**Modelling:**

**Overview of the Dataset and Use Case:**

The CICIDS 2017 dataset is a comprehensive dataset designed for network intrusion detection. It includes both benign and malicious network traffic, reflecting real-world scenarios. The data was collected over five days in July 2017 and includes labelled flows based on timestamps, source and destination IPs, ports, protocols, and attack types. For this report, we focus on the dataset from Thursday morning, which contains various web attacks, including Brute Force, SQL Injection, and Cross-Site Scripting.

The use case involves developing a robust anomaly detection system that can accurately classify network traffic as either normal or malicious, thereby enhancing cybersecurity measures.

**Data Preprocessing Steps:**

1. Data Loading

The first step involves loading the dataset from the provided CSV files. This includes parsing the data and loading it into a pandas DataFrame for further processing.

2. Data Cleaning

This step addresses missing values, duplicate entries, and any inconsistencies in the data and in the column names. To be more specific there were 20 missing values which I removed it from the dataset, found a pair of duplicate columns, and there were trailing spaces in the column names which had to be cleaned and formatted. Data cleaning ensures that the dataset is uniform and ready for analysis.

3. Feature Selection and Engineering

Relevant features are selected based on domain knowledge and prior research. Removed a few features due to the fact that they are highly correlated as shown in a histogram. This step is crucial for reducing dimensionality and improving model performance.

4. Scaling

Scaling is performed to normalize the feature values, ensuring that all features contribute equally to the model. StandardScaler from scikit-learn wasnused to scale the features to a mean of 0 and a standard deviation of 1. In my dataset there were extremely large values and some values even being a infinite number. Hence, clipped the values first to a certain range and scaled the numerical features.

5. SMOTE (Synthetic Minority Over-sampling Technique)

SMOTE is applied to address class imbalance in the dataset. Realised that there was class imbalance in the column “Label” through plotting pie charts in data visualization section. By generating synthetic samples for the minority class, SMOTE helps the model learn better and avoid bias towards the majority class. Oversampled the minority class.

6. Label encoding was performed as well. The column ‘Label’ is the target variable and was in a categorical form. The unique values of the column was ‘BENIGN’, “Web Attack – SQL Injection”, “Web Attack-XSS”, “Web Attack – Brute Force”. Label encode such that “BENIGN” was made 0, and the rest of the attacks are considered as 1

7. Data Splitting

The dataset is split into training, validation, and test sets. This allows for proper evaluation of the model's performance and helps in tuning hyperparameters effectively.

**Justification of Model Choice**

An Autoencoder was chosen for this task due to its ability to learn a compressed representation of the input data and reconstruct it, making it well-suited for anomaly detection. Autoencoders can effectively capture the underlying structure of normal network traffic and identify deviations that indicate anomalies. The objective is to develop a model that can accurately detect anomalies in network traffic, thereby identifying potential intrusions.

**Model Architecture**

The architecture of the autoencoder model for network anomaly detection using the CICIDS 2017 dataset is designed to effectively learn compressed representations of network traffic and identify anomalies through reconstruction errors. Here is a detailed breakdown of the architecture:

*Input Layer*

The input layer is designed to receive the preprocessed network traffic data. Each input instance represents a sample of network traffic, described by a set of features. The number of input neurons corresponds to the number of features in the dataset. For example, since my dataset has 78 features, the input layer will have 78 neurons.

*Encoder*

The encoder part of the autoencoder is responsible for compressing the input data into a lower-dimensional representation. This compression helps in learning the most important features and patterns in the data.

Layer 1: This is the first and only layer in the encoder. It is a dense (fully connected) layer with 14 neurons, or a specified encoding dimension if provided. The number 14 is chosen based on prior experimentation to ensure a balance between compression and retention of important information. The activation function used is ReLU (Rectified Linear Unit), which helps in introducing non-linearity and allows the network to learn complex patterns. ReLU is preferred due to its computational efficiency and effectiveness in handling the vanishing gradient problem, which can occur with other activation functions like sigmoid or tanh.

*Decoder*

The decoder part of the autoencoder is responsible for reconstructing the input data from the compressed representation learned by the encoder.

Layer 2: This is the first and only layer in the decoder. It is a dense layer with the same number of neurons as the input dimension, ensuring that the output has the same shape as the input data. The activation function used is sigmoid, which outputs values between 0 and 1. This is suitable for reconstruction tasks as it helps in modeling the probability distribution of the input features.

