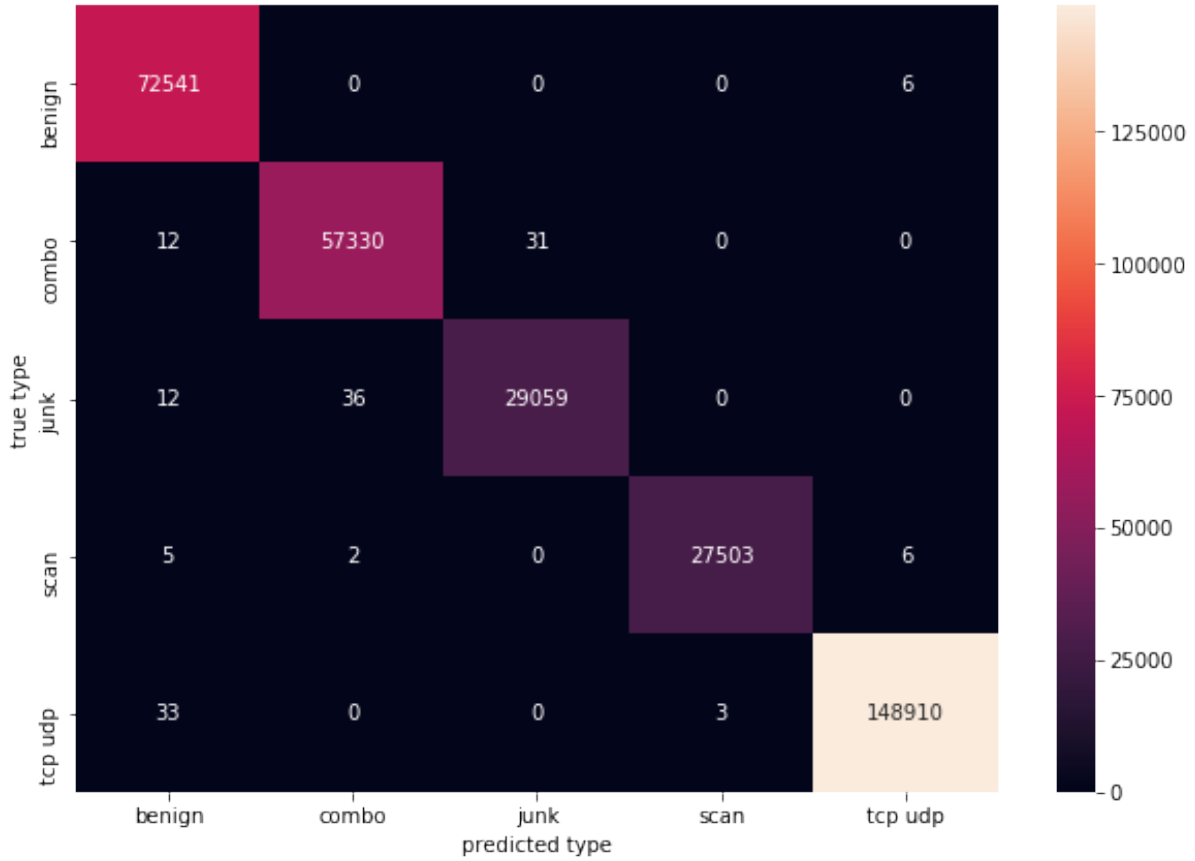

Figure 7: Heatmap for 5 types of Traffic

Traffic	Sensitivity	Specificity
Benign	0.99	0.99
Combo	0.99	0.99
Junk	0.99	0.99
Scan	0.99	0.99
TCP UDP	0.99	0.99

Table 2: New Table of Sensitivity and Specificity

We observe from the heatmap that the model has done a good job of classifying the different types of traffic in this case. This is also supported by the sensitivity and specificity values for each class, which indicate that the model has been able to detect malicious traffic successfully in 99% of the cases in each class.

