UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Physics and Technology

Validating Uncertainty-Aware Virtual Sensors For Industry 4.0

—

Gutama Ibrahim Mohammad

FYS-3941 Master's thesis in applied physics and mathematics - 30 SP

Abstract

In industry 4.0 manufacturing, sensors provide information about the state, behavior, and performance of processes. Therefore, one of the main goals of Industry 4.0 is to collect high-quality data to realize its business goal, namely zero-defect manufacturing, and high-quality products. However, hardware sensors cannot always gather quality data due to several factors. First, industrial 4.0 deploys sensors in harsh environments. Consequently, measurements are likely to be corrupted by errors such as outliers, noise, or missing values. Sensors can, over time, be subject to faults such as bias, drifting, complete failure, and precision degradation. Moreover, direct sensing of a process variable can be unavailable due to environmental constraints such as surface temperature being beyond the range of the physical sensor.
A virtual sensor is a tools to solve these problems by allowing for online estimation of process variables when the physical sensor is unreliable or unavailable. Deep learning method is effective in developing virtual sensors; however, it assumes that the data used for training and deployment are independent and identical (i. i. d). Therefore, deep learning in high-risk environments, such as industry 4.0, is challenging because if i.i.d assumptions fail to hold, the model may make errors that lead to disastrous consequences, such as financial losses, reputational damage, or even death. We can prevent model mistakes only if the model estimates the uncertainty of its predictions. Unfortunately, current deep learning-based virtual sensors are created using frequentist models, making them unable to capture uncertainty accurately. In this thesis, we explore the possibility of Bayesian convolutional neural networks (BCNN) to generate uncertainty-aware virtual sensors for Industry 4.0.
We use two publicly available realistic industrial datasets to generate virtual sensors and conduct experiments. CNC Mill Tool Wear data (CNC) from CNC milling machine provided by the University of Michigan, and Tennessee Eastman Process data (TEP) provided by Eastman Chemical Company for process monitoring and control studies. The root-mean-square error (RMSE), mean absolute percentage error (MAPE), and R-squared ($R2$) is used to evaluate the predictive capability of the generated virtual sensor. The performance is compared to that of the standard neural network-based virtual sensor, namely convolutional neural network (CNN) and long short-term memory (LSTM). We demonstrated Bayesian neural networks' ability to quantify uncertainty

by computing the coverage probability of the uncertainty. Additionally, we tested whether the estimated uncertainty could detect changes in input data distribution using the fault injection method.
Our BCNN virtual sensor had the best R-squared scores, with $R2 = 0.99$ on CNC and $R2 = 0.98$ on TEP data. The result of the coverage probability score indicates a reasonably good uncertainty estimate. However, despite predictive uncertainty detecting faults in input datasets, its accuracy declined as fault length increased.

Acknowledgments

Firstly, I would like to express my deep gratitude to my mother and father, who have made so many sacrifices on my behalf. Thanks to their hard work and prayers, I have come this far. Thank you to Ahmed, Aziza and Sena, my brother and sisters, who are always there for me and motivate me when times get tough.

A special thanks must go out to my main supervisor, senior research scientist Sagar Sen, for his guidance and assistance throughout the entire process. I would also like to thank my supervisor at the University of Tromsø, Professor Robert Jenssen, for his valuable feedback. Thanks to SINTEF Digital for providing me with an internship opportunity and collaborating on my thesis.

Thanks to my classmates for the beautiful times I have shared with you. You are some of the most remarkable people I've ever met.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

The Fourth Industrial Revolution (or Industry 4.0) (Lasi et al., 2014) refers to the ongoing innovative use of technology to automate traditional manufacturing and industrial processes. Large-scale machine-to-machine communication (M2M) and the internet of things (IoT) are integrated for increased automation, improved communication and self-monitoring, and production of intelligent machines that can analyze and diagnose issues without the need for human intervention. Observing the manufacturing process with sensors and collecting high-quality data to improve product quality and the overall process is one of the main objectives of Industry 4.0.

High-quality data should not only reflect the actual real-world states of the manufacturing process but should also help realize the business goal of Industry 4.0, which is zero-defect manufacturing (Wang, 2013) and higher quality products.

A typical Industry 4.0 manufacturing process has several inputs such as the 3D model of the part to be manufactured either via additive (Wong and Hernandez, 2012) (e.g. 3D printing) or subtractive manufacturing (Ehmann et al., 1997) (e.g. milling, grinding, laser cutting), the material for the part, feed-rate of the cutting tool, and clamping pressure on the part.

