An autoencoder is a type of neural network that is trained to encode input data into a lower-dimensional representation, and then decode that representation back into the original data. This process is known as dimensionality reduction, and it can be useful for identifying patterns in data that are not immediately apparent. In this code, the autoencoder is trained on a dataset of Industrial Control System (ICS) data. The goal is to use the autoencoder to identify anomalies in the data.

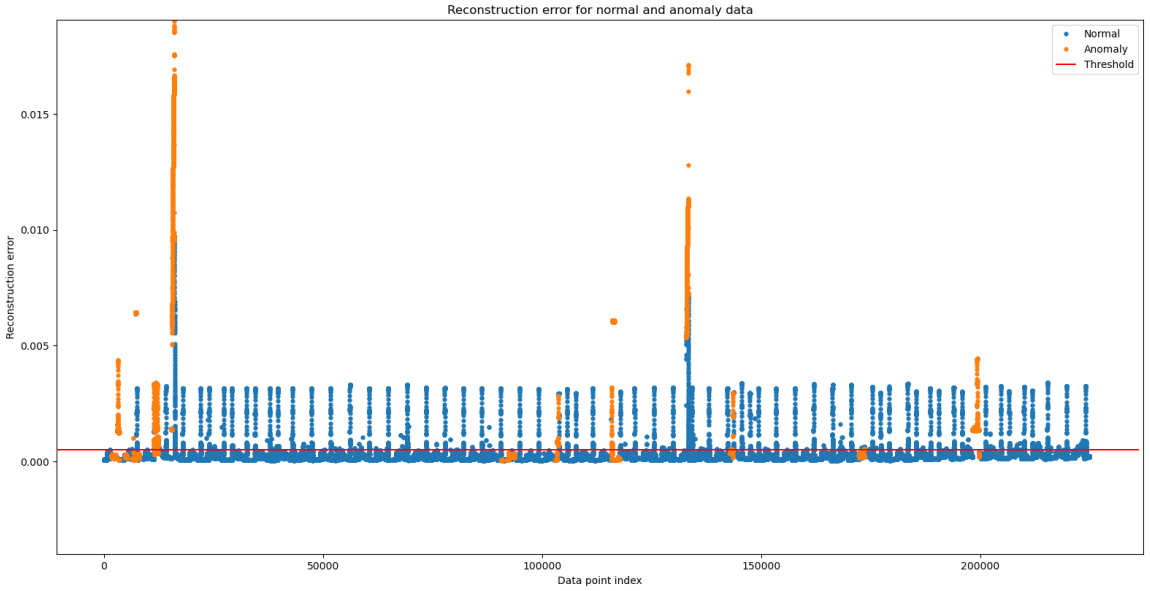
The first step is to load the training, validation, and test datasets from CSV files using Pandas library. Then, the columns in these datasets are preprocessed to prepare them for training the autoencoder. This preprocessing includes standardizing the data, which scales the numeric columns to have zero mean and unit variance. This is done using the StandardScaler class from the scikit-learn library. After preprocessing, the data is split into inputs and labels. The inputs are the actual data that will be used to train the autoencoder, and the labels are the "truth" values that the autoencoder will be compared against to measure its performance. In this case, the inputs are all of the columns in the dataset except for the "Timestamp", "Id", and "Label" columns. The labels are the "Label" column, which indicates whether each data is normal or anomaly. Next, the code defines the architecture of the autoencoder. The autoencoder is a neural network with three fully-connected (dense) layers. The first layer is the input layer, which takes the preprocessed data as input. The second layer is the encoding layer, which learns to compress the input data into a lower-dimensional representation. Finally, the third layer is the decoding layer, which learns to reconstruct the original input data from the compressed representation. After defining the architecture, the code compiles the autoencoder by specifying the optimizer and loss function to use during training. In this case, the Adam optimizer and mean squared error (MSE) loss are used. Finally, the code trains the autoencoder using the fit method, which trains the network using the training data and labels. The training process runs for a specified number of epochs (i.e. iterations over the entire dataset) and uses a specified batch size (i.e. the number of samples to use in each update to the model). The trained model is saved to a file so that it can be used later to make predictions on new data.

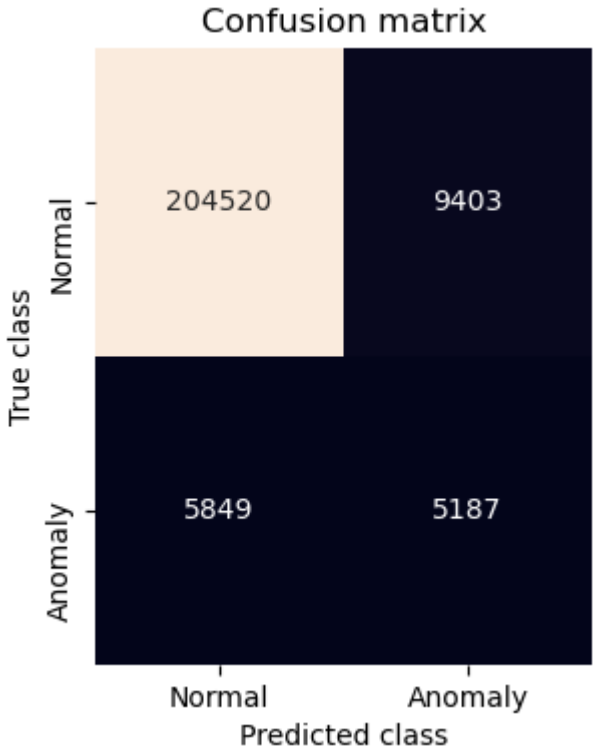
After training, the autoencoder can be used to detect anomalies in the ICS data by encoding the input data and comparing the encoded representation to the original data. If the data falls above a certain threshold (in the code this threshold is specified as threshold_fixed) then it is labeled as anomaly since its reconstruction error is higher than expected. To determine an accurate threshold, the data were first run in validation_set and visualized on a plot with colored points according to the point's label. Looking at the plot, the threshold that seperates the labels(anomaly from the non-anomaly) the best is selected and used. Performance of threshold is determined by looking at the confusion matrix. Through a set of trial and errors, it is concluded that 0.00049995 is a suited threshold to determine anomaly from normal.

It must be noted that the resource below is highly utilized to accomplish a successful completion of this assignment.

https://towardsdatascience.com/anomaly-detection-using-autoencoders-5b032178a1ea

Example plot and confusion matrix:


Reconstruction error for normal and anomaly data
Normal
Anomaly
Threshold
Reconstruction error
0.015
0.010
0.005
0.000
0
50000
100000
150000
200000
Data point index


Confusion matrix
204520
9403
5849
5187
True class
Normal
Anomaly
Normal
Anomaly
Predicted class