Sensor Deviation Detection and Web-based Visualization

By Thomas Biddlecombe

# Abstract

This report presents the work conducted to develop a model for detecting deviations in sensor readings across multiple sensors. The goal of the project was to train a model capable of identifying anomalies in sensor data, which would then be presented through a web interface accessible on the local network. The project involved acquiring data from the Control2K dataset, pre-processing the data, creating a data loader, defining a model using TensorFlow, training the model, compressing it using TensorFlow's built-in converter, and implementing the networking components. The results demonstrate the successful implementation of a model for detecting sensor deviations and a web-based visualization of the data.

# Introduction

The aim of this project was to develop a model capable of detecting deviations in sensor readings. Anomalies in sensor data can indicate potential malfunctions or abnormal behaviour in the system being monitored. The project focused on acquiring sensor data, pre-processing it, training a model to identify deviations, and implementing a web-based interface to visualize the detected anomalies.

# Data Acquisition and Pre-processing

The dataset provided by Simon Osborne from Control2K was utilized for this project. The data was distributed across three separate files, with each row containing three values: the time of reading, the sensor identifier, and the corresponding sensor value. To facilitate analysis, the data was concatenated and transformed into a single table format. Each row in the processed data represented a time step, and the sensor values were arranged as a vector of 19 sensors per time step. Additionally, the data was resampled to create a single reading every 10 seconds.

# Data Loader

To train the model, a data loader was developed. The data loader employed a sliding window approach with a window width of 50 (equivalent to 500 seconds). Random time steps were selected and normalized before being fed into the model for training. Each data point returned by the model included a random sensor with a configurable amount of noise applied to it. This noise information was propagated to the corresponding labels, enabling the model to learn to detect deviations in sensor readings.

# Model Definition

The model was defined using TensorFlow, a popular deep learning framework. A simple sequential model architecture was employed, consisting of a Recurrent Neural Network (RNN) followed by a linear layer with a SoftMax activation. The RNN was chosen for its suitability in processing time series data, while the linear layer with SoftMax activation was employed to appropriately format the model's output for the classification task at hand.

A screen shot of a computer code

Description automatically generated with low confidence

# Model Training

To train the model, the Weights&Bias platform was utilized for run reporting and analysis. One crucial aspect of training involved gradually reducing the noise applied to the randomly chosen output. This reduction in noise was only initiated once the model's accuracy surpassed a pre-defined threshold. This approach aimed to optimize the model's performance in detecting even the smallest deviations, while also preventing overfitting by challenging the model to predict the problem sensor accurately.

The benefits of this progressive noise reduction approach were twofold. Firstly, it facilitated easier initial learning, enabling the model to focus on learning the fundamental patterns and features of the data without being overwhelmed by subtle variations. Secondly, it promoted progressive complexity, gradually introducing more nuanced and subtle deviations as the model improved. This approach facilitated the model's ability to detect smaller variations over time and helped it build robust representations of the underlying patterns.

A graph of a function

Description automatically generated with low confidenceA picture containing text, line, plot, font

Description automatically generatedA picture containing text, line, diagram, screenshot

Description automatically generated

The end result was a STD of ~1.6 and comparing the distributions it can be observed that only a 30% increase in data points above the base distribution.

# Model Compression

To facilitate deployment on resource-constrained devices, the model was compressed using TensorFlow's inbuilt TF Lite converter. The conversion process transformed the model into int8 format, enabling it to be loaded into a char array for use on the device. This format ensured compatibility with various libraries that can execute the model directly from the char array, optimizing memory usage and computational efficiency.

# Networking Components

The project involved the development of three networking components: the server responsible for sending data to the Pico, the Pico client for processing the received data into a dictionary format, and the Pico hosting the web server to display the data. The web server utilized an HTML script that imported Plotly, a powerful visualization library, to generate interactive charts for visualizing the sensor data and detected anomalies.

For the web interface, each packet contains two pieces of data the data the model was fed and the models’ predictions. The data was sorted by the models’ predictions and the top 4 graphs where shown. Because the output is a SoftMax the probabilities across all 19 devices add up to 1 meaning that a value of ~0.71 see shown here, indicates that the model gives *pressure2* a ~71% chance of being the problem device. With the model having an overall accuracy of ~90% the likely hood that the problem senser is in the top 4 is 99.99%.

