Expertise
and insight
for the future

Mark-Felix Müller

# Condition Monitoring of Test Equipment Using Autoencoders

Metropolia University of Applied Sciences

Bachelor of Engineering

Degree Program in Electronics

Bachelor's Thesis

26 November 2020

| | |
|---|---|
| Author | Mark-Felix Müller |
| Title | Condition Monitoring of Test Equipment Using Autoencoders |
| Number of Pages | 40 pages |
| Date | 26 November 2020 |
| Degree | Bachelor of Engineering |
| Degree Program | Degree Program in Electronics |
| Professional Major | Electronics |
| Instructors | Hannu K. Seppänen, Project Manager<br>Heikki Valmu, Principal Lecturer |

This project aimed to research and develop a condition monitoring solution for test equipment of a robotic manufacturing line. Measurement data collected by the test equipment is used to train an artificial neural network to assess the condition of its future measurement performance.

The project involved investigating the measurement data to derive an understanding of the performance of the test equipment. This investigation led to the development of deep learning software used to power a visual display of the results.

Characteristics in the data indicating condition were discovered, which led to the development of an autoencoder used to process measurement data. The processed data is manipulated to determine the condition of the test equipment. The condition of the test equipment is displayed in a graphical view for monitoring.

The result is a monitoring solution which provides timely insight on the tester data that may be used to perform maintenance on the equipment. The project, its research, and its monitoring solution contribute to the development of a predictive maintenance solution.

| | |
|---|---|
| Keywords | Autoencoder, Condition Monitoring, Test Equipment |

## Contents

## List of Abbreviations

| | |
|---|---|
| CR | Count ratio. A value derived from feature engineering during the data preparation stage. |
| *dPxT* | *Product* feature. An alternate name for the feature resulting from the multiplication of the *Leak* and *Test Time* features. |
| GEHC | GE Healthcare, a medical technology company of the American General Electric Conglomerate. GEHC has a headquarters in Helsinki, Finland. |
| LCL | Lower control limit. The lower bound of controlled behavior for a given distribution. |
| LSTM | Long-short term memory. A neural network architecture which allows for patterns over time in the data to be learned. |
| MSE | Mean squared error. A mathematical loss function estimator which returns an average of squares of errors. |
| PR | Product ratio. A value derived from feature engineering during the data preparation stage. |
| ReLU | Rectified linear unit. An activation function that outputs its input if its value is positive. Otherwise, it outputs zero if the input's value is zero or a negative value. |
| RUL | Remaining useful life. An estimation of the time a device or component continues to operate as intended. |
| UCL | Upper control limit. The upper bound of controlled behavior for a given distribution. |

# 1. Introduction

This research aims to identify whether measurement data can be leveraged to assess the condition of the measurement device itself. More specifically, the measurements performed by test equipment are transformed and applied to monitor the equipment's condition. This is achieved by investigating the characteristics of the data, training a deep neural network, and applying domain expertise to attain meaningful insight on the operating condition of the measuring device.

This research was performed in partnership with GE Healthcare (GEHC) and is intended to aid its production function in monitoring valuable manufacturing machinery. By leveraging quality control data that is already being collected, the research hopes to innovate and provide an alternative to investing resources that would explicitly monitor the condition of the machinery.

GE Healthcare, well-known in the healthcare industry for its patient monitoring systems, develops various healthcare technologies and medical equipment. The healthcare industry demands a strict standard of quality for medical equipment. Such equipment requires an appropriate manufacturing process with emphasis on quality control.

A successful manufacturing process outputs a consistent and controlled product. Maintenance of manufacturing machinery is vital to the success of the manufacturing process. However, maintenance of such machinery is often costly and nontrivial. Furthermore, maintenance is only valuable when performed if the condition of the machinery deteriorates. Thus, the condition of the machinery is valuable information.

The research could lead to the development of smart monitoring and predictive maintenance solutions for systems that do not support direct monitoring of the measurement device. A smart solution in this case is one that utilizes machine learning and does not rely on explicitly programmed rules.

Areas of this research are confidential as they are the property of GEHC. Specific instances of data, code, and methods are omitted. This report's focus is on the design and application of the deep neural network. The report attempts to provide the key insights and tools used to guide the development.

# 2. Theoretical Background

## 2.1. Site Information

GE Healthcare has a headquarters in Helsinki, Finland where various healthcare technology is researched, designed, tested, and manufactured. Development and manufacturing focus on medical imaging and patient monitoring hardware and software.

This report discusses processes involved in the manufacturing of a product named D-fend Pro +, a water trap. Water traps are used to capture liquid water into a clear container and are installed in a particular module of a patient monitoring system. Patient monitoring systems are used in healthcare facilities to provide physiological information about a patient to aid their treatment. These systems consist of modules designed for specific applications. The respiratory module, intended for monitoring respiratory parameters of patients in an operating room or intensive care unit, requires a disposable water trap produced by GEHC. The water trap, seen in figure 1, consists of a filtering system and a container.

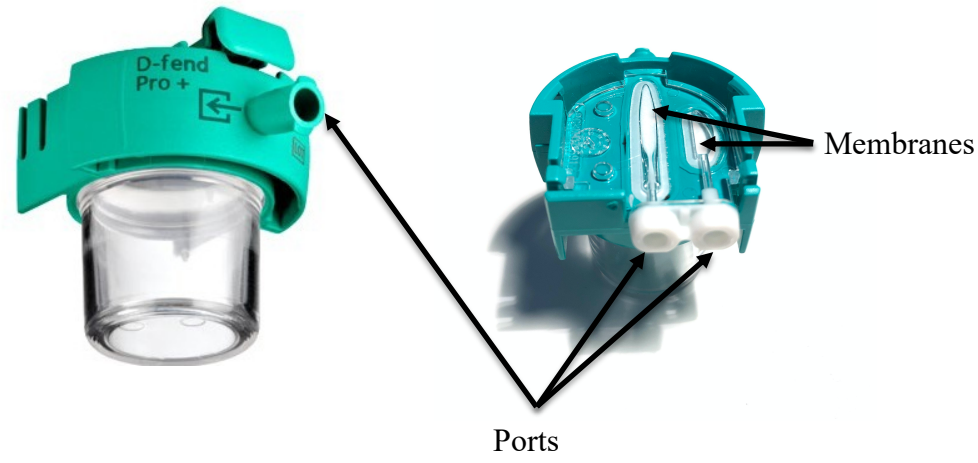


Figure 1. The D-fend Pro + water trap as seen from opposite angles [1]. Ports and membranes are relevant components of the collected data.

The D-fend Pro + water trap has a semipermeable membrane made of polytetrafluoroethylene which protects the respiratory module's gas measurement system from water, dust, and bacterial contaminants that may interfere with measurement accuracy. Gas samples may pass through the membrane to the gas analysis unit, while condensed water is trapped in the clear container. The container may be emptied during the water trap's operating life. [1.]

The water traps are manufactured from unassembled components by a line of industrial robots. These robots are modular and perform varying tasks on individual water traps including heating, moving, assembling, and testing. The manufacturing line is predominantly automated, requiring a technician for troubleshooting and supplying batches of unassembled components. The line may operate continuously for hours and has regular periods of inactivity depending on production needs. This research focuses on quality control data collected by one test point, namely the leak test.

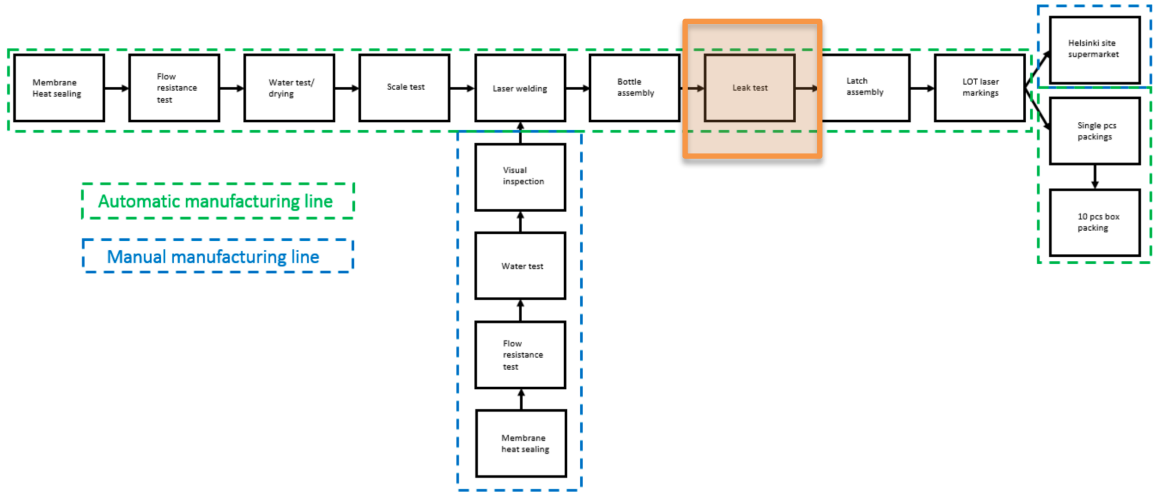


Figure 2. The D-fend manufacturing line block diagram.

The block diagram in figure 2 represents the major functions of the d-fend manufacturing line. The leak test module, the point of focus for this report, is highlighted in orange. In reality, the various blocks of figure 2 are housed in enclosures as a series of assembly and testing points seen in figure 3.

Figure 3. The D-fend manufacturing line. Each enclosure may contain up to several robots and modules seen in the block diagram.

The leak test is one of the final testing stages in the manufacturing line and is situated in an enclosure with a robot arm seen in figure 4. Units passing this stage are prepared and packaged for distribution.

