NTNU | Faculty of Engineering
Department of Structural Engineering

Department of Structural Engineering

TKT4550 - Structural Engineering,
specialization project

Machine Learning-Assisted Evaluation and Predictive Analysis of Concrete Structures

Written by:
Erling Nordli Husøy
Emil Hæreid Steen

December 2023

Acknowledgments

We are glad for introducing this project that has a purpose of showcasing the potential of machine learning applications in reading strain development in a concrete bridge. This project shows the relevance for machine learning in the workflow of structural engineers. As newcomers to the fields of machine learning and computer science, we take pride in demonstrating how we addressed a problem through the view of structural engineering students in Norway in 2023.

We will tank our supervisor, Adjunct Associate Professor Morten Engen, his help played a important role for the success of this project. His valuable support helped us navigate challenges in code collaboration on GitHub, simplify data processing, and tackle complex problems. We appreciate his assistance and commend his expertise as a skilled structural engineer who is commitment to continuous learning.

Through this project we jumped into the realm of open source software. Our experience in this environment has given us a deep understanding of its emphasis on shared learning and knowledge exchange. We also want to thank the open source community, the open source community makes it possible to advance and solve complex problems. We see the potential in open source and want to integrate more open source practices into our future work. Finally, we would like to thank Associate Professor Daniel Cantero and Associate Professor David Morin for their initial guidance and valuable information regarding the project.

We recommend that readers unfamiliar with machine learning and artificial intelligence visit the first part of the methodology chapter, where several machine learning terms are explained. We want this project to be a contribution to other works. We also hope readers will find it accessible and appreciate the practical utility of machine learning.

The potential for machine learning in structural engineering appears to be significant. We are eager to contribute to the future application and look forward to delving further into the ML field in our master's thesis.

We are pleased to present this project, signed by:

Erling Nordli Husøy

Emil Hæreid Steen

Abstract

The Herøysund bridge is a post tention bridge constructed at Herøy, Norway in year. The bridge had a significant challenge during construction when a important loss of pretension led to crushing at the anchorage points, this failure compromised its structural integrity. As part of the FoU project, strain gauges were installed on Herøysund Bridge in September 2020 to monitor strain and crack progression at different locations. There were sixteen sensors in total, including 13 strain gauges, two omega sensors, and a one PT100 temperature element, are currently monitoring the bridge's structural health, with seven gauges added in May 2023. In this report we implement Machine Learning (ML) to train on one strain signal to be able to measure the relationship between time, temperature and strain development.

The reason for the problem statement is to evaluate if Artificial Neural Networks (ANNs) or a Recurrent Neural Networks (RNNs) is better at predicting the relationship between time, temperature and strain development in the Herøysund Bridge. The focus of the assessment comparing the two models predictive accuracy for strain development in the bridge due to temperature and time differences. The effectiveness of each model is assessed based on numerical score metrics and visual comparison between predicted and actual values.

The ANN model was tuned to achieve more accurate predictions, however tuning of the RNN model was to computationally expensive for normal computers, so we unable to implement the tuning process for the RNN model. The tuning process of the ANN model made adjustments in layers and neurons, led to a more tailored approach, enhancing its prediction capabilities. The RNN model could have benefited form fine-tuning to achieve more accurate predictions.

The ANN model, throughout its evaluation, showed a better performance than the RNN model across training, validation, and test datasets. The model's effectiveness was not only in its ability to predict strain development but also in how it processed data and adapted through tuning. The ANN's architecture, known for its straightforwardness, was advantageous in handling the static nature of the input data, making it more suitable for the task at hand. In contrast, the RNN model, specifically designed to handle sequential data, faced challenges. While its design inherently allows for memory of previous inputs, this feature did not translate effectively predictive performance for strain development in bridges.

The ANN model outperformed the RNN across all metrics, particularly in test data prediction, indicating better generalization and predictive accuracy, as shown by MSE, MAE, and R2-score, and reinforced by its more effective visualizations and realistic seasonal variations; however, both models exhibited limitations in predictive accuracy, necessitating careful interpretation of their future and long-term predictions.

Table of Contents

1 Introduction **1**

1.1 Herøy FoU . . . 1

1.2 Sensor Installation . . . 1

1.3 Machine learning . . . 3

1.3.1 Supervised learning . . . 4

1.3.2 ANN - Artificial Neural Networks . . . 6

1.3.3 RNN - Recurrent Neural Networks . . . 6

1.3.4 Time series data (Sequential data) . . . 7

1.3.5 TensorFlow and Keras . . . 8

1.4 Literature Review . . . 9

1.5 Machine Learning for Structural Engineering: A State-of-the-Art Review . . . 10

1.5.1 AI and Machine Learning for Coders . . . 11

1.6 Research Problem . . . 11

2 Methodology **12**

2.1 Definitions . . . 12

2.2 Selection for method . . . 14

2.2.1 Inspection of conducted data collection . . . 14

2.2.2 Procedure of implementation . . . 17

2.3 Data processing . . . 18

2.4 Machine learning . . . 22

2.4.1 Hyperparametric tuning . . . 24

2.4.2 Evaluation of model performance . . . 25

2.5 Critique of methodology . . . 25

2.5.1 Validity . . . 26

2.5.2 Reliability . . . 26

2.5.3 Objectivity . . . 27

2.5.4 Generalizability . . . 27

3 Results **28**

3.1 ANN model 28

3.2 RNN model 32

4 Discussion **36**

4.1 Which model, standard ANNs or RNNs, is more effective for predicting the specific strain development in the bridge? 36

5 Conclusion **40**

6 Recommendations and Future Work **41**

Appendix **44**

A ML predicitons for all sensors at once training-data 44

B ML predicitons for all sensors at once validation-data 45

C ML predicitons for all sensors at once test-data 46

1 Introduction

In this chapter, we'll start with a quick introduction to the history and theoretical background relevant to our thesis. First, we'll describe the case study that forms the foundation of our project. Following that, we aim to explain some fundamental concepts and theories of machine learning (ML). This is crucial for those outside the artificial intelligence (AI) field to comprehend the thesis fully. As ML and AI tools are increasingly integrated into research and the engineering industry, we, as structural engineers, need to be familiar with these advancements.

1.1 Herøy FoU

In the municipality of Herøy in northern Norway stands a post-tensioned bridge, constructed in 1966, that serves as connector for the island community's roads. This bridge faced a significant challenge during its construction. A critical loss of pretension occurred 12 hours after the cables were secured, leading to crushing at the anchorage points for the post-tensioning. This incident compromised the structural integrity of the bridge [1].

Although the bridge remains in use today, it has sustained substantial damage and is scheduled for replacement in 2024 [2]. In 2017, a consulting firm was commissioned by Statens Vegvesen (SVV) to assess the chloride levels in the bridge and determine the potential for reinforcement corrosion. The study revealed high concentrations of chlorides, indicating a significant risk of corrosion. Based on there findings, it was determined that further interventions were necessary. Subsequent investigations reaffirmed the need for the bridge's replacement.

Currently, the bridge is under examination in a collaborative effort between NTNU, SINTEF, The Arctic University of Norway (UiT), Nordland Fylkeskommune (NFK), and SVV. This team is using the bridge as a case study with the aim "to improve current engineering practice in structural assessment of existing structures by integrating the latest scientific advancements" [1]. As a part of the structural health monitoring initiative, the bridge has been outfitted with six sensors (as of September 2023) that track temperature and crack widths [3]. One of the primary objectives of the Herøy FoU project is to amass comprehensive data regarding the bridge's current condition [1]. Our examination revealed that certain sensors relay data as frequently as 20 times per second.

1.2 Sensor Installation

As part of the FoU project, strain gauges were installed in September 2020. The placement of these gauges is detailed in the private *Report on Instrumentation for Nordland Fylkeskommune*, which is also seen in Figure 1. The report further provides information on the installation process and specifications of the measuring equipment [3]

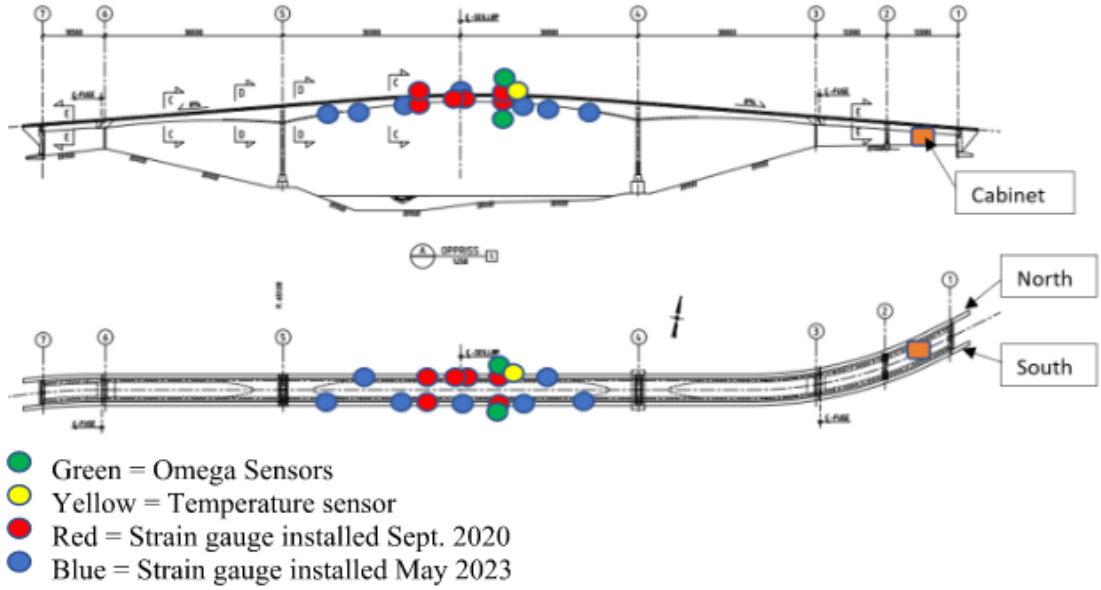


Figure 1: Sensor Locations [3]

The gauges have been installed on Herøysund Bridge to monitor strain and observe crack progression in the bridge's middle span. A strain gauge is a type of resistor used to measure the strain of an object. It operates by detecting changes in resistance as it stretches or compresses along with the object to which it is attached. Currently, sixteen sensors, including 13 strain gauges, two omega sensors, and a PT100 temperature element, are monitoring the structural health of the bridge. These gauges are attached to composite bands affixed to the underside of the bridge, measuring strain longitudinally along the bridge [3]. This monitoring initiative is a collaboration between Aas-Jakobsen, Nordland Fylkeskommune (NFK), and Statens Vegvesen, supporting sensor placement and equipment setup [3]. It is important to note that 7 out of the 13 strain gauges were installed in May 2023, leading to a non-coherent dataset for periods before and after May 2023.

The Omega N sensor, used for measuring crack displacement, was noted as broken during its installation in May 2023. Figure 1 provides an overview of the sensor locations, and Table 1 details the installation dates and types of sensors.

Sensor name	Installation	Type of Sensor	Status
Point_5_S	Sep-20	Strain gauge	ok
Point_3_N	Sep-20	Strain gauge	ok
Point_4_S	Sep-20	Strain gauge	ok
Point_1_N	Sep-20	Strain gauge	ok
Point_2_N	Sep-20	Strain gauge	ok
Point_6_S	Sep-20	Strain gauge	ok
Omega_N	Sep-20	Crack displacement	Broken-03.23
Omega_S	Sep-20	Crack displacement	ok
PT100_Temperature	Sep-20	Temperature	ok
Point_S+2000	May-23	Strain gauge	ok
Point_N+1500	May-23	Strain gauge	ok
Point_S+1000	May-23	Strain gauge	ok
Point_S+0	May-23	Strain gauge	ok
Point_S-1000	May-23	Strain gauge	ok
Point_N-1500	May-23	Strain gauge	ok
Point_S-2000	May-23	Strain gauge	ok

Table 1: Revised sensor list with updated status [3]

The data from the sensors are rapidly uploaded to an online cloud, enabling real-time access for all involved parties of the FoU project. As indicated in Table 1, many of the sensors were recently installed (May 2023). For simplicity in our research, we have excluded data from sensors installed in May 2023, as we aim to analyze at least two seasons' worth of data before including a sensor in our report. This methodology is further discussed in Chapter 2.

1.3 Machine learning

Structural engineering revolves around the assessment and planning of structures that bear loads. When dealing with intricate structural systems subjected to extreme conditions and characterized by nonlinear behaviors, the conventional methods of structural analysis and design demand an calibration process and can become overly complex for practical use. In such scenarios, ML offers an appealing alternative, as it makes streamline the process possible and reduce the time and effort required [4].

ML is a class of AI witch is learning computers to make predictions by leveraging accessible datasets and algorithms. The algorithms provides computer systems the ability to learn and improve themselves on datasets rather than being programmed explicitly. We can in other words say that ML needs dataset and results, before we evaluate it to figure out which rules that made the results the way it is [5]. ML can be categorized into three primary groups depending on the learning methodology: supervised learning, unsupervised learning, and reinforcement learning [4].

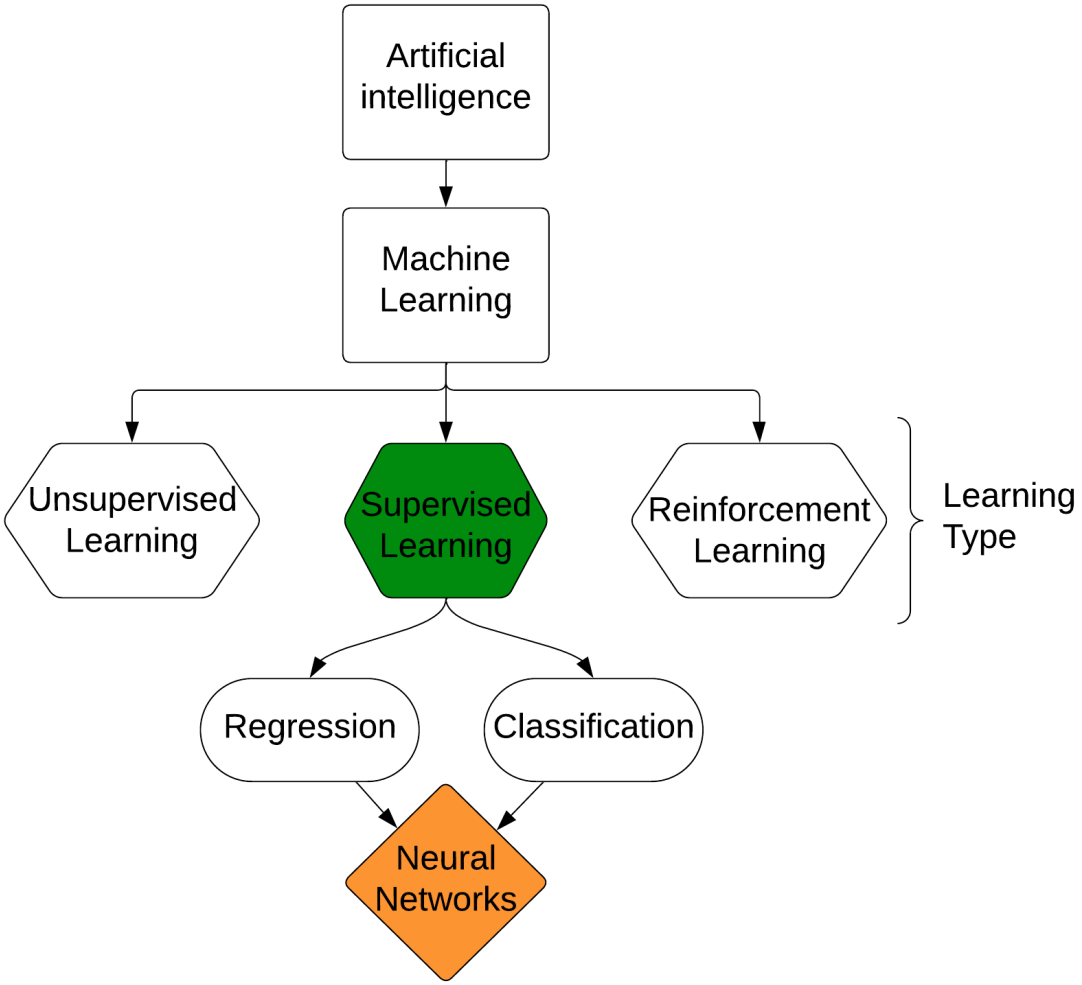


Figure 2: ML algorithms

We can observe various ML categories in Figure 2, with supervised learning highlighted in green. Chapter 1.4 delves deeper into the methodologies historically employed in the field of structural engineering, identifying Neural Networks (NNs) as the main approach. One notable advantage of NNs, compared to other ML categories, is the availability of open-source software and extensive documentation. For structural engineers who may not have vast experience in experienced data science techniques, these resources are important. They significantly ease the initial inroad into the intersection of AI and structural engineering.