7.2 Variant 1 : Unseen Attack

The purpose of this model is to determine that the model is able to recognise an unknown attack which it has not seen before. First, we hide "scan" from our training set. Then, we build the model as previous one. There is one thing that we need to notice, where the dimension of output layer is 4 since we do not have "scan" in our training data. Then, we predict on our test data. The method of obtaining the results is a bit different from the previous model. Here, we use the concept of a threshold (a value defined by us). If the probabilities of belonging to each class (equivalently, the prediction values) are all smaller than the threshold (set by us as 0.5), we consider this test case to be an outlier (our unknown class in this case), otherwise, the class with probability > 0.5 is the predicted class [12]. The results obtained are given below -

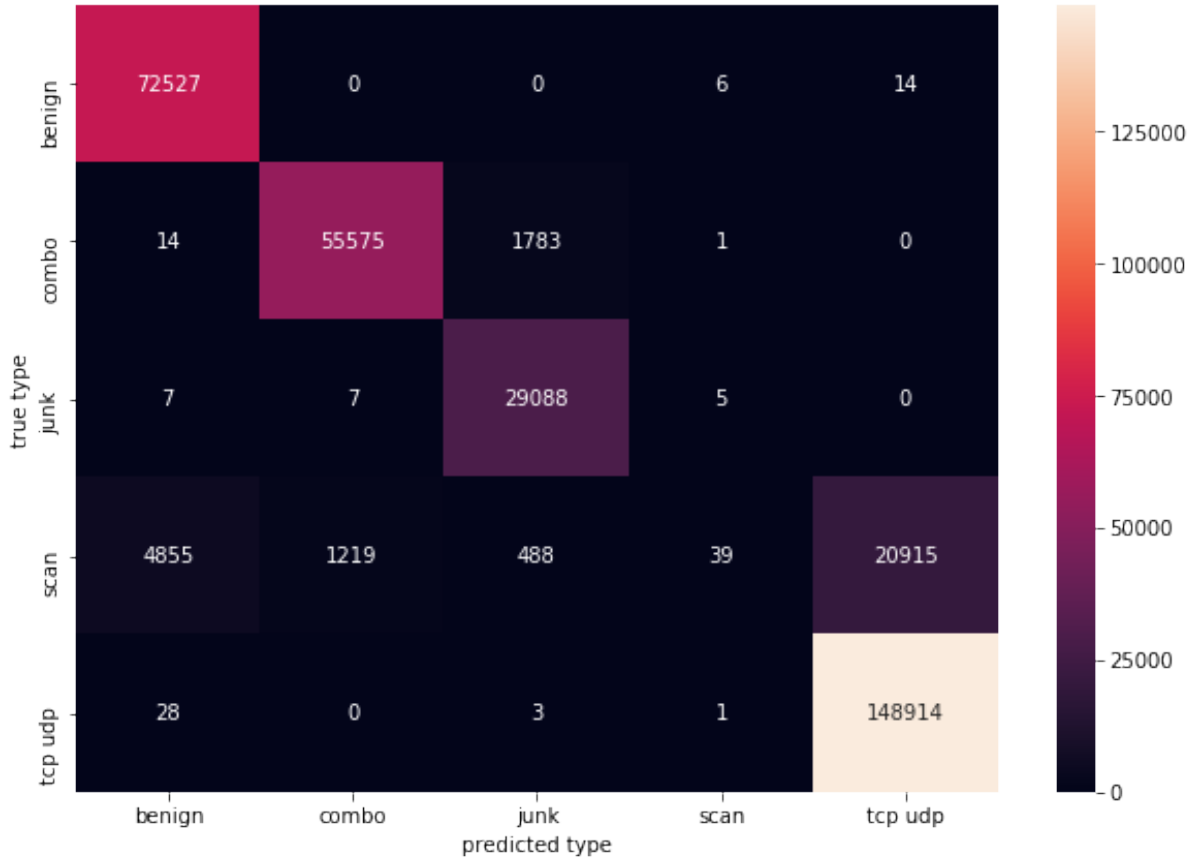

Figure 8: Heatmap when Scan is unseen

Traffic	Sensitivity	Specificity
Benign	0.99	0.98
Combo	0.96	0.99
Junk	0.99	0.99
Scan	0.0	0.99
TCP UDP	0.99	0.89