Sensors provide information about the behavior of a process and information

about its performance. For example, during the manufacturing process, numerous sensors on the machining instrument can measure time-varying data with sub-microsecond sampling frequencies such as accelerations, velocity, and positions of the part and the tool (e.g., spindle) and its internal currents and voltages. In addition, we may install novel sensors close to the tooltip, such as acoustic sensors and accelerometers, to obtain fine-grained information at the interface between the tool and the part's material.

Machine learning or AI models typically use these data to perform predictive and preventive maintenance. However, predictive and preventive maintenance can help improve product quality and reduce waste and malfunction rates only if the sensor data are high quality and reliable.

Quality data from hardware sensors may become unavailable because of different factors. First, industry 4.0 manufacturing deploys physical sensors in harsh conditions. As a result, measurements are likely to be corrupted by errors such as outliers, noise, and missing values. In addition, due to limitations such as life span, battery power, and bandwidth. physical sensors can over time be subject to faults such as bias, drifting, complete failure, precision degradation Moreover, direct sensing of a process variable can be unavailable due to environmental constraints such as surface temperature being beyond the range of physical sensor (Kadlec et al., 2009). Even when collecting data through a physical sensor is possible, mounting a plethora of sensors is impractical in a cost-sensitive industry 4.0.

A virtual sensor, also known as a soft sensor, is one of the tools available for industry 4.0 to solve these problems by allowing for online estimation of process variables when the physical sensor is unreliable or unavailable. In general, two approaches to establishing virtual sensor models can be distinguished, namely model-based and data-driven methods (Kadlec et al., 2009). When the process mechanism is known, the model-based approach can work well. However, when it isn't known, it cannot be used, especially in complex industry 4.0 processes. Furthermore, model-driven models are computationally expensive. As a result, data-driven methods are the mainstream virtual sensor method (Sun and Ge, 2021). Data-driven methods rely only on historical data obtained from the industrial processes and therefore do not require knowledge of phenomenology.

For modeling data-driven virtual sensors, traditional techniques include statistical inference and machine learning, such as principal component analysis(PCA) (Ge et al., 2014), partial least square(PLS) (Zheng and Song, 2018), support vector machines (SVM) (Herceg et al., 2019) and artificial neural network (ANN) (Fan et al., 2019).

However, even though those methods have many applications, some of their

drawbacks include heavy workloads caused by hand-crafted feature engineering or performance issues when dealing with large datasets (Sun and Ge, 2021). Therefore, Deep learning methods have contributed to the growing development of virtual sensors in recent years. Furthermore, due to its powerful learning ability, deep learning can enhance the performance of virtual sensors performance compared to conventional methods.

Deep learning models assume that the training, testing, and deployment data is independent and identically distributed (i.i.d.) (Lakara et al., 2021). Unfortunately, this assumption doesn't generally hold in a real-world setting. Hardware sensors are subject to failures like drift, bias, and freezing in the context of industry 4.0, resulting in data shifts. Suppose the shift in deployment data is significant compared to training data. In that case, there is a likelihood the model will make a mistake that may lead to unforeseeable, potentially disastrous results like financial or reputation losses or even death. Conversely, present deep learning-based virtual sensors are created based on frequentist models that represent weights as deterministic values, so they cannot capture uncertainty appropriately in their outputs. Moreover, most of the current performance evaluation techniques for deep learning models depend on ground truth unavailable during most virtual sensor deployment. Therefore, If the model doesn't produce uncertainty estimates, we can't prevent the model's mistakes. These factors pose a threat to the reliability of virtual sensors.

Bayesian probabilistic methods offer a mathematically grounded way of gaining insight into data and capturing accurate uncertainties in model predictions (Droguett and Mosleh, 2008). There have been many advances in Bayesian deep learning communities for developing better and more efficient approximate methods for extending Bayesian deep learning to probabilistic Bayesian methods in recent years (Graves, 2011; Blundell et al., 2015; Srivastava et al., 2014; Kingma and Welling, 2013; Wen et al., 2018). By combining deep learning with Bayesian networks, Bayesian neural networks can appropriately quantify uncertainty thanks to their robustness and stochasticity. Compared to frequentist methods, Bayesian neural network methods are less often overconfident (Kristiadi et al., 2020) and their uncertainty estimate is more consistent with the observed errors (Ovadia et al., 2019). Another advantage of Bayesian neural network method is that it allow to distinguish between the model uncertainty (**epistemic**) and inherent data randomness(**aleatoric**) (Der Kiureghian and Ditlevsen, 2009). Because we can distinguish between these two sources of uncertainty, we can reduce model uncertainty and accept the randomness in our datasets.

1.2 Research Aims

Bayesian neural networks are a relatively new field of research, and there are no implementations of them for virtual sensors to our knowledge. This thesis will use a Bayesian convolutional neural network (BCNN) to create an uncertainty-aware virtual sensor. We will then test how well uncertainty estimated by BCNN can detect changes in the virtual sensor input dataset.