1.3.1 Supervised learning

As shown in Figure 2, supervised learning is one of the three overall categories within ML. Stephen Marsland suggests that supervised learning focuses more on the dataset itself rather than the methods applied to it [6]. In supervised learning, the dataset must include the correct responses [6]. Therefore, it's crucial to structure or modify the dataset to ensure it includes targets that align with the desired outcomes from our models. A supervised learning algorithm predicts results by creating

mathematical models that generalize the training dataset, enabling predictions of future results based on user inputs.

One outcome of the generalization performed by supervised learning algorithms is the filtering out of noise in the dataset [6]. This inherent feature is particularly beneficial for interpretation, especially when dealing with time series data (As explained in Chapter 1.3.4).

Regression and classification are subcategories within supervised learning, each with distinct characteristics; regression algorithms aim to identify similarities in data, whereas classification algorithms attempt to determine the boundaries of different categories to correctly assign inputs. NNs can exhibit characteristics from both regression and classification algorithms. In deep NNs, these traits often coexist within the network, rendering the algorithm more complex and less interpretable. Overall, the most crucial aspect is the chosen algorithm's ability to improve each time the model is run through the dataset [4].

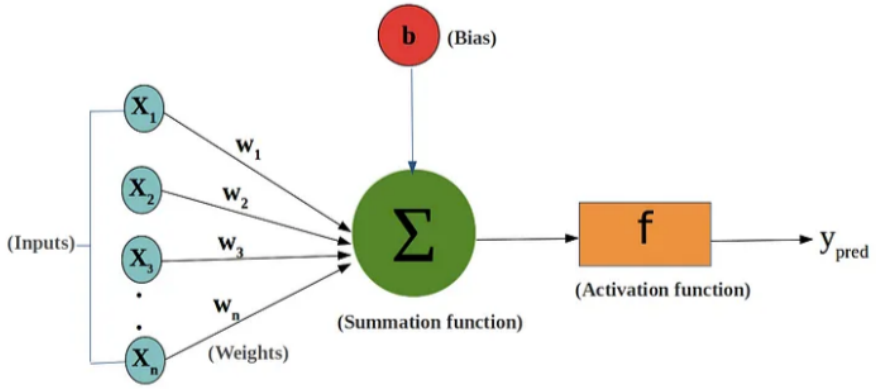


Figure 3: Components of the basic artificial neuron [7]

NN is often seen by non AI practioneers as the core of AI itself. As discussed earlier, NNs is a sub category of ML and under itself have different NNs, including Artificial Neural Networks (ANN) and Recurrent Neural Networks (RNN). NNs mimics the way the brain works with neurons and electric signaling [8]. Shared for them both is that the network is built up by neurons structured in in input layers, hidden layers and output layers. If you look at figure: 4 and think of each "circle" as a neuron, and the vertically aligned groups of circles as distinct layers within a neural network [5]. From figure 3 we see that each neuron is built up by a weight, input and a bias. The way the different neurons form a complete a network, is called Network architecture. As learning is progressing, the biases and weights are altered in order to best possible generalise the dataset. [9]. In this thesis, we aim to evaluate the performance of two distinct types of Neural Networks (NNs); the Artificial Neural Network (ANN) and the Recurrent Neural Network (RNN). While they share foundational principles, their operation and structural attributes significantly differ. Some people might say that RNNs are just a kind of ANN because they are built on a similar basic structure. However, when we compare them, it's really important to point out what makes each one special. In the next part, we'll explain what each of these types of NNs is and how they work.

1.3.2 ANN - Artificial Neural Networks

ANNs are commonly recognized as standard neural networks. They are provided with input, which travels through one or several hidden layers before the model makes a prediction at the output, thus earning the descriptive name *feedforward networks* [8], [10]. Due to their straightforward architecture compared to other neural network types, ANNs tend to be relatively computationally efficient.

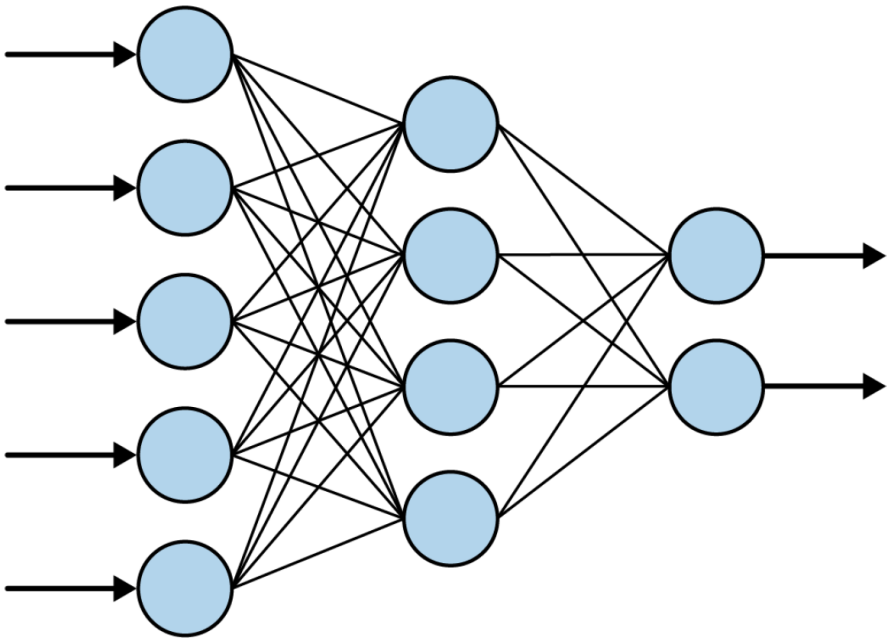


Figure 4: Typical neural network [5]

In *AI and Machine Learning for Coders* by Laurence Moroney, examples of ANN applications include image recognition, pattern recognition, and various types of regression tasks [5]. ANNs lack an inherent memory mechanism, meaning that as information moves from one neuron to the next, the information is not retained except through the trained parameters. A typical example of a neural network with dense layers could be seen in figure

1.3.3 RNN - Recurrent Neural Networks

In contrast to ANNs, RNNs possess a form of memory by retaining output information, which is then used to interpret new input. This memory is facilitated through a so-called recurrent connection. In essence, RNNs loop over one or several neurons, allowing the network to maintain information persistence see figure: 5. Unlike ANNs, RNNs are crafted to interpret sequential data, such as time series data like weather forecasting [8], [10].

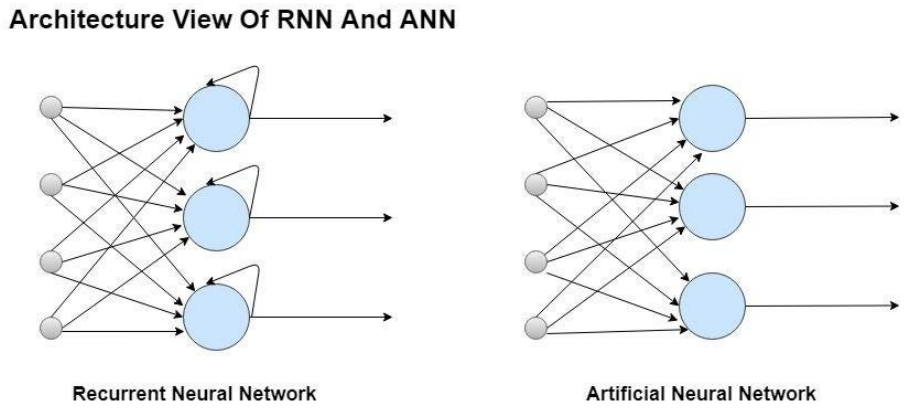


Figure 5: RNN and ANN network [11]

1.3.4 Time series data (Sequential data)

François Chollet defines a time series as "any data obtained via measurements at regular [time] intervals..." [8]. Time series data can often be visualized through plots, with time represented on the x-axis and the measured unit on the y-axis. There are several attributes used to describe time series data: Trend, Seasonality, Autocorrelation, and Noise being the primary ones.

- Trend refers to the overall direction in which the data is moving over time. For instance, when zoomed out sufficiently, a discernible pattern or tendency might emerge, indicating a general increase or decrease in the data values [5].
- Seasonality is depicted as recurring patterns or cycles occurring at regular intervals known as seasons. For example, a time series might exhibit weekly patterns such as lower activity during weekends, or annual patterns like a dip in activity during summer breaks [5].
- Autocorrelation represents the degree of similarity between a given time series and a lagged version of itself over successive time intervals. It reflects the predictability of the time series based on its past values. For instance, a pattern of decline followed by a sudden rise that repeats over time can be a manifestation of autocorrelation. Measuring autocorrelation can provide insights into the predictability and repetitive patterns inherent in the time series [5].
- Noise refers to the random, unstructured fluctuations present in the time series data. These irregular changes contribute to unpredictability, potentially obscuring other significant attributes like trends, seasonality, and autocorrelation. While noise might not entirely nullify the other attributes, it can hinder visual interpretation and analysis [5].

Time series datasets are typically used to train models for the purpose of forecasting future values, enabling predictions based on historical data [5], [8].

1.3.5 TensorFlow and Keras

TensorFlow is by Moroney Laurence defined as an open source tool for implementing algorithms needed for training ML models [5, p. 7]. TensorFlow was introduced in 2015, and is one of the most popular open source platforms for developing ML models [4]. ML algorithms could be hard to understand for new practitioners, and using TensorFlow gives for example structural engineers a lower entry level for implementing ML in their own field. The platform is open source and allows user contributions, making it developing progressively and it is originally developed by the Google Brian team. The open source nature of the platform benefits Google as well, since contributions from outside is directly improving the much used tool in Google.

Keras is an user friendly high-level API that allows users to access the TensorFlow library through languages such as Python. A high-level API is a interface that let users access underlying functionalities through a standardized layout. TensorFlow allows users to code the ML implementation directly, but Keras is a way of accessing previously tested ML paradigms and applying them on your own dataset. In this way could the programmer write understandable code faster, and focus on the dataset (which is many ways the most important aspect) and the outcome of the models [8].

TensorFlow uses the computers CPU or GPU to calculate models and train the ML model. The calculating demand is heavily affected on the data size as well as the model build. When using TensorFlow you need to compile the model to configure the model for training. When configuring the model it requires you to specify at least two important parameters: the optimizer and the loss function. The loss function allows the machine learning model to evaluate how well it is making predictions. It does this by comparing the model's predicted output with the actual output it should have produced. The difference between these outputs is the loss, which the model aims to minimize. The optimizer, on the other hand, is the tool that adjusts the model's parameters, like the weights, in response to the output of the loss function. It uses mathematical optimization techniques to steer the model towards more accurate predictions. The optimizer takes into account the current loss and adjusts the model to reduce this loss over time [5].

According to Chollet, there are 4 common NNs architectures; Convolutional networks, Transformers, *densely connected networks* and *RNNs* [8, p. 436]. The two latter ones are the one we are using in this project since they is practical to use for time series data. As previously mentioned, the structure of the dataset is having a major role in which architecture to use.

ANNs layer could be set using the Keras Model class and Keras Layers API. There are are many popular layers, but for normal ANNs are normally Core layers used i.e. dense layers [12]. A dense layer signifies a fully interconnected set of neurons, as seen in figure 4, where every neuron connects to each other in the subsequent layer. This layers is the most common layer in densely connected networks. Another layer type is a normalizing layers, that scales the input around zero and uses a standard deviation as 1, hence the name normalization. Normalization layer are useful since they can increase the learning rate for dataset, and normalization layers are de facto

in most TensorFlow implementations [5].

Some RNN layes are LSTM- or GRU. LSTM enables context to be maintained not just from step to step, but trough an entire sequence of steps. GRU does the same as LSTM but it has a simpler memory. In essence, RNN layers such as LSTM and GRU are instrumental for sequential data processing, with LSTM handling long-term dependencies and GRU offering a more simplified memory function [5].

When a neural network is being trained dropout can reduce overfitting. Especially in large datasets there is a risk of the neurons being overspecialized leading to overfitting. While training we can use dropout to temporally block a random number of neurons to reduce the chance of the neurons becoming overspecialized. Dropout during neural network training serves as a crucial technique to prevent overfitting by randomly disabling neurons see figure 6.

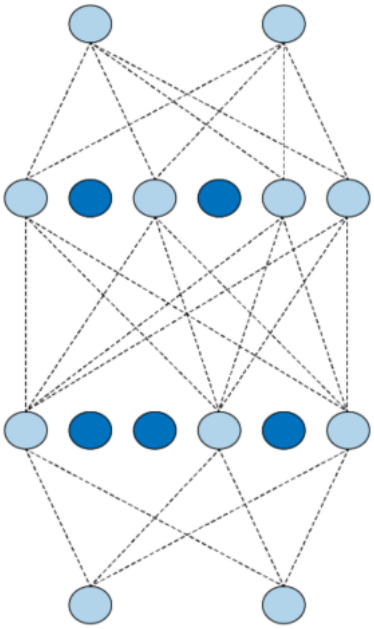


Figure 6: Neural network whit dropout [5]

KerasTuner is a hyperparameter tuning library that allows for the optimization of machine learning models built with Keras. You can define the parameters and run, where the hyperparameters are specified inline with model code. It supports various search algorithms, including Bayesian Optimization, Hyperband, and Random Search, to navigate through the search space efficiently and find a optimal set of hyperparameters. KerasTuner is also flexible, allowing researchers to add custom search algorithms and advanced optimization methods tailored to specific problems [13].

1.4 Literature Review

In this section, we will discuss the findings from the article *Machine Learning for Structural Engineering: A State-of-the-Art Review* authored by Associate Professor

Huu-Tai Thai from the University of Melbourne [4]. This 2022 paper is relevant to our thesis as it provides an overview of machine learning in the context of structural engineering and offers guidance on potential directions to explore. The article also reviews tools for AI implementation in Python. Our commitment to using open-source software and code influences our literature selection process.

We are also looking into the book *AI and Machine Learning for Coders*, which is giving a beginner introduction to ANN using TensorFlow and Keras API [5]. When combined with Associate Professor David Morin's PowerPoint on artificial neural networks, these resources form a foundation for analyzing the data form Herøy FoU[14].

1.5 Machine Learning for Structural Engineering: A State-of-the-Art Review

Thai notes that despite the growing interest in machine learning (ML) within structural engineering, its practical use was still emerging in 2022. This report aims to offer a detailed review of ML applications in structural engineering. Thai outlines three main steps for developing an ML model: preparing the database, training the model, and evaluating its performance.The dataset should be randomly divided, with a larger part for training and a smaller for testing, according to Thai.

In terms of ML algorithms, Thai points out the necessity of selecting the right model for effective learning and recommends testing various algorithms. The model's performance should then be assessed using the testing dataset [4].

Thai describes Artificial Neural Networks (ANN) as a combination of regression and classification, both under supervised learning in ML. Supervised learning involves labeled datasets, where each input is tagged with labels denoting characteristics such as strain or temperature from sensors. Notably, Thai states that 56% of structural engineering applications have used Neural Networks (NN), particularly ANN, which is the most documented technique [4].