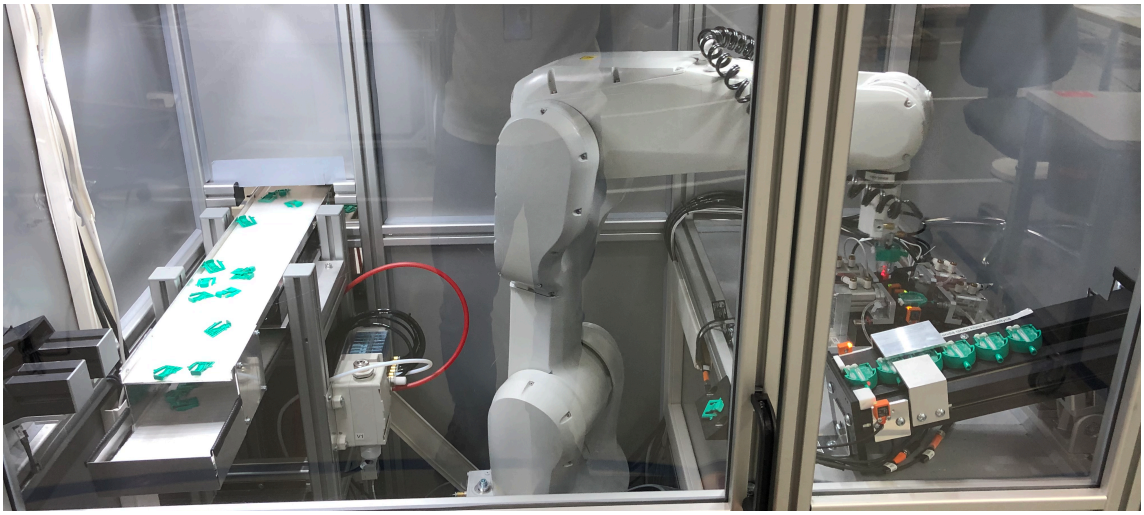

Figure 4. An enclosure housing a robot arm and various modules. The robot arm is loading a water trap unit for testing into the leak tester.

The testers of the manufacturing line are controlled automatically by custom tester software. The task of each tester is to ensure every water trap unit passing through the manufacturing line is in compliance with quality control thresholds. The software instructs the manufacturing line robots to remove water trap units that fail any test. The manufacturing line is designed to operate efficiently, allotting a predetermined maximum duration for each tester to perform its operation.

The tester software logs the measurements performed by each tester in a database. As a water trap unit encounters various testers throughout the line, various parameters are measured and logged. However, individual water trap units are untraced and the measurements of one tester are unlinked to those of another tester.

The leak test involves applying a vacuum pressure to the ports of a water trap unit to determine the leakage of atmosphere via the membranes. Sensors read the leakage value continuously until either the measured value complies to a predetermined threshold or the maximum test duration is exceeded. This is to accommodate a stabilization period of the vacuum pressure when a unit is situated in the leak tester.

The leak tester has two identical unit slots to allow for simultaneous testing during periods of high traffic in the line. Each slot is an individual leak tester with separate pressure

vacuum and sensor. Up to one water trap unit may be situated into each slot for testing. The tester software prioritizes one slot, setting the other as an auxiliary. During normal operation and depending on the line's traffic, the frequency of use of the auxiliary slot is up to, but not exceeding, that of the primary slot.

## 2.2. Problem Scope

The research was conducted by request of GEHC as a means to develop a predictive maintenance solution for the D-fend manufacturing line. This solution should involve a condition monitoring system with visual feedback to notify manufacturing technicians of potential deterioration in the performance or condition of the manufacturing line. Put bluntly, a traffic light should describe the current condition of the line, with yellow or red lights indicating alarming performance. When alarmed, a technician would then be able to investigate and maintain the line to prevent or minimize a halt in production.

My previous assignment with GEHC involved developing statistical visualizations for the D-fend line's tester data. These visualizations are tools aimed to assist a technician in diagnosing an issue related to the line's flow resistance or leak tester stages. The goal was to combine the traffic light system and diagnosing tools into a dashboard viewable in the production workspace, named D-fend Dashboard. This dashboard and research contribute to an ongoing project within GEHC named Testerwatch, whose development focuses on performance and condition monitoring of manufacturing machinery and test systems.

A halt in the D-fend manufacturing line results in costly consequences, scaling in magnitude to its duration. These consequences may cascade throughout several functions of GEHC. There are various events that may result in a halt including mechanical jams or equipment failure. Events that lead to halts often have unique resolutions, requiring the domain expertise of a manufacturing technician. Although many halts are caused by unpredictable events, some events may exhibit evidence in the tester data being collected during manufacturing. The more knowledge a technician has about a problem within the line, the more efficiently it can be resolved. The D-fend dashboard aims to present relevant information of the accessible tester data to enable a quick response to an issue. The D-fend dashboard is scalable by design, allowing further tools and functionality to be added to it throughout its development.

The assignment had preconditions resulting in particular caveats, namely the limited usable data. These constraints were present due to stringent validation and verification protocols set in place relating to the logistics of the manufacturing line. Therefore, the introduction of external monitoring equipment for condition monitoring of the D-fend line was prohibited.

Ideally, the device targeted for monitoring should have relevant data collected about it. For example, monitoring accelerometer data from a sensor mounted on the exterior of a wind turbine engine may provide insight on its operating condition. Vibrations detected by the sensor may be used to construct a representation of typical performance during the turbine's intended operating condition. However, due to the research's preconditions, only measurement data collected by two testers is available for use. The flow resistance and leak tester measurement data directly reflect the parameters of the water trap unit under test, not the tester itself. Considering the prior example, this would be the equivalent of assessing the turbine's condition by monitoring the aerodynamic performance of a unit under test in the wind tunnel. This data constraint is the crux of the research, demanding innovation to diagnose the operating condition of the line using only tester measurement data.

My previous work allowed me to become familiar with the flow and leak tester data by performing time-series analyses and correlative investigation. This work indicated that there was no reliable correlation between the values of either test with respect to the operating condition of the manufacturing line. In the interest of development progress, I decided to focus solely on the leak tester data. The leak tester has unique characteristics I intended to take advantage of to assess its condition. The fact that the leak tester has dual slots performing separate measurements allows me to compare their performances against each other. As both slots are designed to operate equally, their measurements performed should exhibit similar behavior over time. This presumes it is very unlikely both slots encounter the same issue simultaneously, or that an issue affects both slots similarly.

## 2.3. Predictive Maintenance

Predictive maintenance is a technique performed on machinery to reduce the likelihood of malfunction by preemptively maintaining faulty components. Predictive maintenance solutions are often research intensive and require engineers and data scientists to implement. However, incentives including consistent production yields and reduced maintenance costs make these solutions' development a worthwhile investment. Predictive maintenance aims to reduce the cost and frequency of maintenance requirements by preventing unplanned reactive maintenance and guiding preventative maintenance [2].

There are several variants of typical predictive maintenance solutions including remaining useful life (RUL) prediction and anomaly detection. Given the data constraints of this research, RUL prediction is not a candidate as it requires prior failure data and associated maintenance logs. The exclusion of RUL prediction impairs the predictive power of any maintenance solution as scheduling a window for maintenance is unreliable. However, detecting irregular behavior prior to a halt in the manufacturing line could allow technicians to become aware of the situation preemptively. Coupled with diagnosing tools, reactive maintenance or the halt itself could potentially be avoided altogether.

Performing maintenance to alleviate an issue in the manufacturing line requires the ability to investigate an issue's source. Knowledge about the issue improves the investigation's efficiency. Evidence in the data relating to an issue is often crucial in planning a maintenance procedure. The manufacturing line consists of many enclosures, robots, and various modules resulting in numerous potential locations of disruption. Maintenance is a targeted operation which a technician can perform more efficiently with diagnosis tools. Diagnosis tools attempt to aid a user answer the following questions pertaining to an investigation:

1. What is the issue?
2. When does it occur?
3. Where is its source?

The D-fend dashboard is designed to include diagnosis tools to corroborate the results of an alert system. Together, an alert system and corroborative tool form a condition monitoring solution.

## 2.4. Anomaly Detection

The premise of anomaly detection is understanding normal operation in order to detect irregularities. Data collected during normal operation can be constructed into a representation of such a condition. When comparing data collected during irregular operation to the normal representation, distinctions indicate anomalies. Consider the flow resistance data represented in figure 5.