Specifically, we would like to investigate the following research questions (RQ).

RQ1 How can we build and validate uncertainty-aware virtual sensors using a Bayesian neural network?

RQ2 How can we evaluate the accuracy of the uncertainty estimated by the Bayesian neural network?

RQ3 How well is uncertainty estimation able to detect single sensor faults?

a) **Faults due to drift**

b) **Faults due to bias**

c) **Faults due to freezing**

d) **Faults due to precision degradation**

The primary contributions of our work are summarized as follows:

1. Adapting the newly developed estimation approach called Flipout (Wen et al., 2018), we propose an end-to-end probabilistic virtual sensor pipeline based on Bayesian convolutional neural networks (BCNNs).

2. We employ a simple and user-friendly approach of selecting inputs based on information generated by an open-source Python program called Pandas profiling (Brugman, 2020).

3. Through experiments, we demonstrate that the proposed methods can generate predictive uncertainty, allowing for detecting faults in input sensors.

4. We use Data Version Control (DVC) Kuprieiev et al. (2021) to manage the workflow of the generated virtual sensor, that is, tracking the different stages of preparing the data, different model hyperparameters, and the

model itself.

1.3 Thesis Outline

Chapter 2 discusses the relevant theory and demonstrates its relation to our work. First, we will briefly discuss the manufacturing process since we will use a dataset from the manufacturing domain. Next, we will discuss physical and virtual sensors, including faults in sensors, since sensing devices are essential components of manufacturing process monitoring systems and a primary data source for virtual sensor development. We will also cover the theoretical background of deep learning methods used to generate virtual sensors and review some published deep learning-based virtual sensors. In addition, we will discuss types of model uncertainty and how they can be estimated. Finally, we will also describe the theoretical aspect Bayesian neural network, representing our method of creating virtual sensors.

Chapter 3 describes the development and evaluation of virtual sensors in detail. We will first describe the techniques for data feature engineering. Following that, we will explain the virtual sensor generation method, including architecture, explanations of training, and evaluation strategies for the virtual sensor.

In Chapter 4 we present our experiments and results. We will start by describing the datasets used to conduct experiments. Then we will explain how different choices are made during the data feature engineering and training stages. We provide statistical data and discussion to support our conclusions. In this way, we provide an answer to our research questions.

The final chapter 5 concludes with some concluding remarks, and discusses various future directions.

1.4 Nomenclature

In this thesis, we use different abbreviations, symbols, and expressions from different fields. We summarize their definitions and explanations in Table 1.1.

Table 1.1: The nomenclature used in the thesis. The notations are separated into categories.

	Acronyms
EU	Epistemic uncertainty
AU	A Aleatoric uncertainty
PU	Predictive uncertainty
RMSE	Root-mean-square error
MAPE	Mean absolute percentage error
R2	R-squared
CNN	Convolutional neural network
BCNN	Bayesian convolutional neural network
TCN	Temporal convolutional network
RNN	Recurrent neural network
MCMC	Markov Chain Monte Carlo
	Data Sets
$\boldsymbol{X}$	Feature set containing N samples $\{x_1, x_2, .., x_N\}$
$\boldsymbol{y}$	Target(label) set containing N samples $\{y_1, y_2, .., y_N\}$
$x^{(i)}$	i-th example (sample) from data set X
$y^{(i)}$	Target (label) associated with $\boldsymbol{x}^{(i)}$
$\mathbb{D}$	Set containing the complete training data
	Statistical notations
μ	(The greek letter "mu") is used to denote the mean
σ	(The greek letter "sigma") is used to denote the standard deviation
θ	(The greek letter "theta") is used to denote a prameter of a function
ϵ	(The greek letter "epsilon") is used to denote a noise/sample from standard gaussian distribution
$p(w\|D)$	True Probability distribution of weights w given dataset d
$p(D)$	The marginal distribution of dataset D
$q(\mathbf{w};\theta)$	Estimated Probability distribution of weights w parametirized by parameter θ
$KL(q\|\|p)$	Kullback–Leibler divergence between probability distributions p and q
E_q	Expectation with respect to q
log	logarithm
i.i.d	independant and identical distribution

2

Background

This chapter will present and connect essential concepts relevant to our work. We begin with an introduction to industrial manufacturing in §2.1 since we use datasets from industrial manufacturing processes. Next, sensing devices are the critical building blocks of manufacturing process monitoring systems and the primary data source for virtual sensor development, so we will discuss them in §2.2. We will also discuss sensor faults and virtual sensors in the same section. We will use sensor faults described here to create artificial faults to test whether the virtual sensor we generate can detect changes in the input dataset effectively. Since deep learning methods are currently the mainstream virtual sensor methods, we will briefly introduce the theoretical concepts behind deep learning and review some deep learning-based virtual sensors. In section§2.4, we introduces uncertainty in machine learning models. Since this thesis aims to explore the potential of the Bayesian neural network to generate uncertainty-aware virtual sensors for Industry 4.0, we will cover the theoretical concepts behind Bayesian neural networks in the last section of this chapter §2.5.