A picture containing text, screenshot, font, diagram

Description automatically generated

# Problems faced

During the development of the project, several challenges and problems were encountered, requiring careful consideration and problem-solving. The following paragraphs outline some of the key difficulties faced during the project:

One significant challenge was dealing with the complexity and scale of the sensor data. The dataset provided by Simon Osborne from Control2K consisted of multiple files, each containing rows with three values: the time of reading, the sensor, and its corresponding value. Combining and pre-processing this data into a single table format proved to be a non-trivial task. The varying formats, missing values, and inconsistencies in the data required extensive data cleaning and transformation efforts to ensure its compatibility with the model. Additionally, the large volume of data and the need to process it in real-time posed computational and memory constraints that needed to be addressed effectively.

Another challenge involved optimizing the model's accuracy while avoiding overfitting. Detecting the smallest deviations in sensor readings required training the model to be highly sensitive to subtle variations, but this also increased the risk of overfitting to the training data. Finding the right balance between sensitivity and generalization proved to be a delicate task. Iterative experimentation and validation techniques were employed to fine-tune the model and adjust hyperparameters effectively. Techniques such as reducing noise levels gradually and incorporating early stopping criteria helped strike a balance between high accuracy and avoiding overfitting.

A significant hurdle during the project was the deployment and integration of the model on resource-constrained devices. Converting the model into an efficient format and deploying it on the Pico device required dealing with memory limitations and optimizing execution speed. The utilization of TensorFlow's TF Lite converter to compress and convert the model into int8 format was instrumental in overcoming these challenges. However, further optimizations, such as model quantization and pruning, were necessary to ensure optimal performance while minimizing memory usage. Integrating the model with the Pico's networking components and web server functionality also demanded meticulous configuration and debugging to ensure seamless communication and accurate data presentation.

# Conclusion

In this project, a model for detecting deviations in sensor readings was successfully developed and implemented. The model, trained using TensorFlow, demonstrated the ability to identify anomalies in sensor data. The project also included the creation of a web-based interface that allowed users to access and visualize the detected anomalies. Future work could focus on further refining the model's accuracy and exploring additional techniques for optimizing its performance in detecting deviations in sensor data.

# Future work

In the future, the project that was described could be further developed and enhanced in several ways:

Enhanced Model Architecture: The current model architecture, which consists of a simple sequential model with an RNN and a linear layer, can be further explored and optimized. Researchers could experiment with more complex architectures, such as recurrent neural networks (RNNs) with attention mechanisms or convolutional neural networks (CNNs) for capturing temporal and spatial patterns in the sensor data. Additionally, exploring advanced techniques like deep learning architectures such as transformers could potentially improve the model's performance in detecting deviations.

Data Augmentation Techniques: To improve the model's robustness and generalization, data augmentation techniques can be employed. These techniques involve artificially generating additional training examples by applying transformations or perturbations to the existing data. For sensor data, this could include introducing random noise, time shifts, scaling, or other transformations that mimic real-world variations. By augmenting the dataset, the model can learn to handle a wider range of scenarios and improve its accuracy in detecting deviations.

Real-Time Monitoring and Alerting: The current implementation focuses on training the model and visualizing the detected anomalies through a web interface. However, for practical applications, the system can be extended to real-time monitoring and alerting. This would involve integrating the trained model into a live system that continuously receives sensor data, applies the model for anomaly detection, and triggers alerts or notifications when deviations are detected. This would enable timely responses and proactive maintenance in scenarios where immediate actions are required.

Online Learning and Adaptation: In dynamic environments where the sensor data distribution or behaviour may change over time, incorporating online learning and adaptation techniques can be valuable. Online learning allows the model to adapt and update itself continuously as new data becomes available. This could involve techniques such as incremental learning or concept drift detection, where the model can detect and adjust to changes in the underlying patterns of the sensor data without requiring retraining from scratch.

Integration with Advanced Analytics: The project can be integrated with advanced analytics techniques to provide deeper insights and analysis. For example, statistical process control methods, time series forecasting, or anomaly explanation techniques can be incorporated to provide more comprehensive analysis and understanding of the detected anomalies. This would enable users to gain deeper insights into the system's behaviour and make informed decisions based on the detected deviations.

Overall, the future development of the project would involve a combination of model improvements, data augmentation, real-time monitoring, adaptation techniques, efficient deployment on edge devices, and integration with advanced analytics to enhance the accuracy, efficiency, and applicability of the anomaly detection system in various domains.