The article also reviews different structural engineering projects employing ML. It highlights that ANN has been applied across all structural member categories: Beams, Slabs, Walls, and Joints. ANN was predominantly used in studies focusing on bond strength for reinforced concrete, including research on steel bars in concrete and UHPC, and corroded steel bars in concrete. Moreover, ANN is frequently used for serviceability and deflection prediction, similar to the Herøy FoU project dataset [4].

Thai discusses Structural Health Monitoring (SHM), noting two main techniques: image-based and vibration-based. The Herøysund FoU project, although focused on monitoring crack widths and temperature, deals with time-series data akin to vibration-based methods. However, Thai observes that SHM reviews, including studies by Majdi Flah et al., primarily use ANN and image-based methods, such as in a bridge vibration monitoring project. This project, closely related to our thesis, differs by measuring vibration instead of strain, like in the Herøysund FoU project

[15][16].

In his conclusion, Thai emphasizes the significance of Artificial Neural Networks (ANN) in the field, stating, "ANN is that it can express or interpret in terms of empirical equations to be implemented in design codes" [4]. This highlights a major benefit of ANNs in potentially standardizing findings for practical application.

Thai identifies several challenges, including the selection of suitable models and the "black box" nature of complex ML algorithms, which can hinder understanding and integrity in the field. A prevalent view in the ML community is to keep the complexity of ANNs as minimal as possible while maintaining accuracy [17].

1.5.1 AI and Machine Learning for Coders

The book *AI and Machine Learning for Coders*, authored by Laurence Moroney, the AI Advocacy Lead at Google, provides a comprehensive introduction to TensorFlow and its utilization through the Keras Python API. Moroney, known for his leadership role in the Google Brain team, has significantly contributed to the development of TensorFlow, a widely-used open-source library created by Google. This book, particularly chapters 1, 9, 10, and 20, is relevant and beneficial for our thesis.

1.6 Research Problem

In order to conduct a comprehensive assessment of the structural health and integrity of the Herøysund Bridge, we have formulated the following problem statement:

> "Which model, standard ANNs or RNNs, is more effective for predicting the specific strain development in the Herøysund Bridge?"

Based on the results presented in chapter 3, we want to discuss the problem statement in the discussion chapter of this project thesis.

2 Methodology

This chapter outlines the methods and procedures employed to study the application of ML in concrete bridge Herøysund bridge. The chapter also includes a critique of our methodology.

2.1 Definitions

In this section we are going define some ML terms. The list could be used as a look up table while reading this project thesis as well ass describing witch ML steps we have used in our code provided on GitHub. The repository for our project is available at https://github.com/TheCodingBadger/concrete-machine-learning/tree/main The definitions are presented in chronological order, reflecting the sequence in which they were implemented in our code or gained clarity in our understanding.

Activation Function A function that lies within each neuron, for example the 'relu' function. The 'relu' activation function lets only positive values pass the neuron [5].

Artificial Intelligence is an information technology that adjusts its own activity, appearing to be intelligent. It is a scientific field that combines data techniques, logistics, mathematics, psychology, and neuroscience [18].

Batch Training data can be sliced into pieces (batches) of different batch-sizes. In Keras, the *.timeseries_dataset_from_array* objects are defined with a *batch_size*, representing the quantity of time series within a batch [19].

Delay Targets for a sequence are moved to the future with a certain length in reference to the samples [8].

Epochs One epoch is the sum of all batches. The model iterates over the entire dataset for a specified number of batches, making up the total dataset [19].

GRU layer is another variant of recurrent neural network layers, designed to handle sequential data efficiently. Similar to LSTM, GRU is engineered to solve the vanishing gradient problem found in standard RNNs, but it does so with a simplified gating mechanism. GRU combines the input and forget gates into a single update gate and includes a reset gate. This streamlined architecture results in fewer parameters than LSTM, enhancing its computational efficiency [20].

Hyperparameters Parameters set before the training process, often seen as the NN architecture. Influencing the performance of the ML [14].

Learning Rate Decay Describes how the learning rate of the optimizer changes over time [21].

LSTM layer is a type of recurrent neural network (RNN) layer. Its defining feature is the implementation of a unique gating mechanism, which includes input, output, and forget gates. This structure allows the LSTM to effectively address the vanishing gradient problem, a common challenge in traditional RNNs. By managing the flow of information through these gates, the LSTM can retain important long-term dependencies in the data, making it particularly suitable for tasks that require the understanding and retention of information over extended time periods [20].

Machine Learning is a specialization of artificial intelligence where statistical methods are used to find patterns in large datasets. The machine learns from data rather than being explicitly programmed [22].

Mean Absolute Error (MAE) Calculates the average absolute difference between labels and predicted values [23].

Mean Squared Error (MSE) is a loss function in the Keras API that computes the mean squared error between true and predicted values [23].

Metaparameters Higher-level parameters that guide the continuous adjustment of the constructed NN. Examples are optimizers, batch size, number of epochs, and validation monitoring function [14].

Momentum Used for momentum accumulator weights in the optimizer [24].

Optimizer Adam Adam is an optimizer in the Keras API, a gradient descent method based on adaptive estimation of first and second-order moments. Based on values, previous guess, and loss on that guess, uses advanced mathematics in order to adjust the neurons' weights [5][24].

Overfitting Occurs when a model learns noise and inaccuracies, losing generalization [6].

R2 score The R2 score measures the accuracy of the fitted regression line in comparison to the actual data. A score of 1.0 represents the ideal scenario, where the predictive variables perfectly explain the variation in the target variable. While a score of 0.0 suggests that the predictors fail to account for any variation in the target. If the score falls into negative territory, indicating a model's performance that is worse than random prediction [25].

Sample and target Samples are data points associated with certain target data points [8].

Sequence Length The length of a time series object.

Testing Data The data is kept separate from training data and is used to evaluate model performance by comparing predictions with target outputs [6].

Training Data The data used to train the machine learning model [6].

Validation Data The data is crucial for measuring overfitting during training. Without validation data, it is challenging to detect overfitting [6].

2.2 Selection for method

For our project, choosing the right ML and AI approach was a crucial and challenging part of our work. The method selected significantly influences the outcomes of what could be described as "advanced guessing" inherent in machine learning. This section will discuss how we selected a method that fit our problem stated in Chapter 1.6. In this subsection we want to investigate what data we are dealing with and based on that choose a procedure of implementation. The procedure of implementation should decide how we are going to do the data processing and ML.

2.2.1 Inspection of conducted data collection

In research, the process of data collection is crucial, as it gathers the necessary information to accurately reflect the subject under investigation [26, p. 29]. For ML models, the dataset serves as a pivotal foundation. Generally, the model's capacity to learn and enhance its performance increases with the volume of data fed into it [4, p. 457].

The initial sensor data from Herøy Bridge were stored in a hierarchical folder structure consisting of MATLAB files. This organization was primarily chronological, with top-level folders designated for each year: 2020, 2021, 2022, and 2023. The data were further subdivided into monthly folders, which were then broken down into daily subfolders. Corresponding to each day, there were 24 MATLAB files, each encapsulating the hourly sensor data for that day. The data within each MATLAB file were arrayed according to time, yet they were not formatted into a regular table structure. The print statement below shows the format of the data found in the MATLAB files when inspecting them through the code **print_matlab.py**:

```
'__header__': b'MATLAB 5.0 MAT-file, Platform: PCWIN64, Created on: Sat Sep 16
↪  12:18:14 2023', 'Data': array([[(array(['20210305032447'], dtype='<U14'),
↪  array([[(array([[0]], dtype=uint8), array([[3601.65017107]]),
↪  array([[72034]]), array([[0]], dtype=uint8), array([[44260.14221065]]),
↪  array([[50]], dtype=uint8), array(['20210305032447'], dtype='<U14'))]],
              dtype=[('t_0', 'O'), ('t_end', 'O'), ('num_t', 'O'), ('broken',
              ↪  'O'), ('T0', 'O'), ('dt', 'O'), ('T0_datestr', 'O')]),
              ↪  array(['E:\\Herøysund\\Files - Original\\2021\\03\\Heroysund
              ↪  Data_1359.BIN'],
              dtype='<U61'), array([[(array(['Point_1_N'], dtype='<U9'),
              ↪  array([[-47.59076309, -47.51504898, -47.69351578, ...,
              ↪  -43.36167145,
                     -43.59962845, -43.17779922]]), array(['µm/m'],
                     ↪  dtype='<U4')),
                (array(['Point_2_N'], dtype='<U9'), array([[-73.84140015,
                ↪  -73.70079041, -73.80895233, ..., -75.26371002,
                     -74.85269928, -74.93923187]]), array(['µm/m'],
                     ↪  dtype='<U4'))
                (array(['Point_3_N'], dtype='<U9'), array([[-13.37409878,
                ↪  -12.73594952, -13.08206463, ..., -12.38983536,
                     -11.91933537, -11.6921978 ]]), array(['µm/m'],
                     ↪  dtype='<U4'))
```

```
(array(['Point_4_S'], dtype='<U9'), array([[-151.97677612,
↪  -151.93893433, -151.63067627, ..., -152.58789062,
       -151.97677612, -151.98219299]]), array(['µm/m'],
       ↪  dtype='<U4'))
(array(['Point_5_S'], dtype='<U9'), array([[-154.13999939,
↪  -154.41581726, -154.01560974, ..., -154.6645813 ,
       -154.70243835, -154.96743774]]), array(['µm/m'],
       ↪  dtype='<U4'))
(array(['Point_6_S'], dtype='<U9'), array([[-141.43109131,
↪  -141.78260803, -141.12823486, ..., -142.41534424,
       -141.92321777, -141.83668518]]), array(['µm/m'],
       ↪  dtype='<U4'))
(array(['Omega_N'], dtype='<U7'), array([[0.3340303 , 0.33409813,
↪  0.3340739 , ..., 0.34323564, 0.34318557,
       0.34314036]]), array(['mm'], dtype='<U2'))
(array(['Omega_S'], dtype='<U7'), array([[-0.03188037, -0.0317625
↪  , -0.03184324, ..., -0.03056441,
       -0.03044007, -0.0305418 ]]), array(['mm'], dtype='<U2'))
(array(['PT100_Temperature'], dtype='<U17'), array([[0.25431314,
↪  0.25516087, 0.25558472, ..., 0.02712674, 0.02670288,
       0.02627902]]), array(['°C'], dtype='<U2'))]],
```

The provided data exhibit inconsistency across different days. We have documented the file sizes for each month in Table 2. The variation in file sizes can be attributed to multiple factors. Starting from December 2020, the measurement frequency was set at 20 measurements per second. However, the initial month shows significantly larger file sizes, primarily because of an initially higher measurement frequency. As Cantero indicated, this frequency was substantially reduced and later increased after two months. Additionally, some months show lower data volumes, stemming from improper recording of measurements, which leads to gaps in the data.

In Figure 7, we have a visual representation of the strain displacement [µm/m] values for the first minute from an example MATLAB file. This visualization is notable because it captures a span of time when the bridge was disturbed by a vehicle around 28 second on the x-axis.

Table 2: Data Sizes for Each Month and Year in the MATLAB File Structure

Year	Month	Size (MB)	Year	Month	Size (MB)
2020	09	9011.975388	2022	03	1134.255688
2020	10	53.439971	2022	04	1098.766762
2020	11	54.765921	2022	05	1140.859810
2020	12	1114.586146	2022	06	1111.991590
2021	01	1120.790736	2022	07	1130.374326
2021	02	1010.500231	2022	08	1118.300744
2021	03	1145.712627	2022	09	1068.227001
2021	04	1112.143288	2022	10	1102.790491
2021	05	1144.569469	2022	11	1067.492497
2021	06	183.881099	2023	01	304.743116
2021	07	1083.360832	2023	02	717.250480
2021	08	1118.357994	2023	03	1159.371735
2021	09	1052.616550	2023	04	1089.889944
2021	10	1086.883708	2023	05	1213.251089
2021	11	1081.461379	2023	06	1946.156866
2021	12	1126.311034	2023	07	1984.823362
2022	01	1160.100415	2023	08	1965.667456
2022	02	1039.499219	2023	09	13.763291

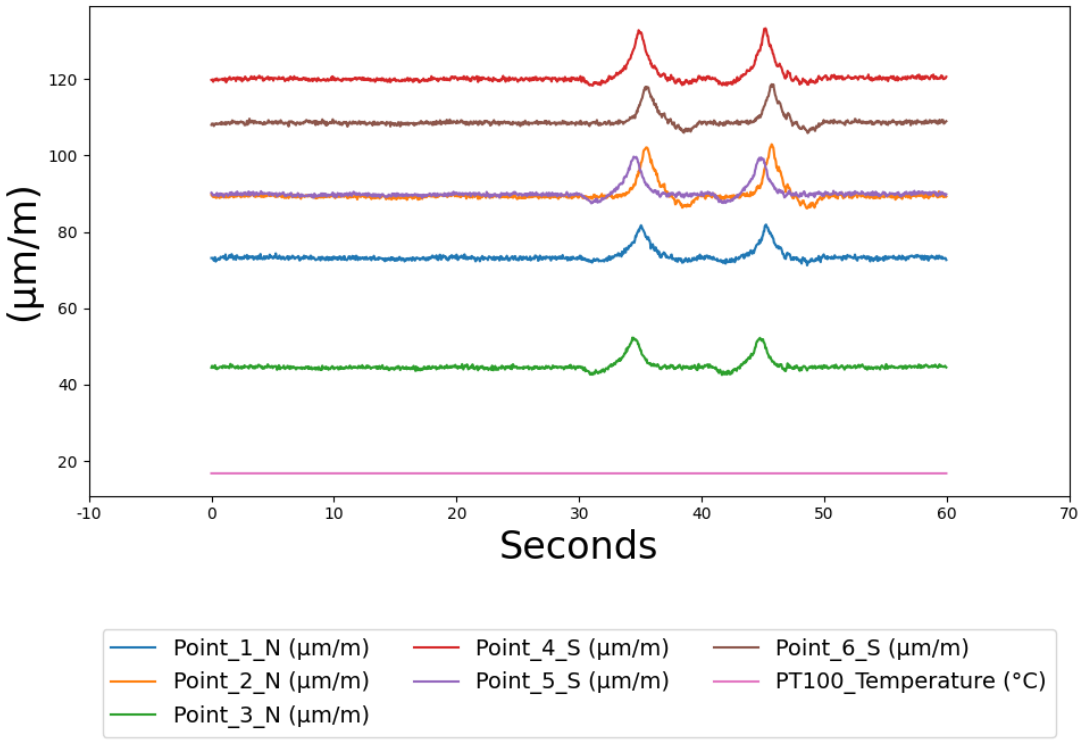


Figure 7: Points, 1min, Car

The data we have encompasses a range of measurement types. While analyzing the files, we discovered numerous gaps in the dataset. Graphical representation of the

data, particularly in the *date since first measurement* column that we incorporated, unveiled pronounced fluctuations (As we can see in figure 8).

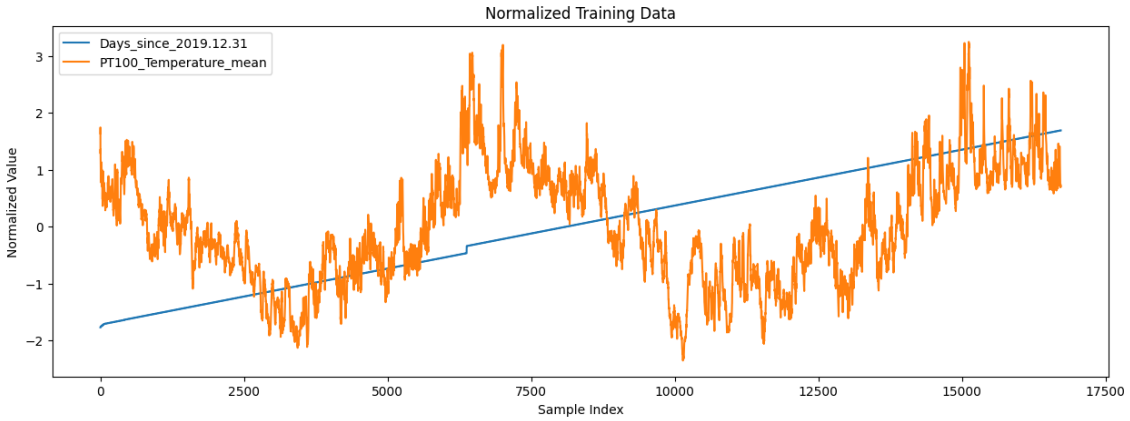


Figure 8: Imput data

The sudden rise in *date since first measurement* signify periods, which could extend to whole days, where measurements are missing. Conversations with Cantero clarified that the wholes in the dataset was due to the integration of new sensors, an event elaborated on in Chapter 1.2. Concurrent with these updates, the Omega_N sensor failed, leading to a lack of data capture from this sensor beyond May 2023. Further on, Cantero were in knowledge of other smaller abnormalities in the dataset.