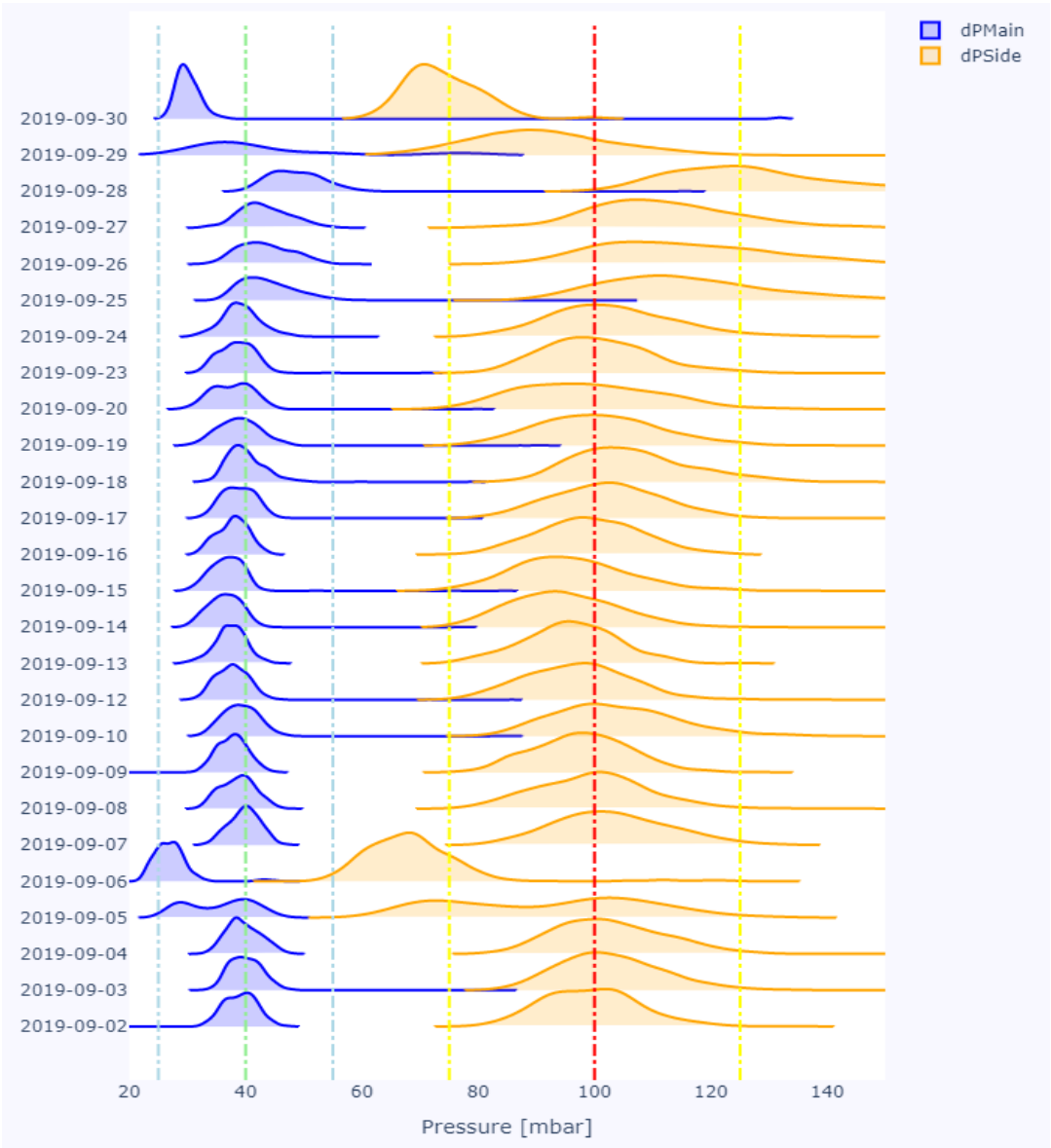


Figure 5. A ridge plot exhibiting a month of flow resistance tester measurement data.

The ridge plot in figure 5 displays the probability distributions of all measurements collected by the tester each day for a month. The main membrane (blue) and side membrane (orange) pressure readings are adjacent to each other, with one distribution of each displayed per day. Vertical dashes extending from the x-axis indicate respective

lower and upper specification limits, whereas the green and red dashed lines indicate the target value for the main and side membranes respectively. The target value is the expected measurement value for which the membranes are designed. Water traps whose pressure readings exceed the specification limits have failed the flow resistance test and are removed from the manufacturing line.

Glancing at the ridge plot from bottom to top, the peaks of the blue and orange distributions seem to tend to the target value dashed line. However, the distributions for September 6th and 30th are particularly irregular in comparison to the other days of the month. Such anomalies may be an indicator of an issue with the tester or a batch of water trap units. When an anomaly is observed in a timely manner, investigation is warranted to assess the need for maintenance.

## 2.5. Machine Learning

Machine learning is a subfield of artificial intelligence in which computer programs with the ability to learn are developed. A program is said to learn if it automatically improves its performance at a specific task through experience, paraphrasing Thomas M. Mitchell [3]. Generally, such programs incorporate self-modifying algorithms that adjust their parameters when experience in the form of data is presented. Such algorithms pass data through their operations, updating a tensor, a multidimensional array, of parameters that represent patterns in the data. Ultimately, an input can be fed to a machine learning algorithm resulting in an output resembling the learned patterns of that data.

### 2.5.1. Unsupervised Deep Learning

Deep learning is a subfield of machine learning in which artificial neural networks perform layers of operations on a dataset in order to learn its patterns. A neural network consists of nodes, called neurons, and their interconnections. Neurons perform varying operations depending on their activation function and connections. Artificial neural networks with multiple layers of operation separating the input and output layers are known as deep neural networks. The manner in which neurons of a deep neural network are interconnected is known as its shape. The architecture of deep neural networks is a description of its shapes and other hyperparameters.

Machine learning applications are developed in accordance with the data they intend to model. Data including examples of how an input affects its output, an input-output pair, lends itself to supervised learning. Supervised learning applications attempt to map input values to a corresponding output value, reflecting the characteristics learned in the example data. Conversely, data which exhibits only input examples may be used in unsupervised learning applications. Unsupervised learning applications attempt to identify patterns in the example data to which new inputs can be compared.

### 2.5.2. Elements of Artificial Neural Networks

Machine learning applications revolve around the data they represent. Data is a collection of values organized in columns and rows, namely features and examples respectively. The features of a dataset describe the aspects being recorded whereas its examples are individual recordings. Within this report, example is the term used to describe individual rows of a dataset. The values of a feature form a probability distribution, representing their overall characteristics. Features may have independent value ranges which could lead to complications in the training of a machine learning model. Thus, values are often normalized to a range of zero to one using an equation such as the min-max scaling observed in equation 1. The scaled feature's value is given with respect to the minimum and maximum values of the feature's distribution. [4.]

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{1}$$

$$Where\ x\ is\ a\ feature's\ value,$$
$$x_{min}\ is\ a\ feature's\ minimum\ value,$$
$$x_{max}\ is\ a\ feature's\ maximum\ value,$$
$$and\ x_{scaled}\ is\ the\ transformed\ x.$$

Data is further manipulated in the process of data preparation. Data preparation involves the collection, preprocessing, and feature engineering of data [5]. Feature engineering is the process of developing additional features which often combine or exhibit informative characteristics of existing features. Feature engineering aims to improve the perfor-

mance of a machine learning model by incorporating domain expertise to guide its training. Domain expertise is the expert knowledge of an experienced researcher or engineer in a particular field or topic.

Datasets of a particular distribution are purposely divided into various subsets to enable the training, validation, and testing of a neural network model. Generally, each subset belongs to the same distribution while containing exclusive and unique examples. The training subset, or example data, often contains the significant majority of examples and is used to update the parameters of a model. The validation subset is used to verify the training process and enables model tuning without influencing the example data. The test subset is used to evaluate the model after its training and tuning processes have taken place.

The neurons of a neural network have parameters associated to them known as weights and biases. Weights describe the connection strength between various neurons within a network and are used to favor and reinforce certain neural pathways. Biases are values reflecting the learned characteristics of example data and are updated as the training process progresses. Deep neural networks propagate examples of training data back and forth throughout the network while updating their weights and biases accordingly.

Neural networks also exhibit hyperparameters describing the configurations of a model. Hyperparameters are generally selected before training and tuned using the validation subset. The number of layers in a neural network, the neurons per layer, and the activation function of a neuron are types of hyperparameters. Activation functions describe how data is processed in a network, particularly how a neuron's input is transformed at its output. The activation functions discussed in this report are the rectified linear unit (ReLU) function and the logistic sigmoid function presented in equations 2 and 3 respectively [6]. The ReLU function returns the input value if is positive, otherwise it returns zero. The sigmoid function returns values ranging from zero to one, where an input of zero results in an output value of 0.5. The sigmoid function returns values tending to zero for negative inputs and values tending to one for positive values, scaling with magnitude.

$$R(x) = \max(0, x) \tag{2}$$

$$Where\ x\ is\ a\ neuron's\ input\ value,$$

$$S(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

$$Where\ x\ is\ a\ neuron's\ input\ value.$$

Another hyperparameter of a network is its loss function. A loss function, such as mean-squared-error (MSE) presented in equation 4, provides a metric to determine how accurately a network's output represents its associated target. Due to the multidimensionality of the input and output layers, their neuron's values are iterated through and their errors are summed and averaged. The greater the overall error between the target and output values, the greater the MSE value.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{4}$$

$$Where\ y_i\ is\ a\ network's\ input\ value,$$
$$\hat{y}_i\ is\ a\ network's\ output\ value,$$
$$i\ is\ the\ iterator,$$
$$and\ n\ is\ the\ total\ number\ of\ iterations.$$

Certain hyperparameters, including the optimizer function and number of epochs, dictate the manner in which parameters of a network are updated during training. The optimizer function updates weights while observing the loss function to minimize overall loss. The adjustable learning rate parameter provides a step size for certain optimizer functions, such as the Adam optimizer based on stochastic gradient descent [7]. Generally, the training process consists of several iterations of parameter updating. These iterations are known as epochs and describe how many times example data is propagated through a network and its algorithms.

### 2.5.3. Autoencoder

A deep neural network architecture designed to efficiently learn a representation of training data is known as an autoencoder. Autoencoders learn from example data to output

values which closely represent their input values. The output of an ideal autoencoder equals that of its input. Although autoencoders can be implemented for unsupervised learning applications, they are self-supervised as the target output is the input [8]. Generally, autoencoders are designed with a bottleneck layer imposing a compression of the original input data. The number of neurons in the bottleneck layer is known as its encoding dimension and indicates the autoencoder's magnitude of compression.

The operation of an autoencoder assumes the features of the example data somehow correlate and possess a discernable pattern. An autoencoder may be conceptualized as the sum of an encoding and a decoding stage. The encoding stage consists of several layers of decreasing dimension. The features of example data are presented at the individual neurons of the input layer and are propagated forwards through the network until they become compressed at the bottleneck layer. The decoding stage attempts to reverse the process, reconstructing the compressed representation of the bottleneck layer into an output matching the input's dimensions. Decoding is an imperfect process, resulting in some reconstruction error. A loss function compares the reconstructed output values to their respective input values and the network's parameters are optimized to minimize the loss. [9.]

The performance of the training process for an autoencoder model may be monitored by its loss history. The loss history consists of training losses recorded after each epoch is performed. The loss should minimize as the autoencoder learns to improve the representation of its example data. Once the training loss has minimized and saturated, the autoencoder's parameters contain a representation of the patterns in the example data. Data not presented during training yet belonging to the same distribution as the training subset can be described as unseen data. Unseen data does not influence a model's parameter updating. Unseen data may be fed to the input of a trained autoencoder to be decoded with accuracy comparable to the final training loss. The loss is expected to be similar to the saturated loss if the unseen data closely resembles the training subset. However, a greater magnitude of loss is observable the more the unseen data differs from the training subset. This quality may be leveraged to detect anomalies by observing the loss response with respect to unseen data. Anomalous data would yield a greater loss than data closely representing the training data. An autoencoder trained on data acquired during periods of normal condition would learn to represent such a condition. Unseen data may be passed through the trained model to determine its loss and consequently its associated condition. [10.]