*Output Layer*

The output layer produces the reconstructed input data. The reconstruction is then compared to the original input to measure how well the autoencoder has learned the data distribution. Any significant deviations between the input and output can indicate anomalies.

**Training Process**

Compilation:

The model is compiled using the Adam optimizer and mean squared error (MSE) loss function.

Adam Optimizer: Adam (Adaptive Moment Estimation) is an extension of stochastic gradient descent that adapts the learning rate for each parameter. It combines the advantages of two other popular optimization techniques: AdaGrad and RMSProp. Adam is particularly well-suited for problems with large datasets and high-dimensional parameter spaces, making it a good choice for training deep learning models.

Mean Squared Error (MSE) Loss Function: MSE is used to measure the reconstruction error between the input data and the reconstructed output. It calculates the average of the squared differences between the actual and predicted values. Minimizing MSE ensures that the autoencoder learns to reconstruct the input data as accurately

Training: The model is trained on the training set, with the validation set used for hyperparameter tuning. Grid search is employed to find the best hyperparameters, such as the number of epochs and batch size.

**Hyperparameter Tuning**:

Grid Search: Performed with parameters: epochs (50, 100) and batch\_size (16, 32). The best parameters were selected based on cross-validation results.

Hyper-parameters and Their Optimality:

1. Hyper-parameters:

* Learning Rate: Implicitly set by the Adam optimizer to adapt during training.
* Batch Size: Evaluated at 16 and 32. The optimal batch size was chosen based on the grid search results.
* Number of Epochs: Evaluated at 50 and 100. The optimal number of epochs was chosen based on the grid search results.
* Encoding Dimension: Defaulted to 14 neurons, capturing the essential features of the data.

1. Optimality:

The chosen hyper-parameters were optimized through grid search, evaluating performance on the validation set. The model with the best parameters was selected based on cross-validation results, ensuring that it generalizes well to unseen data.

**Model Evaluation:**

Reconstruction Error: The mean squared error (MSE) between the input data and its reconstruction is calculated for each sample.

Anomaly Detection: Samples with high reconstruction errors are flagged as anomalies.

Break down each evaluation metric in terms of model performance:

1. Threshold:

This value determines the cutoff point for classifying instances as anomalies or normal. It's a critical parameter as it directly affects the balance between false positives and false negatives. Adjusting the threshold can impact the model's overall performance.

1. True Positives (TP):

These are instances where the model correctly identifies anomalies as anomalies.

In my case, i correctly identified 371 anomalies out of 464, indicating that the model is effective in catching true anomalies.

1. True Negatives (TN):

These are instances where the model correctly identifies normal data as normal.

Correctly identified 30,003 out of 33,610 normal instances, showing that the model is effective in recognizing normal patterns.

1. False Positives (FP):

These are instances where the model incorrectly labels normal data as anomalies.

Have 1,680 false positives, indicating instances where normal data was mistakenly flagged as anomalous. This suggests a need to reduce false positives to improve precision.

1. False Negatives (FN):

These are instances where the model incorrectly labels anomalies as normal data.

There are 93 false negatives, meaning that the model missed detecting these anomalies. Minimizing false negatives is crucial to improve recall and ensure all anomalies are captured.

1. Total Anomalies:

This represents the total number of anomalies present in my dataset.

1. Total Normals:

This represents the total number of normal instances in my dataset.

1. Average Reconstruction Error for Normals:

This metric indicates how accurately the model reconstructs normal instances.

A lower average reconstruction error suggests that the model is better at capturing the characteristics of normal data.

1. Standard Deviation of Reconstruction Error for Normals:

This measures the variability of reconstruction errors for normal instances.

The lower standard deviation implies that the model's reconstruction errors for normal instances are more consistent.

1. Average Reconstruction Error for Anomalies:

This metric indicates how accurately the model reconstructs anomaly instances.

A higher average reconstruction error for anomalies compared to normals is expected, as anomalies represent deviations from the norm. For anomalies, the average reconstruction error was slightly higher at 0.000113, with a standard deviation of 0.000113

1. Standard Deviation of Reconstruction Error for Anomalies:

This measures the variability of reconstruction errors for anomaly instances.

A lower standard deviation suggests that the model's reconstruction errors for anomaly instances are more consistent.

1. Number of Anomalies above Threshold:

This represents the number of anomalies whose reconstruction error exceeds the chosen threshold.

It indicates how many anomalies were effectively detected based on the threshold criterion. A total of 888 anomalies had reconstruction errors above my chosen threshold.