2.2.2 Procedure of implementation

While being new to machine learning, our initial challenge was understanding the necessary hardware and software requirements. Our direction became clearer after viewing Morin's presentation and conducting a literature review. We found that TensorFlow, with its suite of optimized mathematical models and pre-built models, was a suitable tool for our project. Its ability to run on standard desktop computers, like the ones we were using, was a significant advantage. The initial five weeks were dedicated to familiarizing ourselves with the necessary Python libraries, understanding TensorFlow's capabilities, and comprehending other essential packages for implementing ML in our project.

Engen recommended using GitHub for code collaboration, which would serve as both a practical tool for managing our coding work. GitHub's ability to track progress ensured that we had a detailed version control and a clear view of our project's evolution. We adopted the Integration-Manager Workflow for our repository management, as outlined by Scott Chacon and Ben Straub in *Pro Git* [27]. The repository for our project is available at https://github.com/TheCodingBadger/concrete-machine-learning/tree/main. It includes timestamps, reflecting the contributions made to the project. This repository acts as a supplementary appendix to our assignment, allowing readers to examine our progress and the specifics of our work.

We divided the project into two sequential phases: data processing and the application of machine learning techniques. Our primary goal during the data processing phase was to convert MATLAB files into the more efficient Apache Parquet format [28]. We aimed to restructure the data into a comprehensible table format and extend it with additional labels for the ML phase. Upon completing data processing, we utilized the Keras API to apply machine learning techniques to our dataset, thereby developing ML models for our project. Both team members were involved in each step, opting for a collaborative approach to enhance our learning, particularly with an eye towards our upcoming master's thesis on machine learning and concrete structures.

Engen introduced us to a conceptual model to visualize our process: We began with what can be termed as 'bronze' data. This raw data required sorting and modification to transform into 'silver' data, which has a defined structure and is ready for further analysis. Utilizing ML methods on the silver data, we aimed to produce 'gold' data – the final, synthesized information of interest and value derived from the initial raw data. This process, along with its outcomes, is visually represented in Figure 9.

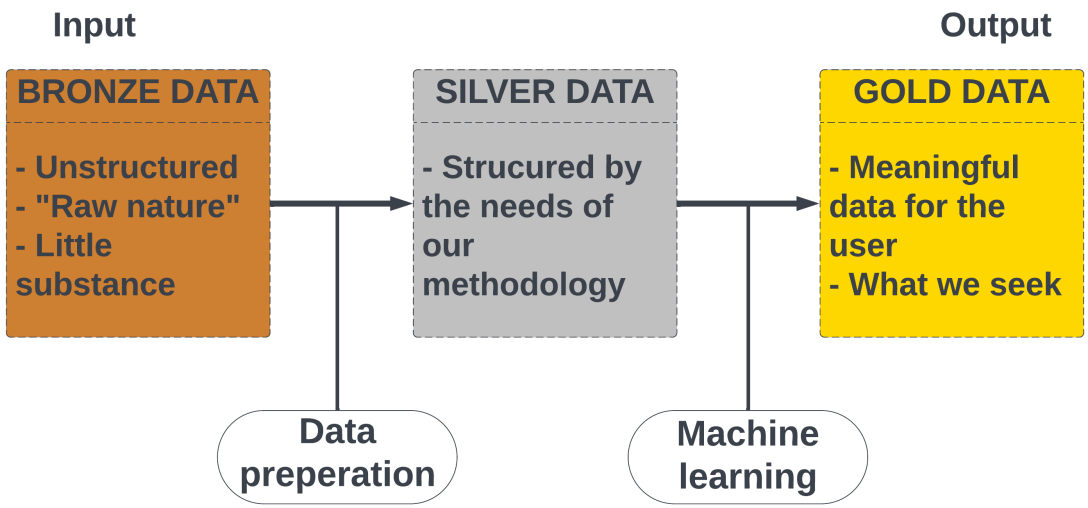


Figure 9: Transformation Process: From Raw Data to Actionable Insights

The transformation process above emphasize the importance of understanding the dataset within the ML field; data preparation is done first in order to choose the correct ML approach. The sections that follow provide descriptions of the data preparation and ML processes conducted in our project.

2.3 Data processing

In data science and machine learning, the size of the data and the speed at which it can be transferred are critical factors in the efficiency of computational processes. Smaller file sizes mean faster data manipulation and reduced processing times, a

significant advantage when dealing with complex models like NNs that require repetitive loading of vast datasets. Enhanced transfer speeds ensure that data retrieval and integration into the model are seamless, a crucial aspect during the iterative training cycles typical in machine learning.

The original dataset contained 44 GB of data across thousands of MATLAB files. To achieve better storage efficiency and increase the data transfer speed, we transitioned the MATLAB files into Apache Parquet file format. The Parquet file format was preferable because of it's open source nature and wide usage in the ML community [29]. We also introduced columns to exhibit dates and hours since 31.12.2019. This made it possible for the potential algorithm to be trained as a time series data, evaluating temporal variations in the measurements. In a data averaging process, we explored different averages: by minute, by hour and daily averages. In our inspection, the daily averages had an insufficient amount of date leading to less adequate learning by the ML model. The minute files were to comprehensive, making the ML model learn fine patterns that were out of the scope of our problem statement. The minute average files contained to much data for to be able to run tuners and RNN models on normal computers. We ended up using hourly averaged files with one row per hour in the dataset, stored as hours since 2019.12.31. The resulting file sizes was 2.35 MB for the hourly averaged file, in stark contrast to the previous arrangement where data was dispersed across thousands of files consuming 44 GB. The averaging also filtered out variability caused by transient events, such as traffic, helping the model to learn from the structural behavior of the bridge rather than outer impact.

We accomplished this task through the utilization of three Python scripts (The datasets and code utilized in this project, which were mentioned previously, are available in the GitHub repository at: https://github.com/TheCodingBadger/concrete-machine-learning.git.):

1. A script dedicated to converting all MATLAB files into a Parquet file format.
2. A script for appending a column indicating the number of hours since 31.12.2019, along with the filename, to each Parquet file.
3. A script for aggregating the data, averaging it to one row per hour, and subsequently storing one Parquet file containing the complete dataset.

The graphical representations of this compressed data can be viewed in figures 10, 11 and 12. Notably, gaps are observed in the measurement timeline within these figures, attributable to periods when the sensors failed to capture data over certain days or months.

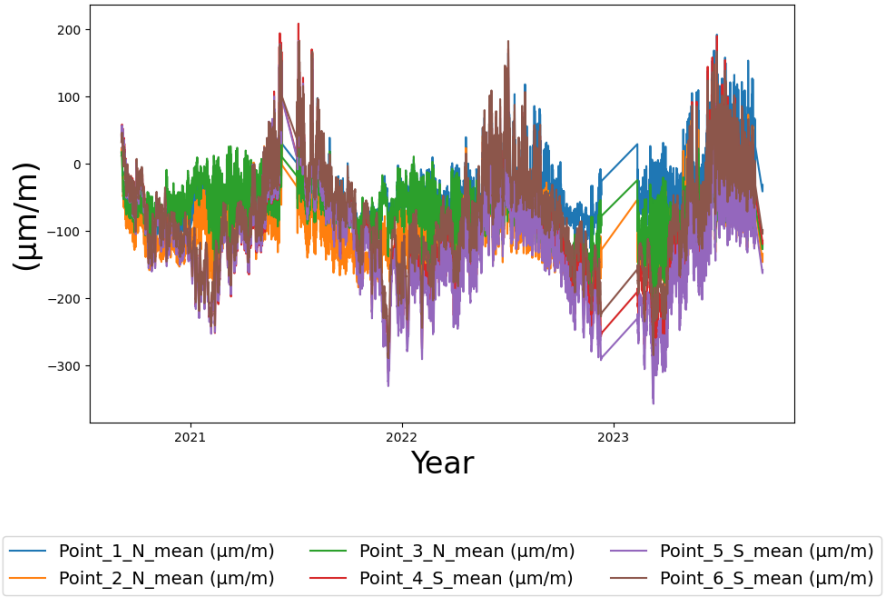


Figure 10: Points, average per minute

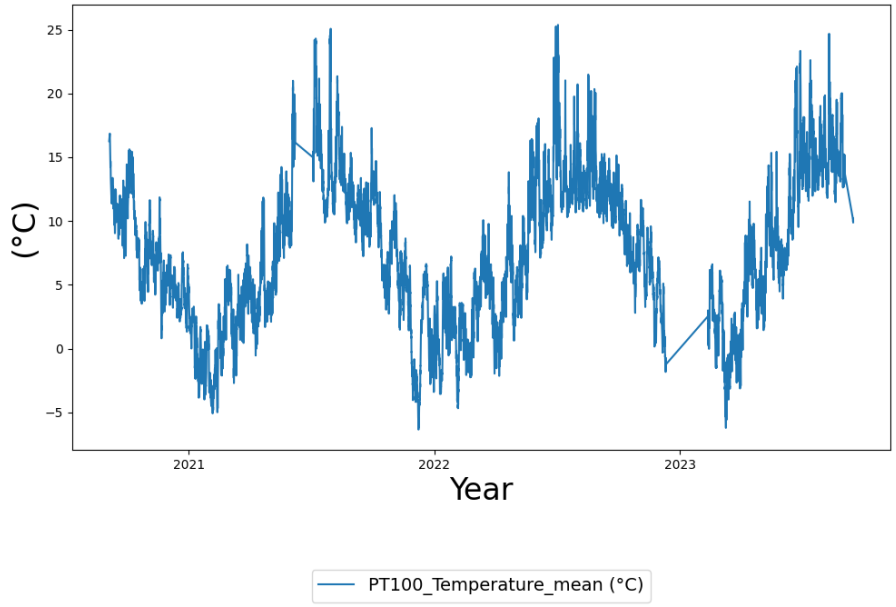


Figure 11: Temperature, average per minute

In Figure 12, an anomaly is observed where the crack displacement briefly records a value of 10^6 for the Omega_N sensor and later, the same anomaly is noticed for the Omega_S sensor. While we can attribute this unusual occurrence to the malfunction of the Omega_N sensor (As seen in table 1), the reason for the Omega_S sensor exhibiting a similar behavior later on remains unexplained.

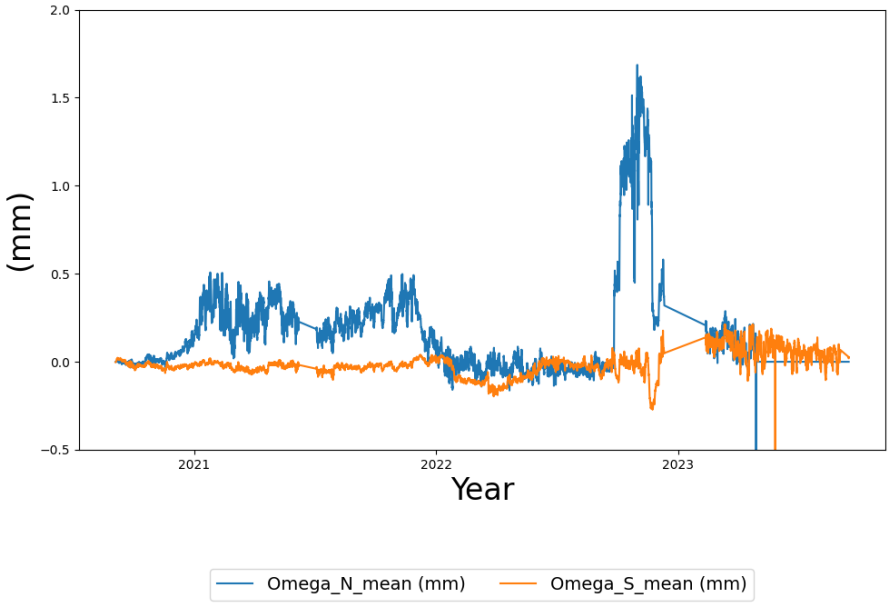


Figure 12: Omega, averaged values per minute

Before ML was applied, the data had to be split into training-, validation- and testing sets. Chollet recommends using a holdout validation set when time encountered dataset is adequately large [8]. Keeping validation- and testing datasets respectively after the training dataset is common practise. The split will affect the learning, but inspecting the dataset we see that is changes relatively much in the later part of the time series. This could speak for using a split with a bigger part of training data, since we want it to catch these last tendencies as well. We will go further on with the split for 70/15/15 for training, validation and testing respectively. The splitting was done by the following code:

```
def split_data(dataframe, test_size=0.15, valid_size=0.15): # Split the data into
↪  train, validation, and test sets without shuffling.
    train_end = int((1 - (test_size + valid_size)) * len(dataframe)) # Calculate
    ↪  index for end of training set
    valid_end = train_end + int(valid_size * len(dataframe)) # Calculate index
    ↪  for end of validation set

    train_df = dataframe.iloc[:train_end]
    valid_df = dataframe.iloc[train_end:valid_end]
    test_df = dataframe.iloc[valid_end:]

    return train_df, valid_df, test_df # train_df: The training set as a Pandas
    ↪  DataFrame, valid_df: The validation set as a Pandas DataFrame, test_df:
    ↪  The test set as a Pandas DataFrame

def separate_labels(data_df, label_columns): # Separate the labels from the
↪  features.

    labels = data_df[label_columns].astype('float32') # Convert the labels to
    ↪  float32
```

```
    features = data_df.drop(columns=label_columns) # Drop the label columns from
    ↪  the dataframe to leave only the features

    return features, labels # features: A Pandas DataFrame containing only the
    ↪  feature columns, labels: A Pandas DataFrame containing only the label
    ↪  columns

label_columns = [ # Define the columns that are considered labels in your
↪  dataset
    'Point_1_N_mean', 'Point_2_N_mean', 'Point_3_N_mean',
    'Point_4_S_mean', 'Point_5_S_mean', 'Point_6_S_mean',
    'Omega_S_mean'
]

train_df, valid_df, test_df = split_data(data_df) # Split the data into train,
↪  validation, and test sets

train_data, train_labels = separate_labels(train_df, label_columns) # Separate
↪  the features and labels for the training, validation, and test sets
valid_data, valid_labels = separate_labels(valid_df, label_columns)
test_data, test_labels = separate_labels(test_df, label_columns)
```

Normalization was done on the split dataset in order to make the ML train more efficient [5]. In the normalization, the standard and mean deviation from the training set is used on training, validation and testing data. This is done to hinder information in the validation and training dataset to leak information about them self into the normalization of the test set (So that the weights are adjusted only by the training data) [8]. Yet, for keeping all datasets at same scale, the same deviation is used at validation and training as well.

The final step before running the dataset through ML models, was defining dateset through the Keras *.timeseries_dataset_from_array*. Procedure can be found in the GitHub repository and in the Keras API docs.

2.4 Machine learning

In order to answer the problem "Which model, standard ANNs or RNNs, is more effective for predicting the specific strain development in the bridge?", we have to configure understandable models that predicts future measurements. If the result is to be used in a critical analysis, i.e. an assessment on how long the bridge will withstand, the predictions has to be validated and tested.