Table 3: Table of Sensitivity and Specificity when Scan is unseen

From the visualisations, we observe that Scan is mostly being misclassified as TCP UDP, while a fair proportion is also being classified as Benign traffic. This is also supported by the values from the table of Sensitivity and Specificity.
Our model is thus unable to detect Scan with a high degree of precision. However, it seems that it does recognise the traffic from Scan to be malicious traffic of some sort because it mostly puts it under TCP UDP.
The more concerning part of the results is the fact that around 18% of Scan traffic is being flagged as Benign by our model. This is definitely not ideal, and is a drawback of our model.

Next, we try the same thing on the "Combo" attack. This time, the results are as follows -

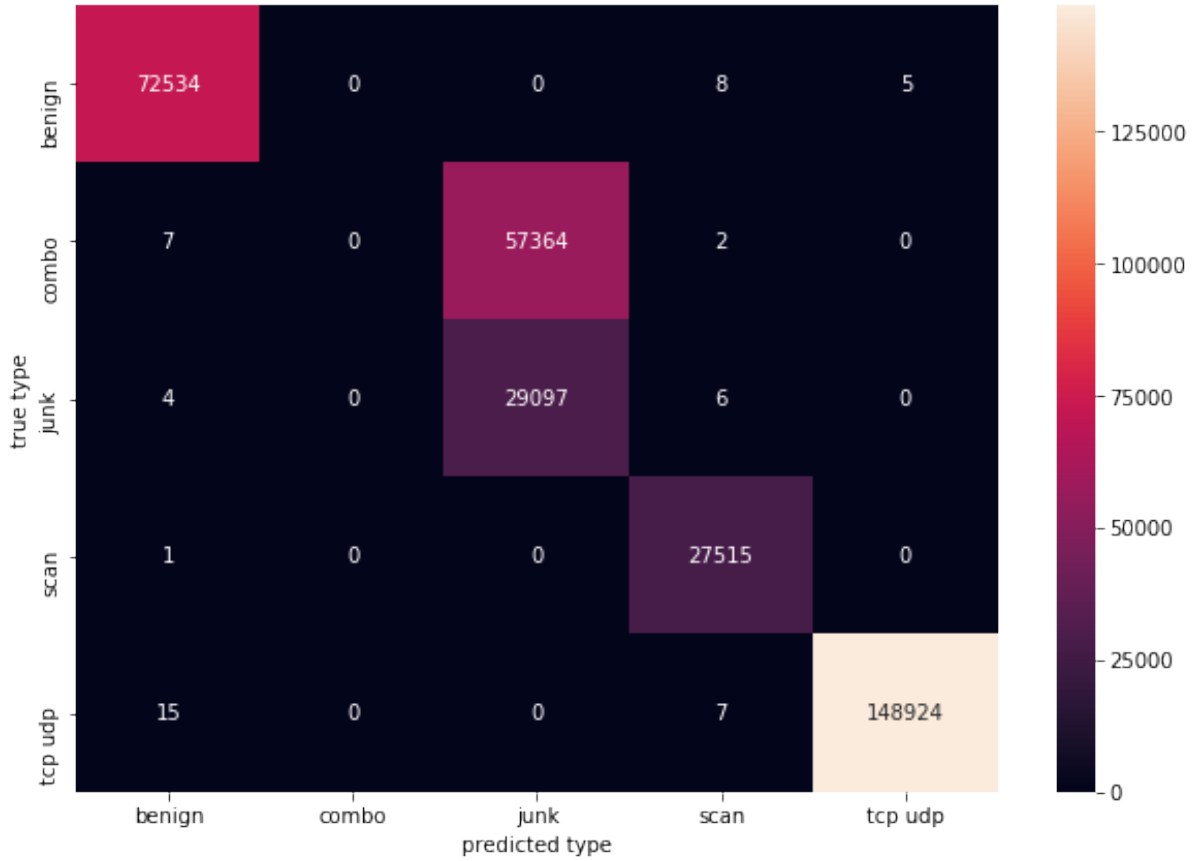

Figure 9: Heatmap when Combo is unseen

Traffic	Sensitivity	Specificity
Benign	0.99	0.99
Combo	0.0	1.0
Junk	0.99	0.81
Scan	0.99	0.99
TCP UDP	0.99	0.99