# 3. Methods and Materials

## 3.1. Workflow

The project was approached with a basic machine learning workflow in mind, adding consideration for the requirement of a visualization for the dashboard. The steps of the workflow are listed as follows:

1. Data acquisition and programming environment setup.
2. Data preparation, engineering, and investigation.
3. Select appropriate machine learning model.
4. Train, test, and evaluate the model.
5. Develop a traffic light system powered by the model.
6. Develop and automate a dashboard.

## 3.2. Environment Setup and Data Acquisition

When beginning the project, a tour of the physical manufacturing line was given. The software controlling the test equipment was described, including its process for storing measurement data. The first priority was to import the data into a programming environment to begin data preparation. Python was selected as the backbone of the programming environment as it is highly supported and compatible with numerous data science and machine learning frameworks. A virtual environment was created using the Anaconda distribution and typical data science packages were installed including Pandas, Plotly, and Jupyter Notebook. The Anaconda distribution is a data science platform and comes preloaded with relevant python packages while providing a convenient way to keep projects self-contained on a computer. Pandas is a data analysis and manipulation tool, used primarily for its dataframe capabilities. Dataframes provide a fast and robust way to manage and manipulate datasets, especially those of large dimensions. Plotly is a graphing library with an assortment of tools designed to help visualize data. Jupyter

Notebook is a web-based development environment that provides a way to experiment with and run code while having a legible and segmented structure. The notebooks have markdown support allowing text to be included which explain code segments. This platform was selected to enable a newcomer to the project to more easily understand and follow the code in the event that development on this project should continue after my contribution has ended.

The data is stored on a company network drive in comma separated value (CSV) text files. The test software controlling the testers generates the CSV files, logging all measurements recorded per tester per month in individual files. Python scripts were written to extract and combine the entire backlog of data into a leak dataset. The dataset was sorted by datetime and contains over a million samples, spanning over a year.

### 3.3. Data Preparation

Promising features were selected from the dataset and unnecessary features, such as test system version numbers, were discarded. The resulting dataset contains the features presented in table 1. All feature names are italicized when referred to in this report.

Table 1. A leak dataset sample with highlighted features.

| | Datetime | Lagtime | TestTime | Leak | Product | Label | DL |
|---|---|---|---|---|---|---|---|
| **0** | 2020-06-01 06:22:08 | 0 days 07:51:30 | 25 | 4.60 | 115.00 | 0 | 1 |
| **1** | 2020-06-01 06:22:22 | 0 days 07:54:17 | 15 | 4.60 | 69.00 | 0 | 2 |
| **2** | 2020-06-01 06:22:51 | 0 days 00:00:43 | 19 | 4.60 | 87.40 | 0 | 1 |
| **3** | 2020-06-01 06:25:38 | 0 days 00:03:16 | 15 | 3.72 | 55.80 | 0 | 2 |
| **4** | 2020-06-01 06:26:13 | 0 days 00:03:22 | 14 | 4.28 | 59.92 | 0 | 1 |

The most relevant features of the leak dataset are highlighted. The *DL* feature (orange) is a binary number indicating whether the example was logged from the primary (1) or auxiliary (2) leak tester slot. The *Test Time* feature denotes the duration (in seconds) the leak tester took to perform its measurement. The *Leak* feature represents the atmospheric leakage across the inside and outside of the water trap.

Feature engineering was performed to obtain the *Product* feature, the multiplication of *Test Time* and *Leak* values. This feature is also referred to by the more appropriate name '*dPxT*', describing the change in atmospheric pressure relating to test time. The project manager suggested the use of this feature as it had been useful in previous Testerwatch research. The *dPxT* feature's main application in this research is to provide a metric of the tester's measurements which is more easily monitored, combining two features into a one-dimensional feature space.

The remaining features in table 1 are not used in this research but are relevant in diagnosis. The *Label* feature is a binary number indicating whether an individual example passed or failed the leak test. However, the result of the leak test is irrelevant as this determination is made after the measurements are performed. Within the scope of this research, the *Label* value does not reflect any more information than the measurement values. The *Label* value may be used in other applications to determine the scrap rate. The scrap rate is an indication of how many water trap units fail the testing stages over time. Scrapped units are unusable and removed from the line.

The test system generating the CSV files performs some data cleaning such as discarding invalid test attempts that occur with negligible frequency and have little impact on the line's perceived performance. By consolidating the tester logs, the dataset inherits the cleaned characteristics. At this stage, the datasets appeared to be in good form as all values of a feature are sensible, numerical, and share the same datatype and scale.

## 3.4. Data Engineering

At this stage of design, the goal was to select a subset of the leak dataset that represents examples under normal operation. A representation of this subset is what will be used to evaluate future measurement data, provided it shares a similar distribution. To accomplish this, the data must be represented in a manner that allows a domain expert to distinguish normal operation from abnormal operation. When comparing an example to subsequent examples, it is not uncommon to notice relatively high variance in the values. This is expected as water trap units are unique, some are expected to fail the leak test.

The number of measurements performed by the leak tester's primary and auxiliary slots per hour is confidential. However, this amount is sufficient and enables the use of averaging. Thus, the dataset was downsampled to average each feature's examples into 20-

minute windows. Prior to downsampling, the examples were grouped with respect to which slot performed the measurement. Downsampling was performed in such a manner that each example contains 20-minute averages of the *Leak, Test Time,* and *Product* values for each slot. These features were grouped with respect to the slot they were measured by, a number one indicating primary and a number two indicating auxiliary. The new *L1, L2, TT1, TT2, P1,* and *P2* features represent each slot's *Leak, Test Time,* and *Product* respectively.

During downsampling, new features, *Count 1* (*C1)* and *Count 2* (*C2)*, were extracted and added to the downsampled dataset. The value of each count is simply the number of examples averaged in each 20-minute downsampling window per slot. The primary slot count is denoted as *C1* whereas the auxiliary slot count is denoted as *C2.* These features describe how active the tester is during each window, scaling in magnitude with increasing traffic in the line.

Examples collected during periods of line inactivity are irrelevant and may negatively influence the distribution of the downsampled dataset. During inactivity, no measurements are performed nor logged. However, due to the nature of downsampling into static windows of time it is possible to encapsulate a few examples immediately before or after a period of inactivity. These examples may negatively skew averaging and should be removed. Thus, the dataset was modified, removing examples with a count value below a certain threshold. This threshold, which will remain confidential, is selected with respect to the average rate of measurements performed by the tester. The dataset is modified using Pandas' dataframe manipulation methods.

Feature engineering was performed after the downsampling stage to produce two ratios relating to the tester's slots performance, namely the *count ratio (CR)* and *product ratio (PR)*. These two features are inspired by the voltage divider equation and are presented in equations 5 and 6.

$$CR = \frac{C2}{C1 + C2} \tag{5}$$

$$PR = \frac{P2}{P1 + P2} \tag{6}$$

The aim of the ratios is to provide a single value that indicates the influence a particular value has over the other. The *count ratio* indicates the influence the auxiliary's count has over the combined count, calculated as displayed in equation 5. As the tester software prioritizes the primary slot over the auxiliary slot, the value of *CR* should be at most 0.5 during periods of high traffic. This is to say the auxiliary slot does not perform more measurements than the primary slot in a given window of time during normal operation.

Similarly, the *product ratio* describes the influence of the auxiliary slot's *dPxT* value over the primary. This value, calculated as displayed in equation 6, describes how similarly each slot performs with respect to the measurements it completes within the 20-minute windows. During normal operation, *PR* should be close to 0.5. The *PR* is more easily understood when observing the interaction between each tester slot's *dPxT* values, observable in section 4 and described in section 5 of this report. It is practical to train deep neural networks with normalized data values ranging from zero to one, an intrinsic characteristic the ratios conveniently possess. The other features were normalized during the subset selection stage.

The dataset and features presented in the Data Preparation section have been substantially modified, resulting in a downsampled dataset with the features presented in table 2.

Table 2. A downsampled leak dataset sample with engineered features.

| Datetime | TT1 | L1 | P1 | TT2 | L2 | P2 | CR | PR |
|---|---|---|---|---|---|---|---|---|
| **2019-06-01 06:20:00** | 18.50 | 4.443750 | 82.90250 | 19.000000 | 4.597500 | 87.335000 | 0.333333 | 0.513019 |
| **2019-06-01 06:40:00** | 19.50 | 4.523333 | 88.55875 | 20.200000 | 4.551000 | 92.227000 | 0.294118 | 0.510145 |
| **2019-06-01 07:00:00** | 20.68 | 4.548800 | 94.37760 | 20.666667 | 4.596667 | 94.995556 | 0.264706 | 0.501632 |

To summarize, the dataset now contains examples of 20-minute averages of the *Leak, Test Time,* and *Product (dPxT)* features per tester slot alongside the *CR* and *PR* values. As a result of downsampling, the length of the dataset is reduced by several tenfold. However, the downsampled dataset remains several tens of thousands of examples long, adequate to successfully train the autoencoder. The actual tester datasets and their dimensions are confidential as they are the property of GEHC and productivity metrics shall not be described.

### 3.5. Selecting Subsets

The dataset now contains downsampled averages of the leak tester's features, including data acquired during periods of normal and abnormal operation. The ratio of normal to abnormal operation is heavily skewed towards normal, by design.

In order to select a subset of normal operation from the dataset containing all operating conditions, domain expertise about the tester is required. Several stages of filtering were applied to the dataset in order to exclude periods of undesired condition. The filters were selected with the aim of influencing which sort of input data would trigger a traffic light warning. All filters only affect the downsampled and averaged dataset. Filtering was performed using Pandas' dataframe manipulation methods.