We decided to create two models, one ANN model and one RNN model. ANN model can be configured in a large variety of ways. In general could the following parameters be tuned:

- Number of layers *(hyperparameter)*
- Type of layer *(hyperparameter)*
- Number of neurons per layer *(hyperparameter)*

- Activation function for each layer *(hyperparameter)*
- Dropout layers and dropout rate *(hyperparameter)*
- Model optimizer and its momentum *(hyperparameter)*
- Loss function, measuring performance on training data to scale the weights *(metaparameter)*
- Metrics, measuring performance on the training and validation sets *(metaparameter)*

In order to make the experimentation less extensive, we chose to limit the vast possibility of combinations. The choices made in the model configuration is heavily inspired by the timeseries models found in *AI and Machine Learning for Coders* and *Deep Learning with Python* [8][5]. We chose to use Dense layers in the ANN model and LSTM layers in the RNN model. Both models were fitted with a dropout layer after ANN/RNN layer, in order to reduce the chance of overfitting. For the activation function in the Dense layers, we used 'relu'. LSTM neurons were not assigned an activation function. An example of the implementation in an ANN model is show in the code snippet below:

```
# Model architecture
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(2, input_shape=(sequence_length, num_features),
    ↪  activation='relu'),
    tf.keras.layers.Dropout(0.5),  # Add dropout after the first layer
    tf.keras.layers.Dense(2, activation='relu'),
    tf.keras.layers.Flatten(),  # Flatten the output sequences
    tf.keras.layers.Dense(num_output_units)
])
```

Both the ANN and RNN model is set to compile using the same compiler:

```
model.compile(
    optimizer='adam',
    loss='mse',
    metrics=['mae']
)
```

We choose to create an model that predicted one out of the 16 sensors. The reason for choosing one sensor instead of multiple sensors is due to the models tendency to learning form one sensor influencing the others sensors predictions. During trail and error we noticed the evaluation metrics where lower for predictions using all sensors see appendix A, B and C. Choosing one sensor was also lowering the computational cost of running the ML models. Therefore we choose to run one instead of multiple sensors.

2.4.1 Hyperparametric tuning

Hyperparameter tuning was done using the KerasTuner. We ran tuning only on the ANN model training on hourly averaged data. The reason for this was that the RNN model was time- and computational demanding, making it too extensive for the scope of this project thesis. The ANN tuning was less demanding, and was carried out to gain experience on using the KerasTuner and exploring ANN-model possibilities.

We configured the tuner to test models varying in amount of layers, different dropout rates in the dropout layers, varying neurons in each layer, varying learning rate and momentum in the optimisation function (Adam in our case). A code snippet of the KerasTuner is shown below:

```
    def build_model(hp):
    model = tf.keras.Sequential()

    model.add(tf.keras.layers.Dense(2))

    # Define a hyperparameter for the number of Dense layers
    num_dense_layers = hp.Int('num_dense_layers', min_value=1, max_value=2,
    ↪  step=1)

    for i in range(num_dense_layers):
        # Add a Dense layer
        model.add(tf.keras.layers.Dense(
            units=hp.Int(f'dense_units_{i}', min_value=8, max_value=32, step=8),
            activation='relu'  # Common activation function for Dense layers
        ))
        # Add a Dropout layer after each Dense layer
        model.add(tf.keras.layers.Dropout(
            rate=hp.Float(f'dropout_{i}', min_value=0.0, max_value=0.5, step=0.1)
        ))

    # Output layer
    model.add(tf.keras.layers.Dense(1))  # Adjust the number of units for your
    ↪  output layer

    # Compile the model
    model.compile(
        loss="mse",
        optimizer=tf.keras.optimizers.Adam(
            learning_rate=hp.Float('learning_rate', min_value=1e-4,
            ↪  max_value=1e-2, sampling='LOG'),
            beta_1=hp.Choice('momentum', values=[0.9, 0.7, 0.5, 0.3])),
        metrics=['mae'])
    return model

# Correct the tuner initialization
tuner = kt.RandomSearch(
    build_model,
    objective='val_mae',
    max_trials=100,
    executions_per_trial=3,
    directory='my_dir',
```

```
    project_name='Dence-Hour-Tuner',
    overwrite=True  # This will clear the previous tuning results
)
```

The variation we tested around was aligned in siumilar range as the timeseries example found in *Deep Learning with Python*. We chose to vary the number of layers outside of the input and output between one and two, with neurons randomized between 8 and 32. The dropout rate is set between 0.0 and 0.5. Furthermore, the learning rate for the Adam optimizer is set to explore values between 1e-4 and 1e-2, using a logarithmic scale for sampling, and the momentum values are among 0.9, 0.7, 0.5, or 0.3.

The tuner can be run for a certain amount of trails and generate a list of the ten best trails evaluated on the validation MAE. We tested 100 different combinations and choose the best model based on the validation mean average error (MAE). The results form the Keras tuning is being saved to a text-file so we with less labour can choose a more optimal model [13]. Information on the 'adam' and 'mae' is further described in the Keras API docs [12]. The chosen ANN model is based on the best model found in the tuning process.

2.4.2 Evaluation of model performance

After running the two different models we will evaluate the models performance by comparing predicted to actual values using the validation and test-data. The models are trained on the training and validation data, while the test data is kept separate. If the model successfully predicts outcomes when using test dataset in predictions, it would indicate its effectiveness in predicting concrete strains as influenced by temperature and time.

We use the metrics of MSE, MAE, R2-score and visual comparison between predicted to actual values. MAE is a measurement of the the average absolute difference between actual and predicted values. MSE offers a perspective on the average of the squares of the errors, this metric is highlighting larger errors in the prediction. By comparing MSE/MAE ratio we can determine of the error is due to a consistent average error or due to a few large errors. R2 values indicate how close the fitted regression line is to the actual data. We will also visualize the the prediction's and evaluate how the predictions fit the actual values. Through a combined assessment of MAE, MSE, R2-score, and visual representation, we can determine which machine learning model, be it an ANN or RNN, is best suited for predicting strains in the Herøysund Bridge.

2.5 Critique of methodology

A critique of methodology evaluates the chosen methods adequacy to make valid conclusions, considering potential alternative approaches. It assesses the conclusion's weaknesses using criteria like validity, reliability, objectivity, and generalizability to gauge the strength of the response to the problem statement and research question.

2.5.1 Validity

The concept of validity is crucial for determining whether the results measure what one has aimed to investigate. Validity, or authenticity, informs us to what extent we can draw valid conclusions from the result. By evaluating the method's validity, we can determine whether the method yielded valid conclusions from the results [30]

The method of choosing to use KerasTuner on the ANN model and not the RNN model is reducing the validity of the results. The KerasTuner was set to evaluate 100 different ANN-model close to the example ANN found in Chollet's literature. The RNN model was constructed on an educated guess based on the LSTM models in Chollet's literature [8]. In what extent the examples could relate to the data we are evaluating, is yet unsure. This clear decision was made to lack of computationally resources and time restrictions, and is affecting the validity in the result from the two different models. There is a large possibility that a more optimized RNN model would give better results. This can result in us wrongly concluding on the problem statement.

There is also a possibility that the data processing, input and output variables was not optimal. These factors play a crucial role in the accuracy of the models predictions. Therefore, it's essential to consider these aspects when interpreting the results and making decisions based on the findings. Differences in data preprocessing or inappropriate selection of input and output could lead to biased or skewed results, which might affect the overall conclusions drawn from the study.

The *timeseries_dataset_from_array* method did not align with our expectations. Based on our review of existing literature, the y-values in a dataset typically represent measurements following the sequence. Despite experimenting with various approaches, our success was limited. Consequently, we assigned the entire set of y-labels to the dataset, a decision we now recognize as a potential error in time series prediction. Addressing this issue is crucial for future work, including a master's thesis, as it significantly impacts the validity of this project.

It is also a factor that the report may used the wrong evaluation metrics to asses the accuracy of the ML models predictions. The choice of metrics can interfere whit evaluating model performance. Using metrics that don't align with the objectives of the study could lead to an incomplete or misleading evaluation of the models performance.

2.5.2 Reliability

Reliability refers to how consistent the results of a test are when it's repeated under identical conditions. It measures whether a study would find similar differences or relationships if it were conducted again with a different sample [31].

When using KerasTuner, results can vary each time it's run. Even if we repeat the procedure under the same conditions, the outcomes may differ because of the inherent randomness in the learning process of the models. This stochastic nature

of machine learning models means they might produce slightly different results even with the same dataset and parameters. This inherent uncertainty in machine learning, and by extension in KerasTuner, affects the reliability of this project. Ideally, more effort should be put into exploring ways to minimize these variations under the same testing conditions.

2.5.3 Objectivity

Objectivity means being factual without personal input influenced by emotions. Objectivity is characterized by facts, not personal input. Subjectivity is the opposite of objectivity and is largely influenced by emotions. [32] The research can choose to act subjectively when choosing a model or a bad model. There is also a possibility that the researcher is evaluating the results subjectively. The computer and machine learning model is only acting objectively and is not influenced by emotions. These models function act independently of emotional influence, ensuring a more impartial and data-driven approach to result generation. The likelihood of personal biases influencing the outcomes due to emotional factors is low in this context.

2.5.4 Generalizability

Generalizability is the extent to which the sample applies to the entire population. [33, p. 147].

We conducted ML research on one for the 16 sensors at Hersøysund bride. To know for certain if RNN or ANN models are more efficient we need to test for multiple sensors and prove statistical significance in the result. Therefore we cannot conclude that the results are generalized for the hole population. However if RNN or ANN models is more efficient for one sensor it would indicate that this model is more efficient. The results provides valuable insights, but is not sufficient to definitively determine the relative efficiency of RNN or ANN models across the entire array of sensors.

3 Results

In this chapter, we will present results from the developed ML models. For a more in-depth examination of the resulting models, please refer to the project's GitHub repository or the appendices at the end of this report.

3.1 ANN model

For the ANN model tuning with KerasTuner was conducted. The results from the tuning was used to build the ANN model. More result from the tuning could be found in the GitHub repository.

ANN: KerasTuner - Best Trial Summary:

Table 3: Summary of the best trial in ANN model optimization using KerasTuner

Parameter	**Value**
Trial Number	016
Number of Dense Layers	2
Dense Units Layer 0	32
Dropout Layer 0	0.0
Learning Rate	0.00967
Momentum	0.3
Dense Units Layer 1	16
Dropout Layer 1	0.0
Score (val_mae)	**22.446**

ANN: Training and validation loss:
Figures 13 and 14 illustrate the development of training and validation loss (MSE) and MAE for each epoch during the model training. The x-axis represents the number of epochs, while the y-axis shows the respective loss (MSE) and MAE values.

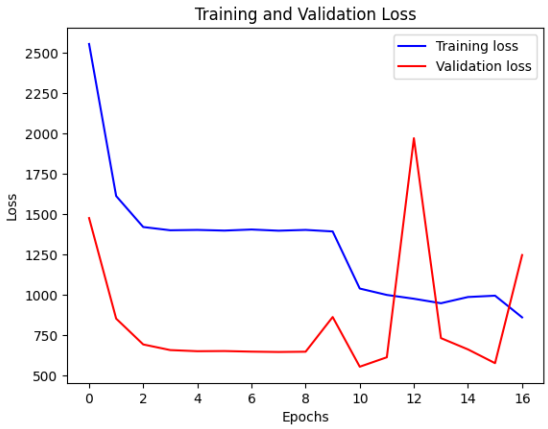


Figure 13: ANN training and validation loss (MSE)

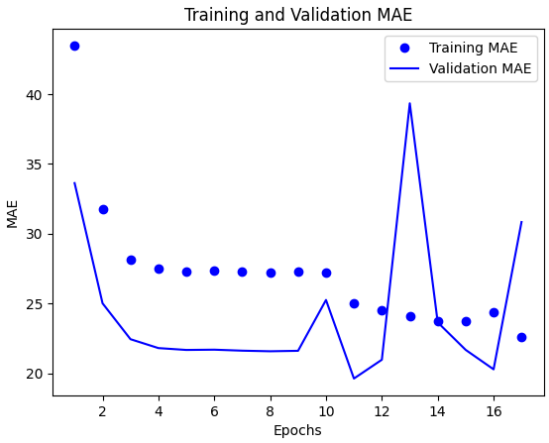


Figure 14: ANN training and validation MAE

ANN: Training predictions vs. actual values, 'Point_1_N'

Table 4: The ANN model's performance metrics.

Metric	Value
Mean Squared Error (MSE)	305.944
Mean Absolute Error (MAE)	13.183 μm/m
MSE/MAE Relationship	23.2
R2-Score - Training	0.721

The ANN model's performance metrics indicating its predictive accuracy on the training data. Figure 15 shows the actual versus predicted strain values in the training dataset.

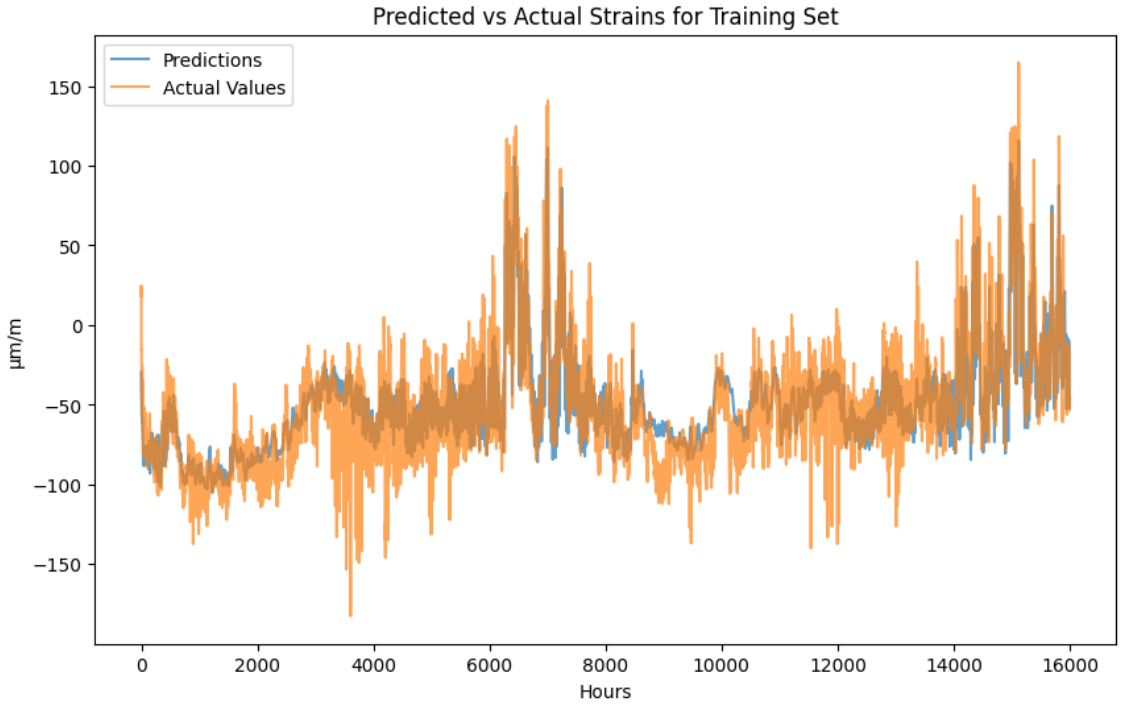


Figure 15: Training predictions and actual strains Point 1 N

ANN: Validation predictions vs. actual values, 'Point_1_N'

Table 5: The ANN model's performance metrics on validation data.

Metric	Value
Mean Squared Error (MSE)	348.110
Mean Absolute Error (MAE)	15.056 μm/m
MSE/MAE Relationship	23.1
R2-Score - Validation	0.443

The ANN model's performance metrics indicating its predictive accuracy on the validation data. Figure 16 shows the actual versus predicted strain values in the validation dataset.