Table 4: Table of Sensitivity and Specificity when Combo is unseen

7.3 Variant 2 : Unseen Device

In this model, instead of keeping a particular traffic type unseen, we keep all observations corresponding to a certain device unseen. However, before that, we run a general neural network model on the entire data to see how well our model predicts device labels. Those results are tabulated below -

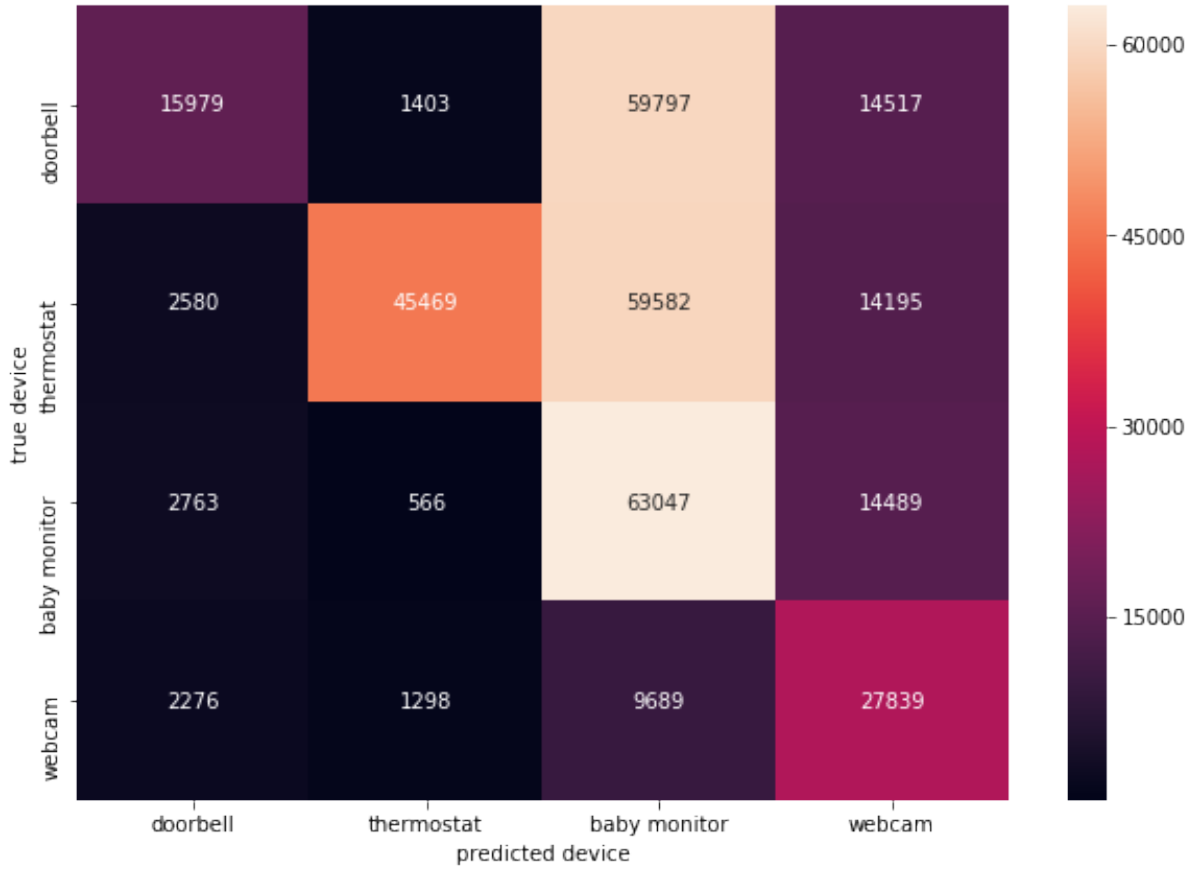

Figure 10: Heatmap of device labels

Device	Sensitivity	Specificity
Doorbell	0.17	0.96
Thermostat	0.37	0.98
Baby Monitor	0.77	0.49
Webcam	0.67	0.85