Firstly, the dataset was filtered to only include examples with a *CR* value less than 0.5 indicating the primary slot is indeed used primarily. Secondly, the averages of 20 minutes of *dPxT* values should fall within the specification limits set in the tester software. This ensures the majority of tests are passed, expected during normal condition. Thirdly, the *dPxT* values of each slot should be similar indicating that both slots' tester components are operating similarly. This characteristic is exhibited with a *PR* value of about 0.5. At this stage, the average of the subset's *PR* values was also very close to 0.5. This precondition suggests the use of applying control limits centered around the average as a means to quantify similar slot operation. Using control limits it is possible to determine when values deviate from the controlled behavior [11].

$$Upper\ control\ limit\ (UCL): \quad UCL = \overline{PR} + 3 \times \sigma \tag{7}$$

$$Lower\ control\ limit\ (LCL): \quad LCL = \overline{PR} - 3 \times \sigma \tag{8}$$

$Where\ \overline{PR}\ is\ the\ average\ and\ \sigma\ is\ the\ standard\ deviation\ of\ PR\ in\ the\ filtered\ subset.$

The upper control limit (UCL) and lower control limit (LCL) equations are presented in equations 7 and 8, respectively. The upper and lower control limits provide the bounds for values that are considered in control. The control limits were used to filter the subset. The dataset was filtered accordingly with the intention of preparing a training subset that enables the autoencoder to represent controlled and normal tester behavior.

The resulting filtered subset was visualized to make final determinations, displayed in figure 6.

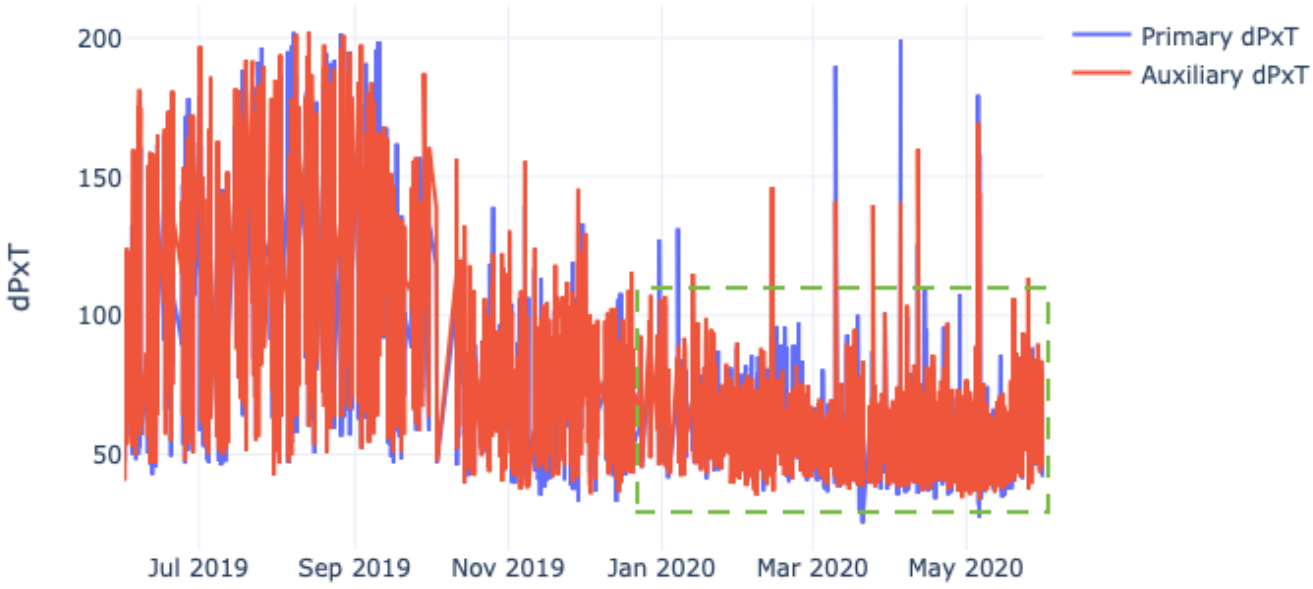


Figure 6. dPxT of filtered subset displayed, including various biases.

The graph in figure 6 displays about a year's worth of each tester slot's *dPxT* values after being filtered, indicating that these values correspond to normal tester condition. While it is nearly impossible to determine individual data points from such a view, an overall trend is discernible. Although all values displayed are considered normal, the variation and bias of their distribution improves dramatically beginning October 2019, considering lower *dPxT* values indicate better test performance. The zone denoted by a green dashed line, beginning in January 2020, exhibits further improvement of the distribution. The outlined area displays a particularly low bias and variance. This is consistent with the desired test performance GEHC's engineers work diligently to improve.

At this stage, consideration of the desired performance of the autoencoder was significant in selecting its training subset. If the entire dataset displayed in figure 6 were to be used to train the autoencoder, it would not necessarily flag values with a relatively high bias as alarming. This is undesirable as the improvements to the line would not be reflected in the autoencoder's representation. Instead, the outlined portion of data was to be used as the training subset. Although this effectively halved the dataset length, experimentation indicated the subset length was adequate for training performance. The resulting dataset will be referred to as the normal subset.

The training and validation subsets consist of the normal subset. The validation subset will be created during training by performing a 20% validation split handled conveniently by Keras. The model will be trained exclusively using the training subset. The validation subset is used to evaluate training performance and will not influence training the autoencoder. The test subset consists of the abnormal examples and the normal examples prior to January 2020. The test subset will be used to verify the traffic light colors respond appropriately, discussed in section 3.8.

Lastly, the features of the training (and thus validation) and testing subsets were normalized to improve the training performance of the autoencoder model. Scikit-learn's preprocessing package was installed into the environment for its MinMaxScaler class which implements min-max scaling. Using the MinMaxScaler, the *L1, L2, TT1,* and *TT2* features were normalized to a range of zero to one. The normalized dataset is in the form presented in table 3.

Table 3. A normalized training subset sample (Datetime index to be omitted).

| Datetime | TT1 | L1 | TT2 | L2 | CR | PR |
|---|---|---|---|---|---|---|
| **2019-06-01 06:20:00** | 0.149706 | 0.035722 | 0.163399 | 0.045357 | 0.333333 | 0.513019 |
| **2019-06-01 06:40:00** | 0.182975 | 0.036775 | 0.202614 | 0.044462 | 0.294118 | 0.510145 |
| **2019-06-01 07:00:00** | 0.222231 | 0.037112 | 0.217865 | 0.045341 | 0.264706 | 0.501632 |

Scaling was performed on the entire downsampled dataset, including both normal and abnormal operation, in order for the calculations to take the full distribution of the data into account. The *P1* and *P2* features were discarded as the neural network will inherently perform more relevant combinations of the *Leak* and *Test Time* features during training.

### 3.6. Model Selection

An autoencoder must be designed such that it learns to efficiently represent the normal dataset. The programming environment was updated with the TensorFlow 2 framework. TensorFlow is a fully featured machine learning platform developed by Google. It provides tools to build, train, evaluate and deploy machine learning models. This version of TensorFlow is preloaded with Keras, a deep learning API that leverages TensorFlow's infrastructure. Keras is useful for developing and prototyping deep learning models.

Using Keras, an autoencoder is built by stacking layers of neurons in the desired network shape while specifying other hyperparameters. The training data features dictate the input and output layer dimensions of the autoencoder.

```
input_layer = Input(shape=(6,))
encoded = Dense(5, activation='relu')(input_layer)
encoded = Dense(4, activation='relu')(encoded)
encoded = Dense(3, activation='relu')(encoded)

decoded = Dense(4, activation='relu')(encoded)
decoded = Dense(5, activation='relu')(decoded)
decoded = Dense(6, activation='sigmoid')(decoded)
autoencoder = Model(input_layer, decoded)
```

Listing 1. Stacking layers of the autoencoder using Keras, written in python [8].

The code in listing 1 builds an autoencoder model consisting of seven layers with six neurons at the input and output layers and an encoding dimension of three. The neurons in the layers of the autoencoder were specified with the activation functions depending on their purpose. The rectified linear unit (ReLU) function was used as the activation function for every neuron except for those in the output layer. The activation function for the output layer's neurons was specified as the sigmoid function.

The code presented in listing 1 produces an autoencoder neural network whose shape is displayed in figure 7.

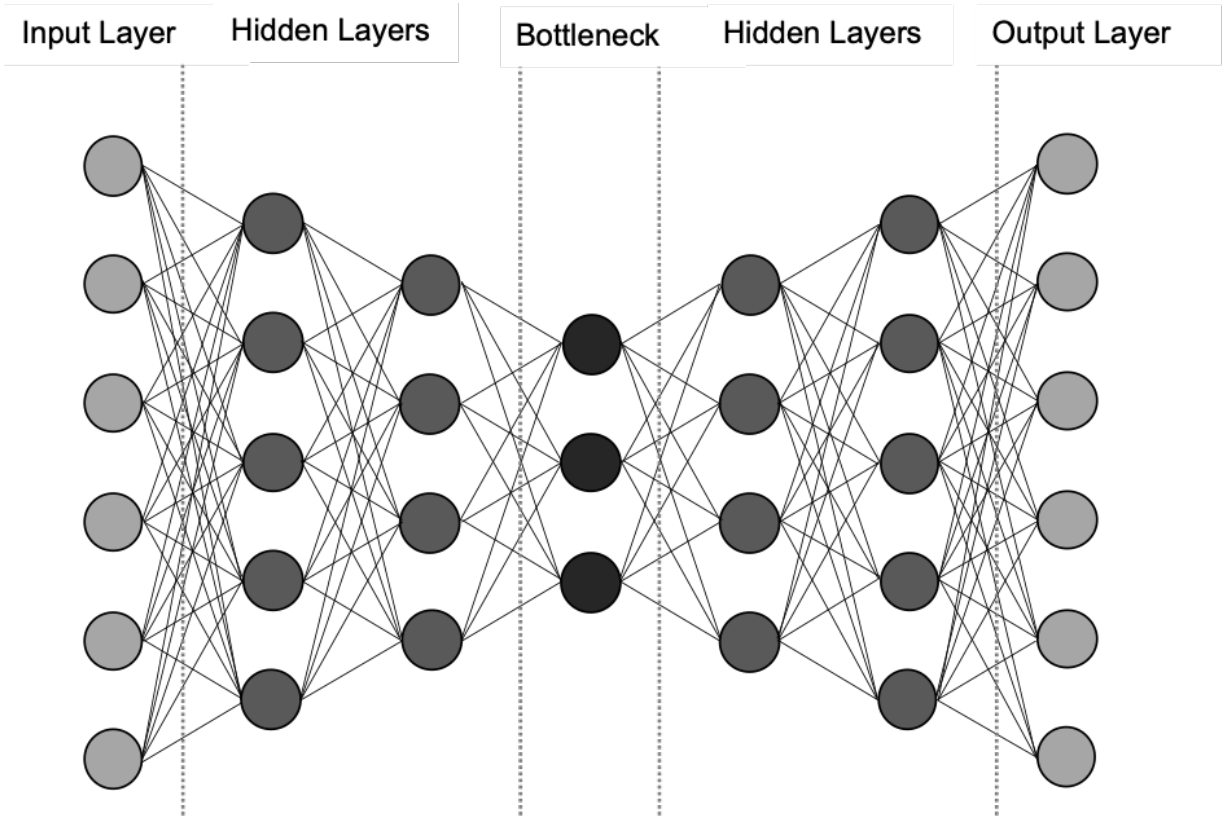


Figure 7. The autoencoder's shape with indication of layer types.