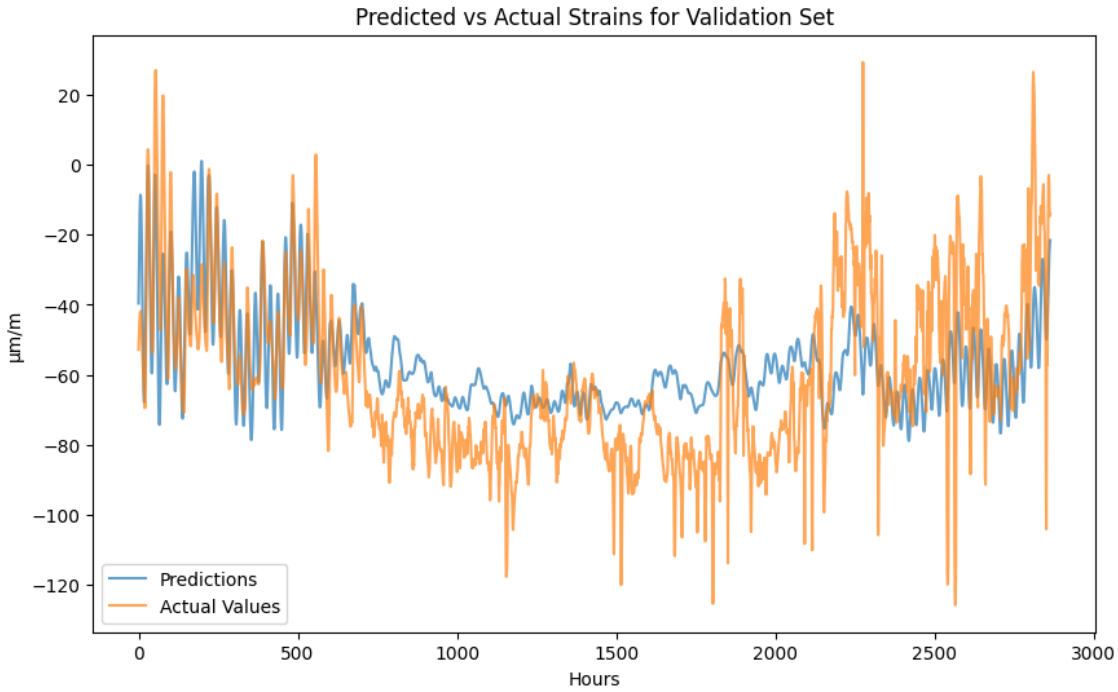


Figure 16: ANN validation predictions and actual strains, 'Point_1_N'

ANN: Test predictions vs actual values, 'Point_1_N'

Table 6: The ANN model's performance metrics on test data.

Metric	**Value**
Mean Squared Error (MSE)	1360.218
Mean Absolute Error (MAE)	28.454 μm/m
MSE/MAE Relationship	58.88
R2-Score - Test	0.236

The ANN model's performance metrics indicating its predictive accuracy on the test data. Figure 17 shows the actual versus predicted strain values in the test dataset.

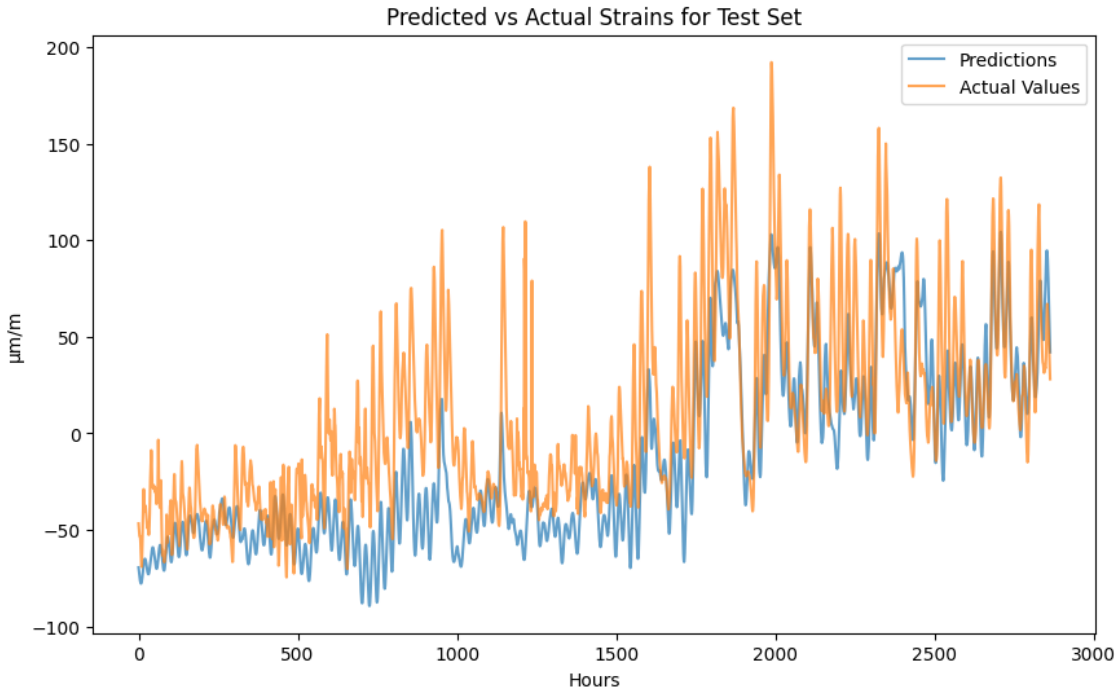


Figure 17: ANN test predictions and actual strains, 'Point_1_N'

ANN: Future predictions for 'Point_1_N'

Using average monthly temperatures we are plotting the ANNs prediction of strain development from year 2023 to 2028:

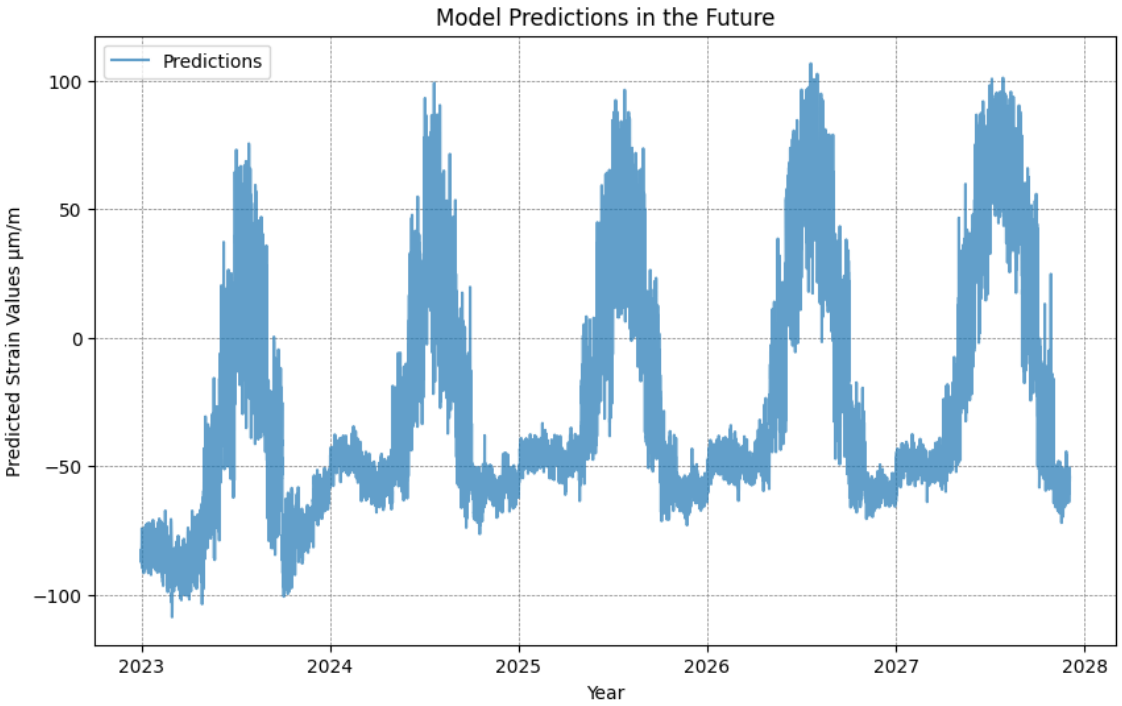


Figure 18: ANN predictions for the future

3.2 RNN model

The RNN model was constructed using similar build as examples from *Deep Learning with Python* [8]. KerasTuner was **not** used on this model.

Training and validation loss
Figures 19 and 20 illustrate the development of training and validation loss (MSE) and MAE for each epoch during the model training. The x-axis represents the number of epochs, while the y-axis shows the respective loss (MSE) and MAE values.

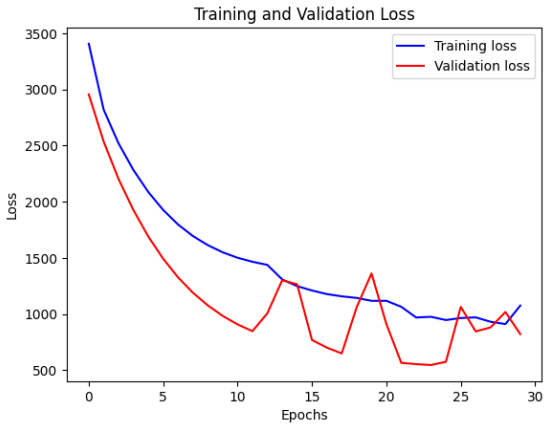


Figure 19: RNN training and validation loss (MSE)

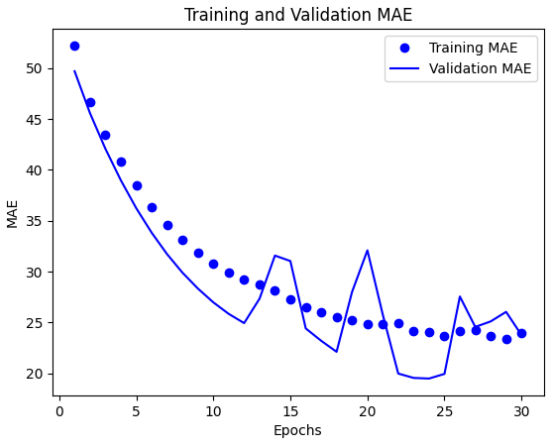


Figure 20: RNN traing and validation MAE

RNN: Training predictions vs. actual values, 'Point_1_N'

Table 7: The RNN model's performance metrics on training data.

Metric	Value
Mean Squared Error (MSE)	948.200
Mean Absolute Error (MAE)	23.577 μm/m
MSE/MAE Relationship	41.04
R2-Score - Training	0.316

The RNN model's performance metrics indicating its predictive accuracy on the training data. Figure 21 shows the actual versus predicted strain values in the training dataset. Visually assessing the accuracy, the predictions appear to fluctuate around the actual values with a notable variance. The overall accuracy seems to be compromised by periods of significant divergence for the RNN model.

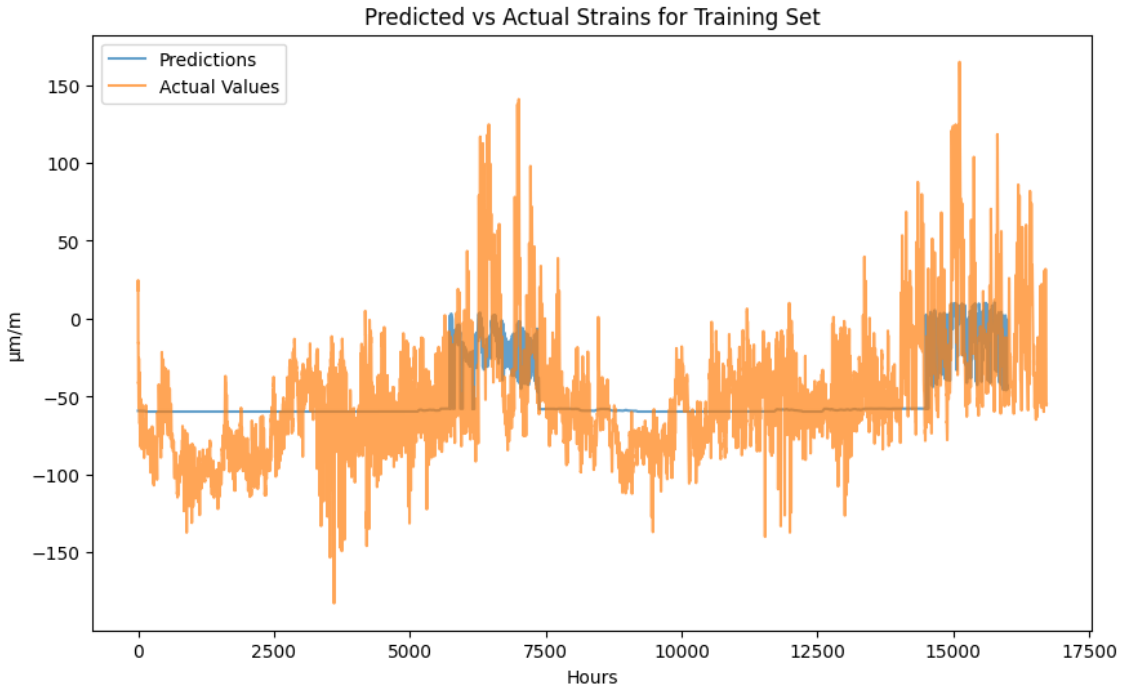


Figure 21: Training predictions and actual strains, 'Point_1_N'

RNN: Validation predictions vs actual values, 'Point_1_N'

Table 8: The RNN model's performance metrics on validation data.

Metric	**Value**
Mean Squared Error (MSE)	546.031
Mean Absolute Error (MAE)	19.493 μm/m
MSE/MAE Relationship	28.01
R2-Score - Validation	0.0373

The RNN model's performance metrics indicating its predictive accuracy on the training data. Figure 22 shows the actual versus predicted strain values in the training dataset.

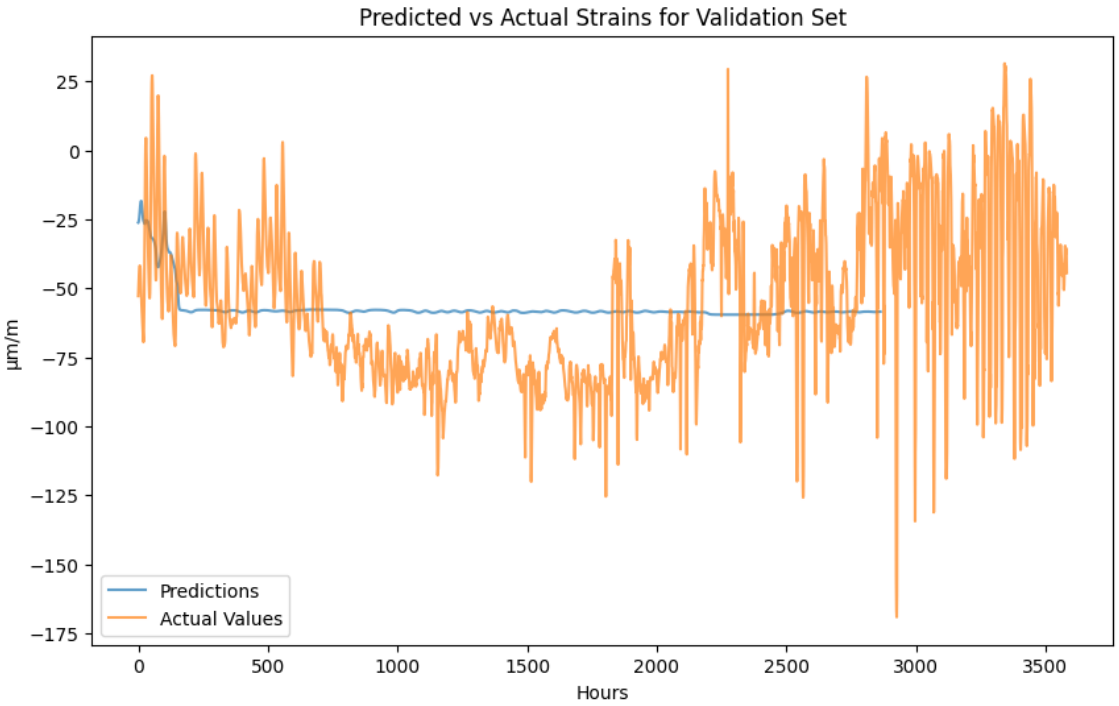


Figure 22: RNN validation predictions and actual strains, 'Point_1_N'

RNN: Test predictions vs. actual values, 'Point_1_N'

Table 9: The RNN model's performance metrics on test data.