Table 5: Table of Sensitivity and Specificity for Device Labels

From the results, we observe that the model does not do a very good job of classifying the data according to the devices. The proportion of data classified correctly ranges from 17% to 77%, which is not a sufficient range of precision, by any means. In fact, it is significantly poorer than the results we got when predicting attack labels.
One of the reasons for this could be that the data differs significantly, when they are clustered according to attacks, but when they are clustered according to devices, maybe they exhibit similar trends on a macroscopic level. This makes it harder for the model to classify them correctly according to devices.

Next, we remove the data corresponding to 'Doorbell' from our training data, and perform the same analysis as before. The results are as follows -

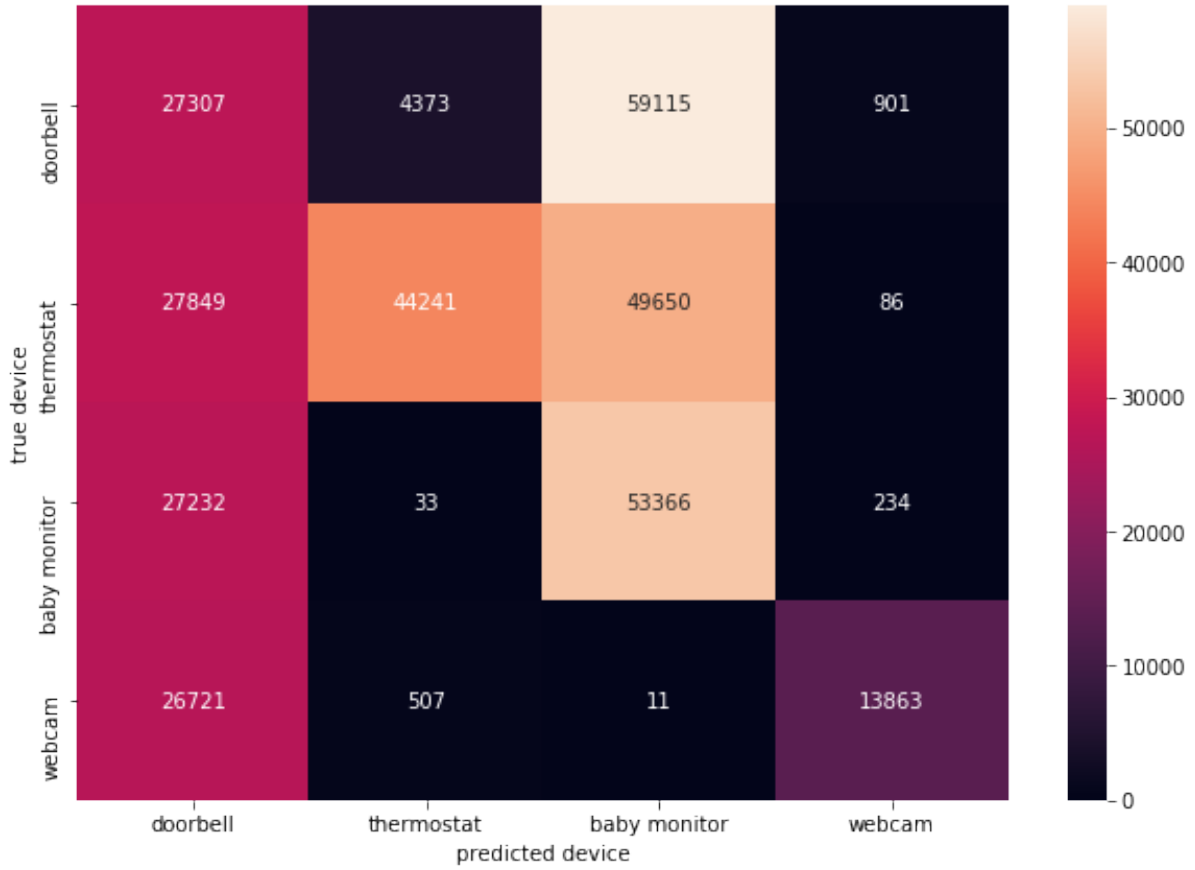

Figure 11: Heatmap of device labels when doorbell is unseen

Device	Sensitivity	Specificity
Doorbell	0.29	0.66
Thermostat	0.36	0.97
Baby Monitor	0.65	0.57
Webcam	0.33	0.99