The network displayed in figure 7 demonstrates how the input data would be encoded into a smaller dimension and decoded into its original dimension. The bottleneck layer consisting of three neurons may also be referred to as the autoencoder's encoding dimension.

## 3.7. Model Training

The selection of hyperparameters greatly influences the performance of the autoencoder. Brute force, educated guessing, and trial-and-error were the main methods of finding the best hyperparameters, such is the standard practice at the time of writing. Many combinations of hyperparameters were experimented with while monitoring their resulting training and validation losses. The final hyperparameter selection is discussed in the following.

Keras does not accept Pandas dataframes as an input type. The environment was modified with the installation of the NumPy package which provides a form of data management compatible with Keras. The dataset dataframes are ordered by a datetime index. However, this information was not required during training as the datasets would undergo shuffling. Thus, the training and test subsets were reindexed using integers and the datetime feature was removed. The subsets were then converted to NumPy arrays.

Prior to training, Keras requires a model to be compiled with the desired optimizer function and its associated learning rate as well as the loss function. The autoencoder was compiled using the Adam optimizer with a learning rate of 0.0001 and Keras' mean squared error (MSE) loss function. Listing 2 presents the code declaring the optimizer and loss function.

```
autoencoder.compile(optimizer=Adam(learning_rate=0.0001),
                    loss='mean_squared_error')

ae_fit = autoencoder.fit(train, train,
                         epochs=20,
                         validation_split=0.2,
                         batch_size=10,
                         shuffle=True,
                         verbose=2)
```

Listing 2. Code to compile and train the autoencoder with specified hyperparameters [8].

Keras' training method allows users to specify several arguments, hyperparameters related to training. Firstly, the training data was specified to be the training subset. Secondly, the test data was also declared to be the training subset due to the nature of the autoencoder attempting to reconstruct its input. Thirdly, a batch size of 10 and 20% validation split ratio were selected. Lastly, training was instructed to occur for 20 epochs with a shuffle performed at the beginning of each epoch. Shuffling performed at every epoch prevents the model from learning order in the data.

The model was trained as configured and its associated history of training and validation loss was recorded. The trained model and loss history were saved and can be reloaded for use without retraining. The autoencoder code was inspired by the Keras guide involving autoencoders and the MNIST dataset [8].

## 3.8. Traffic Light System Development

The traffic light system consists of two visual components, namely the traffic light graph and the input data graph. The trained autoencoder model is used to determine how closely input data resembles the learned representation of normal operation. This is measured by the MSE loss metric which is used to quantify how distinct an input is from the representation. The traffic light system was designed to leverage this characteristic to determine the magnitude of change in condition of the leak tester.

This was implemented by applying thresholds of the MSE loss values relating to the colors of the traffic light. A green color should indicate that the input data resembles a normal condition. A yellow color should indicate an abnormal, although not extreme, deviation from normal operation. A red color should indicate the input data has little to no resemblance of a good condition.

The respective thresholds were selected using the autoencoder's loss history resulting from an input of the normal subset. This history was transposed by subtracting the training's final epoch's loss value from each history value. This transposition simply moves the average of the history's distribution to be very close to zero, allowing the thresholds to be determined on a near absolute scale. This characteristic improves the visual distinction of abnormal cost values when observing a graph of the loss history by eliminating the training loss offset. Further mentions of the transposed loss values are referred to as absolute loss within this report.

Given the dataset represents normal operation, the vast majority of examples corresponding to this absolute loss history should yield a green color in the traffic light. Examples with a cost greater than any value given in this history's distribution should raise yellow or red colors, respective to magnitude. In the interest of leaning towards false alarm rather than missed alarm, the threshold for a yellow color was set with some tolerance in mind. A yellow color should correspond to values that are greater than the upper control limit of the history's distribution. The yellow color's threshold was set to be this upper control limit. Values with a red warning should greatly exceed any values observable in the loss history. This threshold was set to be twice the maximum observed value. The test subset was used to validate adequate performance of the declared thresholds.

Once the thresholds were established, a script was written to handle the inputs and outputs of the autoencoder. The purpose of the script is to prepare data for visualization in the graphs by applying the autoencoder and the color thresholds. Any autoencoder input data used is prepared in exactly the same manner as the training subset, undergoing the same manipulations and using the same normalization scaler. Its loss value is also transposed in the same manner before filtering by thresholds. Within the context of this report, the output of the autoencoder is the input's associated loss, not the reconstructed values at its output layer.

The script transforms and prepares an input example or dataset and passes it through the autoencoder. The script adds a feature to the input dataset containing the absolute loss. A conditional script creates another new feature containing the color relating to an input absolute loss. The input dataset's original *Leak, Test Time, CR,* and *PR* features remain unaltered. The input dataset's new features for absolute loss and color are used in the traffic light graph.

The traffic light graph was designed to indicate condition by displaying the color associated to a 20-minute window of tester performance. The data presented in this view is created by the autoencoder's loss response to its input. The graph is essentially a bar graph with bars of constant height and varying color. The graph was designed to be interactive, allowing users to adjust the time on the x-axis to access a historical view of the traffic light colors over various periods of time.

The input data graph's purpose is to corroborate the results of the traffic light graph. The data presented in this corroborative graph is not influenced by the autoencoder. The leak tester's data is displayed on an interactive line graph, presenting the *dPxT* (per slot)*, CR, and PR* features. Additionally, dashed lines indicating the target value and specification limits for *dPxT* are included. The data presented in this graph was downsampled and the same features were generated. However, the data was not filtered and thus accurately resembles the tester's performance for each 20-minute window. The graph is interactive, allowing users to adjust the time on the x-axis to access different ranges of examples. The scale of the ratios is intentionally left unaltered as to not visually interfere with the *dPxT* lines. Below the graph is a rangefinder, enabling quick jumps along the time axis. The graphs and interactive controls were designed using Plotly's Graph Objects module.

As the development of the traffic light system is considerably ‘hands-on’, a brief summarization is given. The system consists of scripts, a trained autoencoder, and two graphs. The system transforms input data into one of two views. One view displays a historical graph of colors related to the condition of the leak tester, powered by the autoencoder. The other view is kept separate from the autoencoder and displays the input data used to determine these colors, enabling a user to corroborate the traffic light graph’s results. Samples of these views are presented in section 4.2.

The code used to build and populate the graphs is omitted as it is the property of GEHC and is not the focus of this report. However, the scripts and graphs are mentioned to provide insight on any influence they may have had on the development of the autoencoder-based condition monitoring solution. Furthermore, their mentions help to explain the traffic light graphs’ appearances in section 4 of this report.

### 3.9. D-fend Dashboard Development

In order to observe the condition monitoring solution, the D-fend dashboard was developed to be displayed on a large monitor in the production area. To enable further development of the dashboard and given previous use of Plotly, the dashboard was developed using JupyterDash. JupyterDash provides Plotly Dash tools integrated into Jupyter Notebook to deploy a web-based dashboard while providing support for Plotly graphs. The structure of the dashboard was built using mainly Dash Core Components, included in the installations.

The dashboard consists of visual and nonvisual elements. The visual elements have been discussed and include the traffic light graphs, their interactive buttons, and tabs for accessing the diagnosis tools. The nonvisual elements include a data pipeline and automation. The purpose of automation is to enable the graphs to periodically update to include new tester data stored on the server. The automation processes are triggered via callbacks implemented within the dashboard, which are only triggered when the dashboard is running. The purpose of the data pipeline is to access and prepare newly recorded tester data on the server for use with the graphs. Any incoming data must be processed and prepared in exactly the same manner to ensure consistent performance of the autoencoder. This includes feature generation, downsampling, filtering, dataframe manipulation, normalizing (with the preconfigured scaler), and prediction.

The code used to build, automate and deploy the dashboard and graphs is omitted as it is the property of GEHC and is not the focus of this report. However, the dashboard and graphs are mentioned to provide insight on any influence they may have had on the development of the autoencoder-based condition monitoring solution. Furthermore, their mentions help explain the dashboard's and graph's appearance in section 4 of this report.

# 4. Results

## 4.1. Training Performance

During training of the autoencoder two loss values were recorded after each epoch, the training subset loss and the validation subset loss.