Metric	**Value**
Mean Squared Error (MSE)	2584.458
Mean Absolute Error (MAE)	38.768 μm/m
MSE/MAE Relationship	66.66
R2-Score - Test	-0.0908

The RNN model's performance metrics indicating its predictive accuracy on the test data. Figure 23 shows the actual versus predicted strain values in the training dataset.

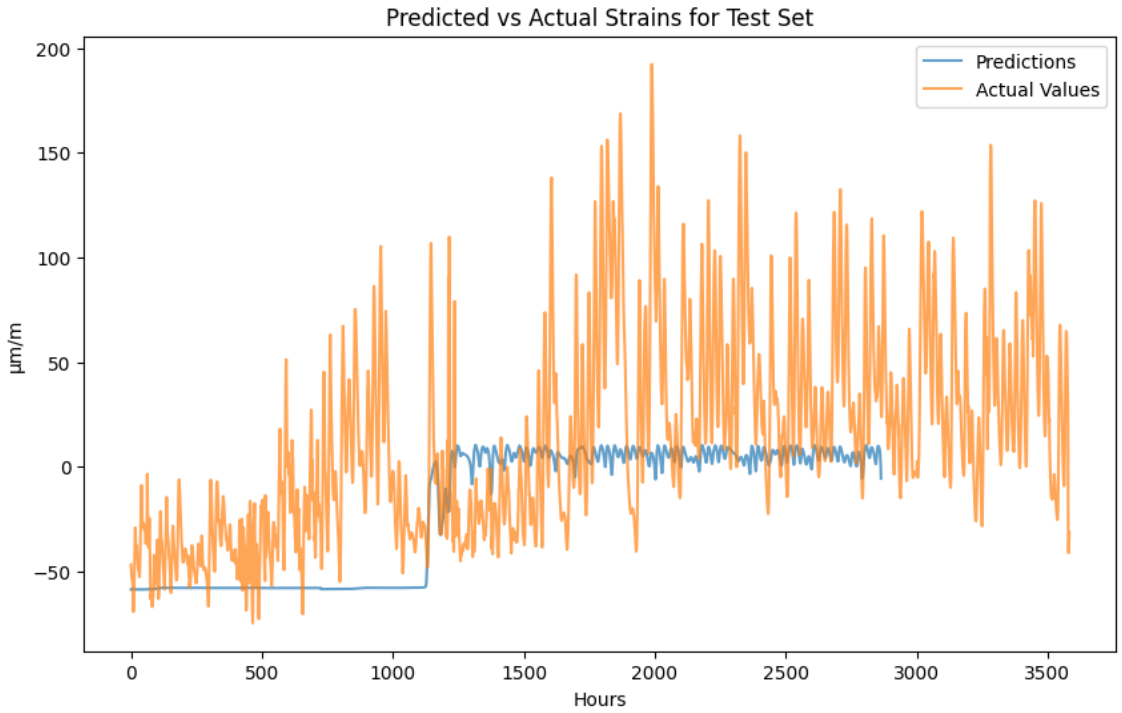


Figure 23: RNN Test predictions and actual strains, 'Point_1_N'

RNN: Future predictions for 'Point_1_N'
Using average monthly temperatures we are plotting the RNNs prediction of strain development from year 2023 to 2028:

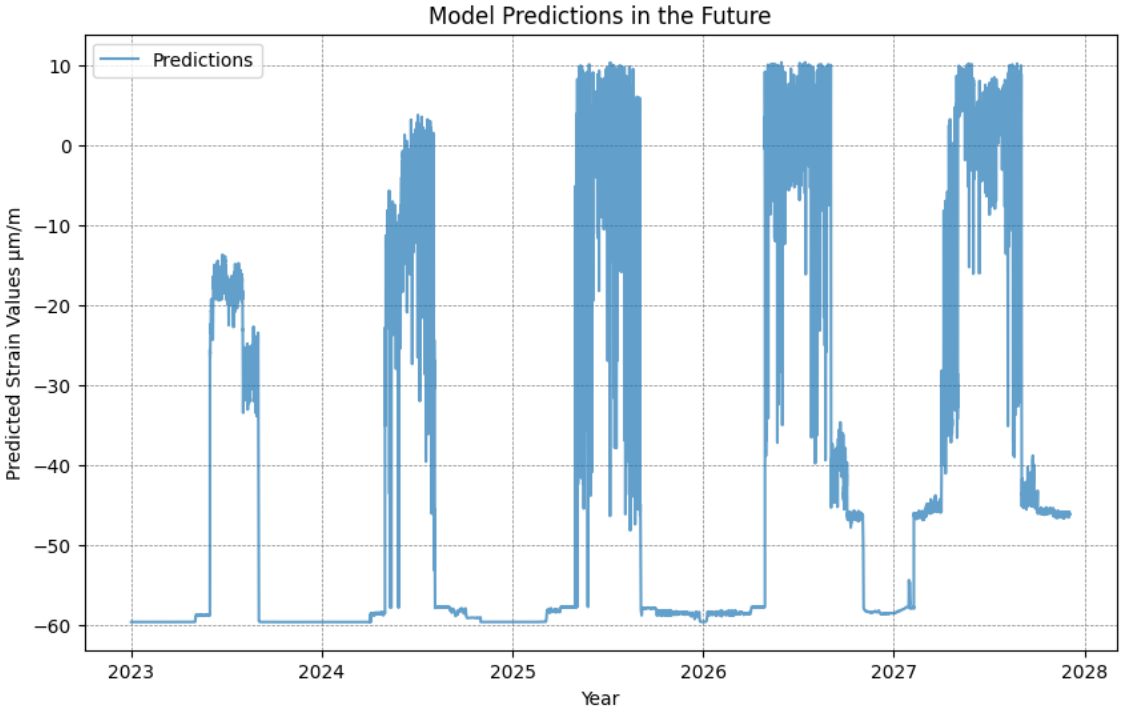


Figure 24: RNN models predictions for the future

4 Discussion

In this chapter we want to answer the problem statement "Which model, standard ANNs or RNNs, is more effective for predicting the specific strain development in the Herøysund Bridge?". With results from Chapter 3, we want to discuss the problem statement.

The discussion is intended to pave path for a conclusion on the problem statement and report. At the end of this chapter, an elaboration on potential future work would be carried out.

4.1 Which model, standard ANNs or RNNs, is more effective for predicting the specific strain development in the bridge?

We are evaluating the two chosen models performance using the metrics MSE, MAE, R2 score, as presented in Table 10. An visual comparison on the two models future prediction of strain values is also carried out. The metrics can give indications on the models effectiveness for predicting strain development by temperature and time. Through this analysis, we aim to determine whether the ANN or RNN model accurately forecasts the existing strain variations.

Table 10: Comparison of ANN and RNN Models Performance

Model	Data Type	MSE	MAE (μm/m)	R2 Score	MSE/MAE
ANN	Training	305.9442	13.1833	0.72	23.2
ANN	Validation	348.1107	15.0568	0.44	23.1
ANN	Test	1360.2180	28.4540	0.2364	58.88
RNN	Training	948.2007	23.5776	0.31	41.04
RNN	Validation	546.0319	19.4939	0.31	28.01
RNN	Test	2584.4583	38.7685	-0.09	66.66

ANN Model
The ANN model is able to predict the training data whit an average absolute error of 13.18 μm/m and a mean squared error or 305.94. The relationship between MSE/MAE is 23,2. We can see the ANN's predicting on the training dataset in Figure 15. A R2-score of 0.72 for the training dataset predictions, can indicate sufficent fit to the training data. Generally can a R2-score close to 1 be a sign of overfitting, we want our models to not come to close to 1.

The ANN model is able to predict strains in the validation data whit a MAE of 15.05 μm/m and a MSE of 348.11, both measures being higher than for the training dataset. The ratio of MSE to MAE in our analysis is 23.1. Additionally, an R2-score of 0.44 in the validation set suggests that approximately 44.35% of the variance in the dependent variable can be predicted from the independent variables. A R2 score of 0.44 can be considered moderate meaning that the model has some predictive power but a significant proportion of the variance is not being captured (And lower than

predictions for the training dataset). We can observe the predictions in Figure 16. The plot suggests that the model has challenges in accurately tracking the actual strain variations. This is particularly noticeable in the period from approximately 750 hours (about 31 days after first measurement) to around 2250 hours (roughly 94 days).

The ANN model is able to make prediction on the test data with a MAE of 28.4540 μm/m and a MSE of 1360.21. Additionally with a R2 score of 0.23 and a MSE/MAE ration on 58.88 can suggest that the model makes poor estimations on data it yet have not seen. The R2 score reveals that the Artificial Neural Network (ANN) is capable of predicting approximately 23.64% of the variance in the dependent variable for the test dataset. The low score, especially when compared to the training- and validation dataset's R2 score, can tell us that the models projecting power is limited. As seen in Figure 17, the plot suggest that the predicted values are consistently lower than the actual values. The relationship between the predicted and actual values does not align precisely, indicating the presence of noticeable discrepancies and inaccuracies in the model's predictions.

RNN Model

The RNN model is able to predict the training data with a MAE of 23.57 μm/m and a MSE of 948.20. The ratio of MSE to MAE in our analysis is 41.04. This high MSE/MAE ratio may indicate the presence of outliers or large errors that impact the MSE value. The R2 score of 0.31 suggests that your model is only able to adapt the actual values with about 31.65%. The actual and the models predicted strain values on the train dataset can be seen in figure 21.

The RNN model is able to predict the validation data with a MAE of 19.49 μm/m and a MSE of 546.03. A R2 score of 0.31 is suggesting that the model do not fit the validation data well and has a limited predictive power. Figure 21 represents the actual and the models predicted strain values, and telling by the plot, we can see that the predicted values seems to be linear and following a seasonal pattern of approximately 250 days.

The RNN model is able to predict the test data whit a MAE of 38.76 μm/m and a MSE of 2584.45. The MSE/MAE ratio of 66.66 suggests the presence of significant outliers or large errors, which are substantially impacting the MSE. The model predicts a R2 score of -0.09. The negative R2 score is particularly concerning. It indicates that the model does not fit the test data well and is less predictive than a model that always predicts randomly.

Determining the best model

To determine the which model is more effective in predicting, it is necessary to compare the test predictions. The test data performance is crucial for determining the models ability to generalize predictions outside of the sampled training and validation data. If the metrics indicate that the model is effective for predicting training data there is reason to believe that the model can predict strains in the future outside of the sampled data based of temperature and time.

Our chosen numerical evaluation metrics, demonstrate that the ANN model exhibits better predictive capabilities compared to the RNN model. This could be said to be

valid across all used metrics. When analyzing the two models performance, using the results from Table 10, the following observations can be made:

- The average absolute error of the RNN model is 38.45μm/m, which is higher than the ANN's 28.45μm/m, indicating the RNN has lower average predictive accuracy.
- The mean squared error for the RNN, at 2584.45, is significantly higher than the ANN's 1360.21. This suggests a greater impact of large errors or outliers in the RNN model than for the ANN model.
- MSE/MAE ratios of 58.88 for the ANN and 66.66 for the RNN highlight the influence of substantial error outliers on the MSE of both models.
- Despite this, both models show limited predictive accuracy for the test data. The RNN's negative R2 score of -0.09 is especially concerning. This could indicate a performance worse than randomly predicting. In contrast, the ANN's R2 score of 0.236, though not particularly high, suggests some level of predictive ability, surpassing random guesses.

The ANN model is preforming better than the RNN model on all the numerical evaluation metrics. Both of the models are still limited in there predictive accuracy outside of the sampled training and validation data, due to both model significantly becoming worse in making predictions on the test dataset.

The model is trained to adapt to the variations present in the training data, and the best-performing model is selected based on its MAE score on the validation data. Since the model is not directly trained on the variations specific to the test data, this likely explains why the model demonstrates better predictive accuracy on the training and validation datasets compared to the test data.

The figures 17 and 23 provide a visualization of the predicted versus actual values from the two different models, when making prediction based on the test data. From these visualizations, it may be interpreted that the ANN model demonstrates a more nuanced understanding of the test dataset's variations, in contrast to the RNN model, which seems worse at capturing these variations. This difference in performance is particularly noticeable, suggesting that the ANN model aligns more closely with the actual data trends.

Additionally, figures 18 and 24 represents the models' predictions for 'Point_1_N' spanning from 2023 to 2028. Here, the ANN model illustrates a volatile pattern with noticeable seasonal variations over the years, while the RNN model's predictions appear more linear, with abrupt shifts between seasons. However, it's crucial to note that both models exhibit low predictive accuracy, casting doubt on the reliability of these future projections.

The apparent seasonal variation in the ANN model's predictions, although more realistic than for the RNN model, does not necessarily translate to accuracy for the ANN model itself. Given the limited duration of the training data (1.5 years),

extrapolating five year of future predictions lacks scientific backing. Such forecasts might have been more credible if the models were trained on a more extensive, multi-year dataset and had demonstrated strong alignment with actual test strains, as indicated by metrics like a high R2 score. From our understanding of the conducted literature review, making prediction in a greater period of time than the dataset itself can be challenging and misleading. The prognosis on the long-term predictions has to be encountered with scepticism in it's reliability.

The long-term predictions highlights an aspect: the RNN model, particularly with its LSTM layers, struggles to capture seasonal variations. This is unexpected, as literature generally suggests that RNNs, especially those equipped with LSTM layers, are adept at identifying such patterns. LSTM neurons are designed to remember previous input variables, a feature the dense neurons lack. This error in the RNN model is likely due to how *tf.keras.utils.timeseries_dataset_from_array* was implemented in our RNN's architecture.

In our exploration of structuring time series data, we initially considered a predictive approach where the model would forecast future strains based on samples (time and temperature). The original intent was for the model to predict both long-term temperature variations and the resultant strains. We hoped-for a training process focused on predicting future strain from a given time series sequence and estimating strains at a specific time beyond the days covered in the sequence. This approach was meant to using a time sequence of a certain length to predict a single value ahead.

When defining the input- and output for the ML models using *timeseries_dataset_from_array* you have various chooses of sequencelenth, batch-size, delay and much more. We adopt a dataset whiteout delay. Implementing delay would compromise the both models ability to see the relationship between time, temperature and strain because this would create more uncertainty in the relationship between samples and targets. Our current model adopts a more direct, samples = truth relationship, where the samples and targets are compared for the same moment. The present configuration focuses on understanding the relationship between strains and temperature, rather than on forecasting future values using by using delay. While it is possible that a different configuration might enhance the RNN model's performance, our prediction results with the chosen data configuration do not provide any evidence to support this hypothesis.

5 Conclusion

The purpose of the analysis was to analyse witch model is the most effective for forecasting the strain development in bridges: recurrent neural networks (RNNs) or standard artificial neural networks (ANNs). We evaluated both models using various metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and R2 score, the report also evaluated by visual comparisons of predicted and actual values.

The ANN model i whit dense neurons is demonstrating better performance across all metrics in comparison to the RNN model. The test data is particularly important for generalising the models accuracy outside of the sampled training and validation data. The ANN model is preforming better at all the numerical predictive accuracy metrics like MSE, MAE and R2-score than the RNN model at all datasets. In addition the ANN model also performed better predictive accuracy than the RNN model at the visual comparisons between predicted and actual stain values. The ANN model demonstrated a better fit and showed more realistic seasonal variation in its predictions further strengthen its position as the more effective model for this specific application. However, it's important to note that both models showed limitations in predictive accuracy, therefore there is a need for cautious interpretation of their outputs, especially when considering long-term future predictions.