Table 6: Table of Sensitivity and Specificity for Device Labels when Doorbell is unseen

From the results we observe that the model does a very poor job of classification, when Doorbell is unseen. This is also supported by the sensitivity and specificity figures. For the unseen device, the model is able to detect only 29% of its traffic accurately, and also misclassifies other devices as the unseen device. This is a pretty bad figure when compared to the previous model (where it was trained on all types of data), but a bit of perspective shows us that even without training, it still manages to correctly flag the unseen data 30% of the time. So, if this value is the baseline for the model, it can be expected that it would improve a great deal when it has been trained on all the types of data it is expected to detect.

8 Conclusion

The three models that we wrote all performed to various degrees of precision. One of the early hurdles we had to overcome was the mixing up of TCP and UDP data. After overcoming that by merging, the general neural network model gave very good results, which was expected from a deep learning algorithm.

When we left an unseen attack in the test data, the model could not quite replicate its earlier precision. In fact, it fell well short, and was mostly unable to classify Scan and Combo correctly.

However, it did classify both of them mostly as 'attacks', which shows that based on the limited data it was trained on, it had learned some macro characteristics of the data (i.e, attack or benign), but failed to learn the more subtle differences.

The same reasoning could be applied for the model where we left an unseen device in the data. Here too, the model had very poor precision values, but it is worth noting that even without any training, it managed to correctly classify the unseen device 30% of the time. This might not be a bad initial value of precision to begin with.

References

[1] Jason Brownlee. One hot encoding. https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/, 2017.

[2] itdxer. Batch sizes. https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network/, 2015.

[3] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[4] Reuel O Launey, Peter A Grendler, Donald L Packham, James M Battaglia, and Howard E Levine. Expandable home automation system, February 4 1992. US Patent 5,086,385.

[5] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[6] Yair Meidan, Michael Bohadana, Yael Mathov, Yisroel Mirsky, Asaf Shabtai, Dominik Breitenbacher, and Yuval Elovici. N-baiot—network-based detection of iot botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, 17(3):12–22, 2018.

[7] Stacey Ronaghan. Loss functions. https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8/, 2018.

[8] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[9] Suryansh S. Neural networks. https://towardsdatascience.com/nns-aynk-c34efe37f15a/, 2018.

[10] Adesh Shah. Visualizing artificial neural networks (anns) with just one line of code. https://towardsdatascience.com/visualizing-artificial-neural-networks-anns-with-just-one-line-of-code-b4233607209e?fbclid=IwAR1Ikuv_ZIQXPk1Dzu8He9YBDxtxAIVxa4ko81D-qOQ4t7dMpnm0OoUNKA0, 2018.

[11] Sagar Sharma. Activation functions. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6/, 2017.

[12] VinjaNinja. Predicting unknown classes. https://datascience.stackexchange.com/questions/37928/how-to-identify-the-unknown-class-in-machine-learning?fbclid=IwAR0DlkXFX54G00tmZtuxvmrIOBvyUzV1qeiOfaBkjvYTuVsoGPhp6iYKlbM/, 2018.

[13] Yael Mathov Yisroel Mirsky Dominik Breitenbacher Asaf Shabtai Yair Meidan, Michael Bohadana and Yuval Elovici. detection of iot botnet attacks. http://archive.ics.uci.edu/ml/datasets/detection_of_IoT_botnet_attacks_N_BaIoT/, 2018.

[14] finnw Yuriy Zaletsky. Data scaling. https://stackoverflow.com/questions/4674623/why-do-we-have-to-normalize-the-input-for-an-artificial-neural-network/, 2015.