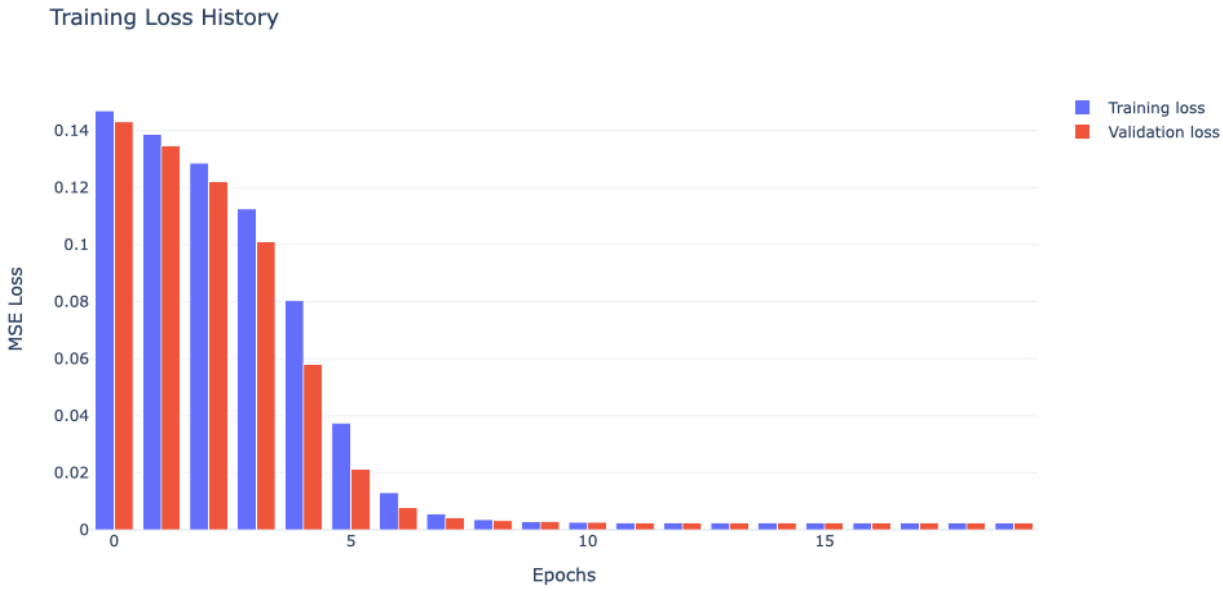


Figure 8. The autoencoder's training and validation loss history.

Training loss and validation loss have converged and saturates at an MSE value of approximately 0.0024. The training loss history is evaluated in section 5 of this report.

## 4.2. Dashboard Samples

The dashboard, including the traffic light graph and input data graph, automatically updates periodically. The graphs are interactive allowing users to pan and zoom the graphs. Additionally, both graphs feature mouseover text, revealing actual values for the targeted data point. To accommodate the static nature of a report, several samples are presented with a description of their point of interest.


Figure 9. Typical sample of the dashboard and its views, describing a situation for July 15, 2020.

Figure 9 displays a typical view of the dashboard with the traffic light tab selected. The other tabs, visible in the topmost portion of the sample, are implemented for diagnosis but are not the focus of this report. The visible elements and the current situation are described as follows.

The upper graph observable in figure 9, displaying colored bars, represents a historical view of the traffic light. This view is powered by the autoencoder. Each bar represents a 20-minute window of the leak tester's performance. The x-axis displays a range of time which may be shifted, narrowed, or widened to the user's needs. The lower graph, displaying several lines, exhibits the features listed in the legend over a selectable window of time. This view is not influenced by the autoencoder. Each point along a continuous line represents a 20-minute average of the tester's actual measurements for the selected window of time. The lines interpolate from point to point to improve visual clarity. The upper and lower dashed lines represent the upper and lower specification limits, respectively. The central dashed line represents the target value for which the tester is designed. Below this graph is a rangefinder, displaying a large range of the dataset to enable quick jumps along it.

The situation presented in this sample can be quickly determined by looking at the upper traffic light view. The majority of the line's performance over the period of three days is considered normal and is indicated by the green bars. The normal performance is also visible in the corroborative view, displaying the manner in which the *dPxT* values for each tester slot are closely similar throughout time. Generally, normal performance is visible when the orange and blue lines are near each other. White bars simply denote inactivity of the manufacturing line. However, around midnight of July 15, the traffic light warns of abnormal tester performance indicated by yellow and red bars. This abnormal period lasts for several hours and is followed by line inactivity. Directly below the bar graph, occurring at the same time, a sudden upwards jump of the *dPxT 1* value is visible in the line graph.

A second sample of the dashboard and its condition monitoring views is presented in figure 10, describing a situation for July 15, 2020.


Figure 10. A second sample of the condition monitoring views for a situation on July 15, 2020.

Figure 10 displays a similar view to figure 9 of the same situation. The window of time selected for both graphs has been narrowed down to a day centered around the abnormal performance. Furthermore, mouseover text is visible for the red bar representation at 23:00. The bar graph reveals the absolute loss, a value of 0.165, for its respective example data resulting from application of the autoencoder.

A third sample of the dashboard and its condition monitoring views is presented in figure 11, describing a situation for June 14, 2020.


Figure 11. A third sample of the condition monitoring views for a situation on June 14, 2020.

Figure 11 displays a different situation of a short period of abnormal operation occurring on June 14. The emphasis of this sample is on the mouseover text visible in the line graph. Tags appear next to each line graph indicating its value associated to the point in time the mouse is targeting. Additionally, the tag for *dPxT 1* displays the color of the traffic light of the example. However, this is merely a feature added to the dataset and does not influence the line graph's representations.

The fourth and final sample of the dashboard and its condition monitoring views is presented in figure 12, describing a situation for July 30-31, 2020.

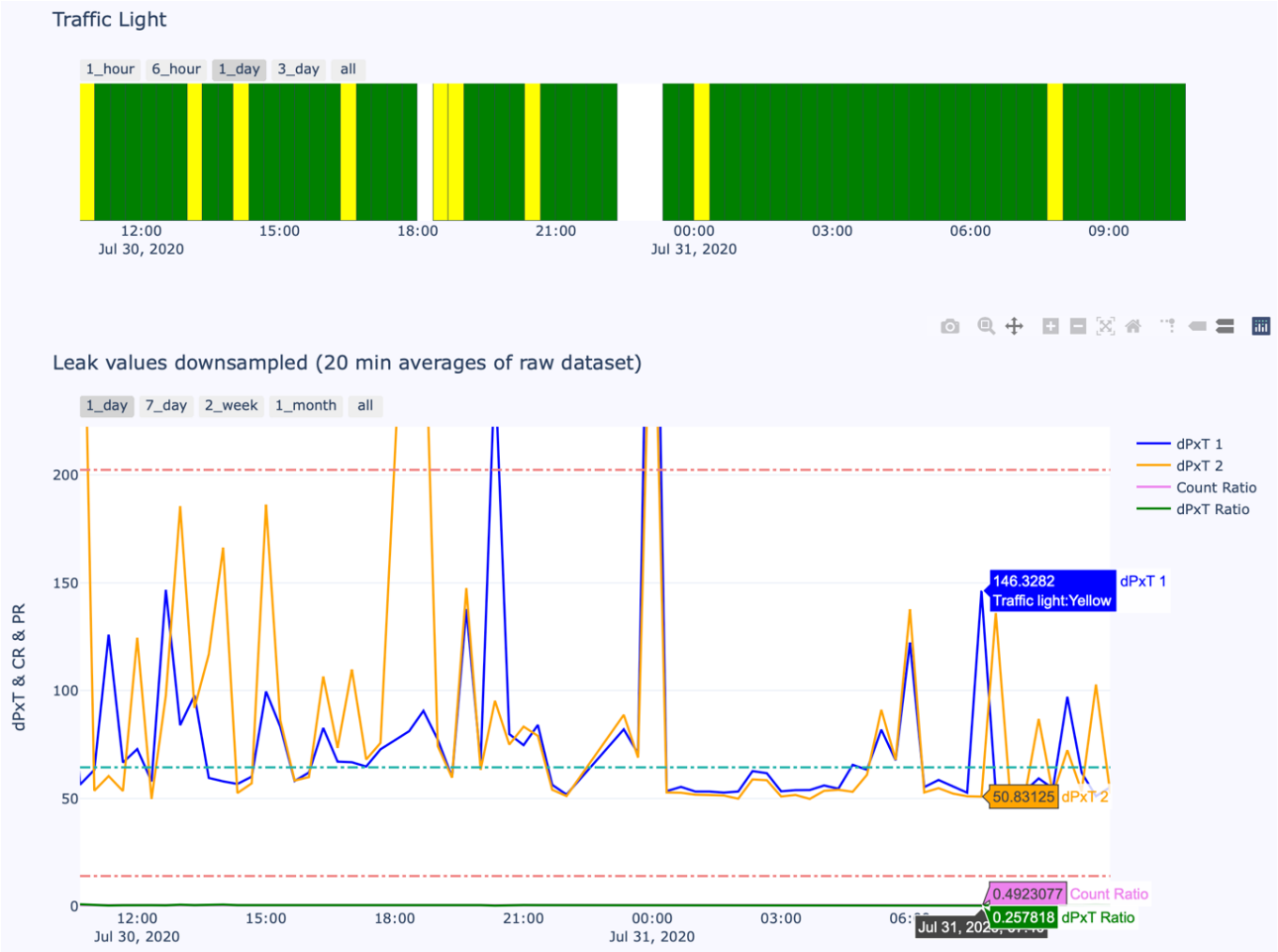


Figure 12. A fourth sample of the condition monitoring views for a situation occurring during on July 30-31, 2020.

Figure 12 displays yet another situation occurring during July 30-31, 2020. The situation displayed is a period of uncharacteristic tester performance. This sample intends to reveal the autoencoder's attempt to discern a particularly difficult case.

All the samples presented are evaluated in further detail in section 5 of this report. However, the fact that the traffic light view's alerts correspond to the input data's alarming examples is an indication of a working condition monitoring solution. This is satisfying result considering the two views' data are derived with completely separate methods.

# 5. Discussion

## 5.1. Training Evaluation

The graph in figure 8 represents the training performance of the autoencoder with respect to its declared loss function. At a glance, the graph displays a very satisfying trend for training performance. The training loss and validation loss both decrease uniformly as the parameters of the neural network update after each epoch. The losses are converged and saturated. This indicates that the model is neither underfitting nor overfitting, suggesting it will generalize well to unseen data. This assumes the unseen data follows the characteristics of the training data's distribution. Furthermore, saturation is established by approximately the tenth epoch. The more epochs trained after this point the greater the risk of overfitting. It is unknown the exact impact the additional epochs may have had in this regard. This result was the most promising out of the entirety of the hyperparameter experimentation phase.

The MSE loss value is a measure of how well the autoencoder reconstructs the input at its output. This is measured for both the training subset and the validation subset. The lower the value, the better the reconstruction. However, these values are likely deceptively low due to the normalization of the input data. Normalization is required to effectively train the neural network. However, I suspect the distribution used to set the normalization scaler was not adequately prepared. As the MinMaxScaler is prone to outliers, further filtering of the training data would likely improve the validity of the MSE loss metric. Despite this, I believe the generalization assumption holds given any input data uses the same scaler.