The analysis suggests that while the ANN model is more suited for our defined machine learning problem. This highlights the importance of choosing the right model based on the specific requirements and constraints of each predictive task. The RNN could potentially offer advantages in other more complex scenarios or for another data configuration. Our analysis of the prediction outcomes, based on the selected data configuration, does not yield any substantiating evidence to suggest that the Recurrent Neural Network (RNN) model outperforms the Artificial Neural Network (ANN) model.

The deviations in the results is highlighting weaknesses in both models ability to predicting strains outside of the training and validation data. Particularly in a machine learning task such as ours, it becomes obvious that the methodology employed is of the highest importance in structural engineering research. The findings underscore significant insights about the application of machine learning in structural engineering:

1. There is a critical need to deepen our understanding of machine learning, especially in recognizing its potential limitations.
2. We must acquaint ourselves with avenues for problem-solving within this domain.
3. The exploration of machine learning's capabilities is essential; without such exploration, progress and learning are hindered.

6 Recommendations and Future Work

In this chapter, we will focus on several topics for future work. This aspect of the study is particularly interesting, considering the limited knowledge we had at the beginning. The newfound knowledge has been enlightening, and if given the chance to redo this project, we would approach it differently.

There are specific aspects that are especially interesting in light of the upcoming master's thesis. We aim to delve deeper into the same field of study, focusing on ML applications in the context of concrete structures.

In a future scientific work, it would be interesting to investigate the following subjects:

- Comparing purely statistical methods with machine learning. As Thai said, not all problems have a perfect machine learning solution. Sometimes, advanced statistics might predict better. It would be good to use p-tests to see how well the methods work in general.
- The effect of using several strain gauges for predictions. We only looked at one sensor because of limited time and initial data issues. It would be interesting to see how using multiple sensors affects predictions for a specific sensor.
- Trying different time frames and including weather predictions. Right now, our model predicts based on a specific date and temperature. It would be interesting to either predict the weather in the model or use detailed forecasts from places like Yr.no. Our model used both types of data, but studying them separately would be useful.
- Experimenting with more data. We wanted to divide the data into training, validation, and testing sets. But to better understand seasonal patterns, using a larger set without setting aside a testing portion could have helped the model learn better. The bridge is still being monitored (as of November 2023), and the data keeps growing. Doing a similar study with more data, maybe before the bridge is taken down in 2025, would be interesting [2].
- Investigating changes in the bridge's internal structure due to different strains. If we could create a model, like a Finite Element Model (FEM), that takes strain data and shows the bridge's internal condition, it would be interesting to observe how this condition evolves over time through simulations.

References

[1] NTNU, *Herøy fou*, Accessed: October 23, 2023, 2023. [Online]. Available: https://www.ntnu.edu/kt/research/concrete/projects/heroy-fou.

[2] N. Fylkeskommune, *Fv. 828 herøysundbrua*, Website, Last accessed on November 15, 2023, 2023. [Online]. Available: https://www.nfk.no/tjenester/kollektiv-og-veg/fylkesveg/vegprosjekter/fv-828-heroysundbrua/.

[3] M. Selmurzaev, F. Hobbel and A.-H. Andersson, 'Report on instrumentation for nordland fylkeskommune', HBM Norge AS, Herøysund Bru, 8850 Herøy, 29th May 2023, Revision B, Approved by Finn Hobbel. Instrumentation details from Sept 01-03, 2020 and May 24-25, 2023.

[4] H.-T. Thai, 'Machine learning for structural engineering: A state-of-the-art review', *Structures*, vol. 38, pp. 448–491, 2022, ISSN: 2352-0124. DOI: https://doi.org/10.1016/j.istruc.2022.02.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352012422000947.

[5] L. Moroney, *AI and Machine Learning for Coders: A Programmer's Guide to Artificial Intelligence*, eng, First edition. O'Reilly, 2020, ISBN: 1492078190.

[6] S. Marsland, *Machine learning : An algorithmic perspective*, eng, Boca Raton, Fla, 2009.

[7] K. S. Ganesh, *What's the role of weights and bias in a neural network?*, https://towardsdatascience.com/whats-the-role-of-weights-and-bias-in-a-neural-network-4cf7e9888a0f, Accessed: [13.12.2023], 2023.

[8] F. Chollet, *Deep learning with python*, eng, Shelter Island, New York, 2022.

[9] A. Tidemann, *Neural networks*, Norwegian, https://snl.no/nevralt_nettverk, Accessed on October 27, 2023, from Store norske leksikon at snl.no, 2023.

[10] I. Goodfellow, Y. Bengio and A. Courville, *Deep learning* (Adaptive computation and machine learning), eng, 1st ed. London, England: The MIT Press, 2016, ISBN: 9780262035613.

[11] P. Walpita, *Recurrent neural networks in deep learning - part 1*, https://medium.datadriveninvestor.com/recurrent-neural-networks-in-deep-learning-part-1-df3c8c9198ba, Accessed: [12.12.2023], 2023.

[12] Keras. 'Api reference'. Accessed: 2023-11-07. (), [Online]. Available: https://keras.io/api/.

[13] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi *et al.*, *Kerastuner*, https://github.com/keras-team/keras-tuner, 2019.

[14] D. Morin, 'Machine learning in structural engineering', PowerPoint presentation by Associate Professor David Morin from NTNU, Sep. 2023.

[15] M. Flah, I. Nunez, W. Ben Chaabene and M. L. Nehdi, 'Machine learning algorithms in civil structural health monitoring: A systematic review', *Archives of computational methods in engineering*, vol. 28, pp. 2621–2643, 2021.

[16] M. Mishra, 'Machine learning techniques for structural health monitoring of heritage buildings: A state-of-the-art review and case studies', *Journal of Cultural Heritage*, vol. 47, pp. 227–245, 2021, ISSN: 1296-2074. DOI: https://doi.org/10.1016/j.culher.2020.09.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1296207420304441.

[17] 'Performance and efficiency of machine learning algorithms for analyzing rectangular biomedical data', eng, *Laboratory investigation*, vol. 101, pp. 430–441, 2021, ISSN: 0023-6837.

[18] A. Tidemann. 'Kunstig intelligens'. (), [Online]. Available: https://snl.no/kunstig_intelligens (visited on 20th Oct. 2023).

[19] *Training with built-in methods - keras documentation*, https://keras.io/guides/training_with_built_in_methods/, Accessed: 2023-11-07.

[20] P. R. Srivatsavaya, *Lstm vs gru*, https://medium.com/@prudhviraju.srivatsavaya/lstm-vs-gru-c1209b8ecb5a, Accessed: [12.12.2023], 2023.

[21] *Optimizers*, https://keras.io/api/optimizers/.

[22] A. Tidemann. 'Maskinlæring'. (), [Online]. Available: https://snl.no/maskinl%C3%A6ring (visited on 20th Oct. 2023).

[23] *Regression metrics - keras documentation*, https://keras.io/api/metrics/regression_metrics/, Accessed: 2023-11-07.

[24] *Adam optimization algorithm - keras documentation*, https://keras.io/api/optimizers/adam/, Accessed: 2023-11-07.

[25] TensorFlow, *R2Score - TensorFlow 2.10*, https://www.tensorflow.org/api_docs/python/tf/keras/metrics/R2Score, Accessed: 2023-11-16, 2023.

[26] A. Johannessen, L. Christoffersen and P. A. Tufte, *Introduksjon til samfunnsvitenskapelig metode*. Abstrakt forlag, 2020.

[27] S. Chacon, *Pro git*, eng, Berkeley, CA, 2014.

[28] Databricks Inc., *What is parquet?*, https://www.databricks.com/glossary/what-is-parquet, Accessed: 2023-11-02, 160 Spear Street, 13th Floor, San Francisco, CA 94105, 2023.

[29] M. Hosseini. 'Why is parquet format so popular? and how it compares to pandas dataframe?' A Data Scientist/Machine Learning Engineer, passionate about solving real-world problems — PhD in Computer Science. (Jan. 2023), [Online]. Available: https://morihosseini.medium.com/why-is-parquet-format-so-popular-da553f8dd9dc.

[30] S. Grønmo. 'Validitet'. (), [Online]. Available: https://snl.no/validitet (visited on 24th Oct. 2023).

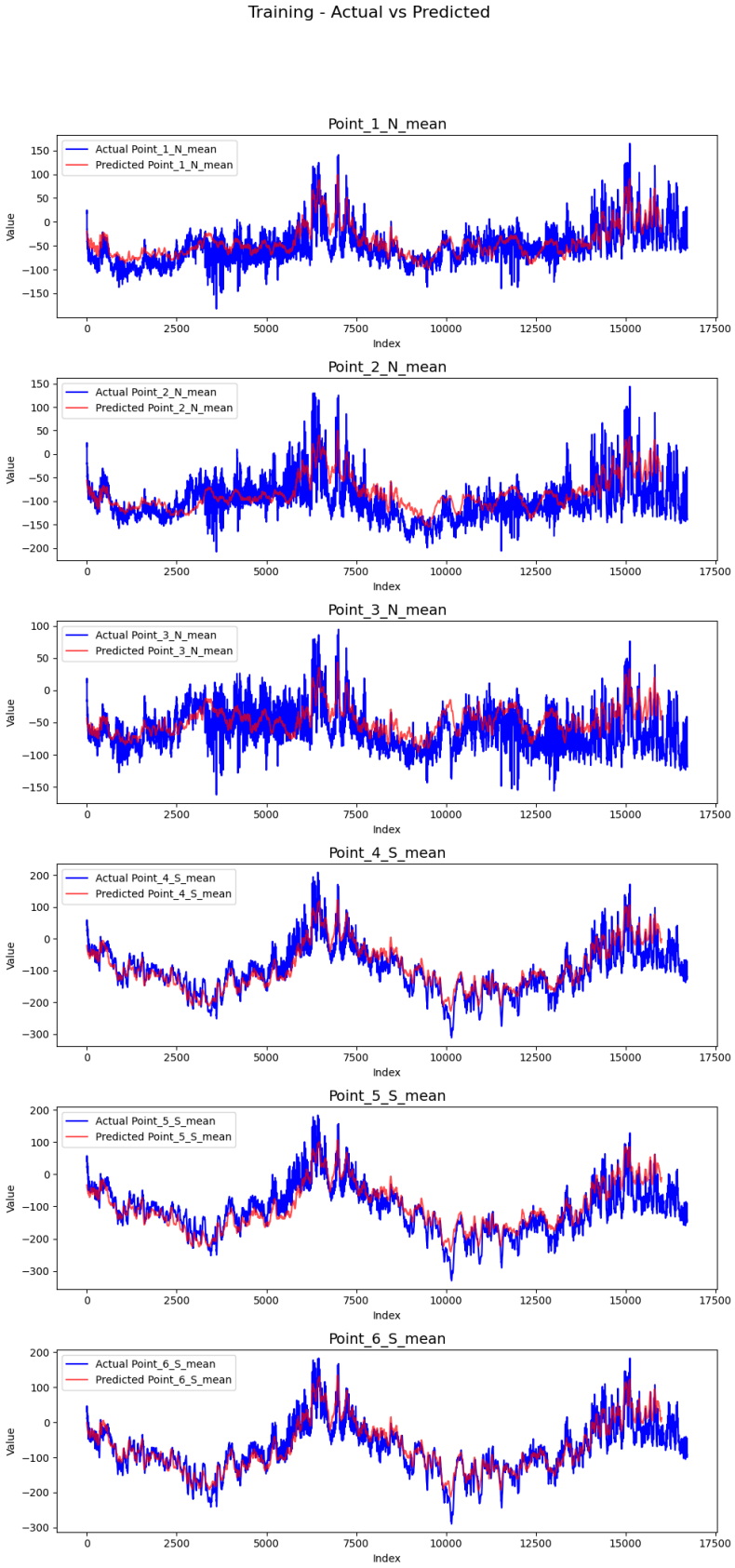
[31] S. Grønmo. 'Reliabilitet'. (), [Online]. Available: https://snl.no/reliabilitet (visited on 24th Oct. 2023).

[32] C. Nilstun. 'Objektiv'. (), [Online]. Available: https://snl.no/objektiv_-_saklig (visited on 24th Oct. 2023).

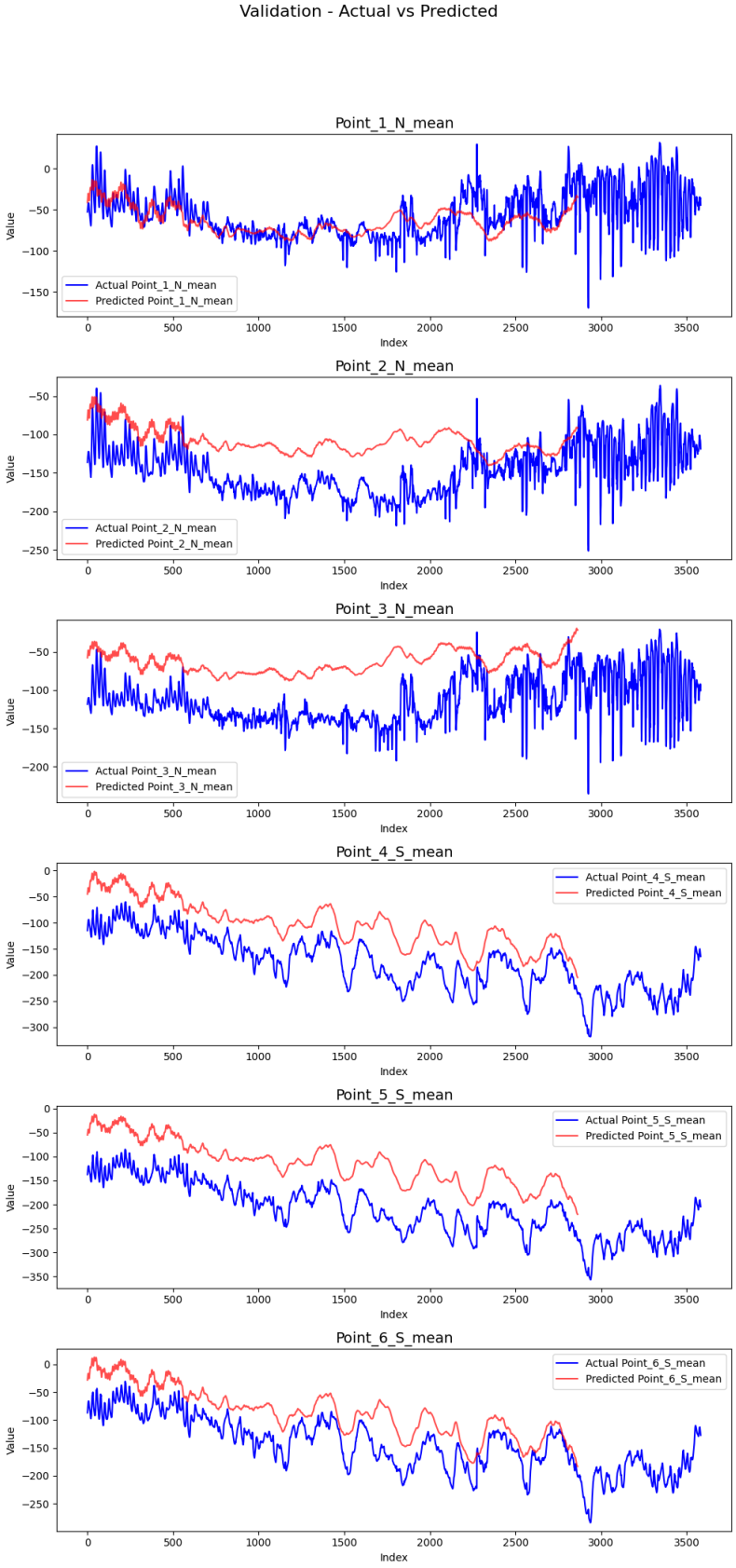
[33] O. Dalland, *Metode og oppgaveskriving (6. utg. utg.)* Gyldendal akademisk, 2017.

Appendix

A ML predicitons for all sensors at once training-data



B ML predicitons for all sensors at once validation-data



C ML predicitons for all sensors at once test-data