The decision in section 3.5 to select a training subset from the distribution with the lowest variance and bias also impacts generalization. In the event that the tester's distribution continues to change over time, the autoencoder must be retrained on a training subset representative of the updated distribution.

The test subset's examples and loss responses were investigated case-by-case to ensure adequate separation from the saturated trained loss. The investigation was performed to determine whether the traffic light thresholds were set appropriately. About 98% of all examples considered abnormal were flagged as either a yellow or red warning.

The distinction between yellow and red warnings should be determined by a technician with regards to the desired sensitivity for alerts.

## 5.2. Dashboard Sample Evaluation

The dashboard sample visible in figure 9 displays a satisfying result. The goal of monitoring the leak tester's performance via an autoencoder is essentially accomplished. The model has learned to represent the filtered training dataset with practical accuracy. This is apparent when considering the line graph and focusing on the behavior of each tester slot's *dPxT* values, which are a representation of the tester's overall measurements. When these values are within specification limits and are similar to each other their resulting loss detected via the autoencoder does not trigger a yellow or red warning. However, when these values differ by a substantial amount a warning is triggered.

Figure 10 displays an absolute loss associated to the divergent behavior which is significantly greater than the saturated training and validation losses. This indicates that the thresholds deciding traffic light color are impactful and perform the desired function. These thresholds may be tuned to adjust the sensitivity of the traffic light and should ultimately be decided by an engineer with domain expertise over the matter.

The motivation for creating the product ratio, *PR or dPxT Ratio*, is clearly visible in figure 10. When each tester slot's product (*dPxT)* value behaves in a manner visible in the latter half of the line graph, the *PR* value is close to 0.5. However, in figure 11 the mouse-over reveals a green tag displaying a low *PR* value of 0.026 when the slot's measurements are divergent. Furthermore, the targeted case also has a count ratio, *CR,* above 0.5 indicating the auxiliary slot has taken over the majority of the process. Both of these are undesired characteristics and may warrant investigation for maintenance needs.

Figure 12 exemplifies an instance of when the traffic light system may be used to detect less obvious abnormalities. While the majority of measurement values are within specification limits, the traffic light indicates a deviance in performance between the slots. This feature may be beneficial in identifying maintenance needs before an issue has become drastic. Figure 12 also demonstrates the value of a corroborative view for the traffic light bar graph, improving its diagnosis capabilities.

## 5.3. Design Considerations and Shortcomings

The interesting characteristic of training an autoencoder to represent normal operation is the absence of explicitly declaring limits for various combinations of input features. Due to the nature of the neural network, all input features are combined in complex variations that are likely unintuitive for a human engineer. There may be interactions occurring that would indicate alarming performance in the input data that are completely unnoticed by myself and other expert observers. By only applying thresholds to the resulting loss, it is possible to be alerted of a situation to which human experts are oblivious to. However, this characteristic also increases the ambiguity of such a solution. As a result, the system should only be used as complimentary solution to tradition statistic monitoring methods as it cannot guarantee accurate and reliable condition monitoring.

When designing the system, a particularly difficult data aspect to take into account was the temporal state. The raw data stored on the company server is timestamped and several measurements are performed every few minutes. The decision to downsample the data arose from my previous work with the leak datasets. I had failed to find correlation in the measurement values as a time series when applying my, albeit limited, data science techniques. However, research and discussion indicated that this is to be expected as the nature of maintenance requirements is undefined and limitedly understood. Thankfully, downsampling provided a crude, yet practical, means of incorporating a temporal aspect of the data. In discussion with the project manager, it was agreed that a 20-minute resolution of awareness is adequate to monitor the ongoing condition of the tester.

The development of the monitoring solution involved the creation of several systems ranging from data acquisition and transformation to displaying the results. In the interest of reaching a practical use case of the application, development progressed as swiftly as possible. Although the outcome was practical, hasty design decisions were implemented.

The preparation of the training data was limited by my understanding of the tester's condition and its reflection in the data. This limitation is an indication of my level of domain expertise in the matter. Data preparation inherits any assumptions made about the condition of the tester. The level of domain expertise and assumptions made are reflected in the representation of normal condition. Closer cooperation with the D-fend line's experienced engineers would have undoubtedly improved the validity of the representation.

The development of the autoencoder and its history of hyperparameter tuning were not logged nor recorded. The development progressed predominantly with a trial-and-error approach. The autoencoder's hyperparameter selection was determined with a desired loss response in mind. The intention of linking traffic light colors to the MSE loss of input data led to rudimentary hyperparameter selection choices. Hyperparameter tuning was performed while comparing the autoencoder's loss response for the validation and test data. As soon as the responses were distinct enough to apply thresholds to, the hyperparameter tuning phase ended. Although this provided practical results, the autoencoder design is not completely theoretically sound. For example, the use of the ReLU activation function is prone to vanishing outputs at the neurons for values that approach zero. Coupled with the MinMaxScaler issue discussed in section 5.1, it is possible the network had issues performing backpropagation. Furthermore, different optimizer functions were not experimented with.

The normal condition representation also lacks a ground truth to be used in validation. Thus, the traffic light's warnings must be investigated case by case to evaluate their reliability and validity. This results in a lack of empirical analysis of the true performance of the traffic light system. However, should adequate maintenance information be collected alongside its related measurement data, such an analysis could be conducted.

## 6. Conclusion

The D-fend dashboard is used alongside other D-fend manufacturing line monitoring tools in the production area of GEHC. The other tools monitor statistics related to the scrap rate and measurement values of the water trap units. The D-fend dashboard's value is in assessing the tester's data to determine its condition without relying on the scrap rate. This is significant as the scrap rate may only indicate abnormal tester condition retroactively. The dashboard's most significant capability is being able to determine a state of condition without explicitly monitoring the tester itself. Although results represented in the dashboard may not be completely reliable, they do provide a justifiable warning for maintenance needs.

The actual normal condition of the tester is dynamic and may be represented in endless ways. The approach of training an autoencoder to build a reference representation is

flexible as further development allows for the representation to be adjusted. This adjustable characteristic essentially allows engineers with domain expertise to improve the reliability of the traffic light's warnings. This is accomplished by improving the training subset preparation process, resulting in a more concise definition of normal condition. The thresholds defining traffic light colors would then be updated accordingly, adjusting the sensitivity of the warnings.

As the development of the Testerwatch project and D-fend dashboard continues, other condition monitoring techniques and methods may be worthwhile investigating. With regards to the research described in this report, emphasis on the selection and preparation of the training subset are paramount. The performance of any machine learning and deep learning application is superseded by the quality of its training data. Furthermore, hyperparameter search tools are constantly being developed and improved during the time of writing. These tools may save substantial experimentation time, allowing development time to be more appropriately allocated.

Alternative machine learning techniques may also improve the development of a condition monitoring system. An autoencoder leveraging a long short-term memory (LSTM) network may be able to detect useful patterns in the leak tester data. An LSTM is an artificial recurrent neural network that is trained on blocks of data spanning over a determined time frame. LSTM and autoencoder techniques may be coupled to circumvent the downsampling convention used in this research. However, the development and research involved to implement such a system is much more intensive.

In conclusion, the developed D-fend dashboard is a functional condition monitoring solution for the D-fend line's leak tester. The approach of implementing an autoencoder is practical given the undefined nature of actual normal condition. The implementation of a traffic light view with accompanying corroborative view allows for the warnings to be explained. The research performed sheds light on the possible capabilities of predictive maintenance solutions given the current state of the D-fend line's data collection. Further development would not require a complete redesign of the monitoring solution. Improvement of reliability and validity would begin by revisiting training subset preparation and threshold tuning.

## References

Layout of this page in the number (Vancouver) referencing system:


1. GE Healthcare Services. D-fend Pro+ Water Trap Product Page Containing Image and Description [online]. GE Healthcare Storefront: GE Healthcare;
   URL: https://services.gehealthcare.com/gehcstorefront/p/M1200227.
   Accessed 25 November 2020.
2. What is Predictive Maintenance? Benefits and Examples [online]. Fiix;
   URL: https://www.fiixsoftware.com/maintenance-strategies/predictive-maintenance/.
   Accessed 11 August 2020.
3. Mitchell TM. Machine Learning. Singapore: McGraw-Hill; 1997.
4. Loukas S. Everything You Need to Know About Min-Max Normalization: A Python Tutorial [online]. Towards Data Science; 28 May 2020
   URL: https://towardsdatascience.com/everything-you-need-to-know-about-min-max-normalization-in-python-b79592732b79.
   Accessed 11 November 2020.
5. Scott L. Data Preparation for Machine Learning: The Ultimate Resource Guide [online]. Lionbridge; 9 July 2020
   URL: https://lionbridge.ai/articles/data-preparation-for-machine-learning-the-ultimate-resource-guide/.
   Accessed 29 October 2020.
6. Wood T. What is the Sigmoid Function? [online]. DeepAI;
   URL: https://deepai.org/machine-learning-glossary-and-terms/sigmoid-function.
   Accessed 30 October 2020.
7. Keras API Reference. Adam Optimizer [online]. Keras;
   URL: https://keras.io/api/optimizers/adam/.
   Accessed 12 July 2020.
8. Chollet F. Building Autoencoders in Keras [online]. The Keras Blog; 14 May 2016
   URL: https://blog.keras.io/building-autoencoders-in-keras.html.
   Accessed 13 May 2020.
9. Jordan J. Introduction to Autoencoders [online]. Self-publication; 19 March 2018
   URL: https://www.jeremyjordan.me/autoencoders/.
   Accessed 10 May 2020.

10. Kienzler R. Using Keras and TensorFlow for Anomaly Detection [online]. IBM Developer; 2 March 2018
URL: https://developer.ibm.com/tutorials/iot-deep-learning-anomaly-detection-5/.
Accessed 24 July 2020.

11. Berardinelli C. The Complete Guide to Understanding Control Charts [online]. ISIXSIGMA;
URL: https://www.isixsigma.com/tools-templates/control-charts/a-guide-to-control-charts/.
Accessed 14 November 2020.