2.1 Industry 4.0 Manufacturing

2.1.1 Additive Manufacturing

A key component of industry 4.0-based smart factories is an additive manufacturing(AM), also known as 3D printing (Gibson et al., 2014). AM creates

complex and straightforward geometries by combining layers of often powder or liquid into 3D models by building them up slowly (layer by layer). By utilizing this technology, manufacturing companies can produce parts with high creative-design freedom, mass customization, and minimal cost. AM methods fall into the following seven categories.

1. Vat PhotoPolymerization(VPP).
2. Powder Bed Fusion(PBF).
3. Material Extrusion(MEX).
4. Material jetting(MJT).
5. Binder Jetting(BJT).
6. Sheet Lamination(SHL).
7. Directed Energy Deposition(DED).

2.1.2 Subtractive Manufacturing

Subtractive manufacturing (SM), also known as machining, is used to remove material selectively and controllably from raw materials to obtain the desired object (Youssef and El-Hofy, 2008). It is possible to perform the procedure manually, but computer numerical control (CNC) (Thyer, 2014) is today's most popular method. A CNC system consists of a machine tool and computer that controls the machine. The machine allows the user to pre-program the tool's speed and position through installed software. This software acts like a robot, operating without human intervention. CNC machining techniques include milling, drilling, boring, turning, and reaming. There are also multi-mode CNC machines that combine various operations into one system to handle complex objects (Moriwaki, 2008).

In summary, AM and SM manufacturing methods produce products in various ways. Most SM processes entail heavy milling, drilling, boring, and turning, taking multiple steps and routes (Ehmann et al., 1997). In AM, some processes use liquid material cured by some form of energy, whereas others use powders or wires melted by a laser or electron beam. Other processes use a print head to spray binder into powdered material (Wong and Hernandez, 2012). Process parameters in both SM and AM are sensitive to environmental variations and affect one another. Product quality and machine health are directly related to those parameters. There are also other factors affecting the quality of products,

including void formation (Eiliat and Urbanic, 2018), anisotropic behaviours (Kok et al., 2018).

2.2 Physical Sensors

According to (Tönshoff and Inasaki, 2001), a physical sensor is a device that receives input signals or energy (such as heat, light, sound, pressure, magnetism, or particular motions) and converts those signals or energies to an output signal or energy. Six general categories of sensor output signals exist:

- mechanical.
- thermal.
- electrical.
- magnetic.
- radiant.
- chemical.

.

A mechanical sensor measures mechanical phenomena. Position, velocity, acceleration, force, pressure, stress, strain, mass, density, momentum, torque, shape, roughness, and orientation are among the different measurands targeted by mechanical sensors (Pasquale, 2003). Mechanical sensors are a big deal in manufacturing because most measuring elements are essential in most manufacturing processes. Temperature is one of the most critical parameters to measure, and control in manufacturing environments (Lu and Wang, 2018; Childs et al., 2000). Hence, thermal sensors are crucial to a smart factory. A magnetic sensor measures the magnitude of a magnetic field generated by an electric current or a magnet. They are excellent for measuring non-magnetic signals like angular velocity and acceleration (Yaghobi, 2016) or proximity detection, and This makes them ideal for use in complex manufacturing environments. The electrical sensors measure parameters like charge (Li et al., 1996), current (Li et al., 2000), permittivity (Pérez and Hadfield, 2011), and electric field (Lin et al., 2011). The major purpose of electrical sensors in manufacturing is to measure how a product behaves. Chemical sensors provide information about the chemical structure of their environments. Among other things, they measure liquid, gas, and concentration. Such devices are valuable

for monitoring manufacturing processes.

2.2.1 Sensor Faults

A sensor fault means that all or part of the sensor's functionality is lost. Several factors can cause sensors to malfunction, including aging, low batteries, harsh working environments, improper calibration, and hardware failure. Currently, there is no full consensus on a list of all types of sensor faults. For example, (Isermann, 2005) distinguishes between two kinds of sensor faults based on how they occur: abrupt (step-wise) and incipient (drift-wise). By contrast, (Liu et al., 2011) categorized sensor faults as follows according to the shape of deviation:

1. Drift.
2. Bias.
3. Precision degradation.
4. Freezing.

The last categorization has appeared repeatedly in different literary works. Therefore, we will follow this classification.

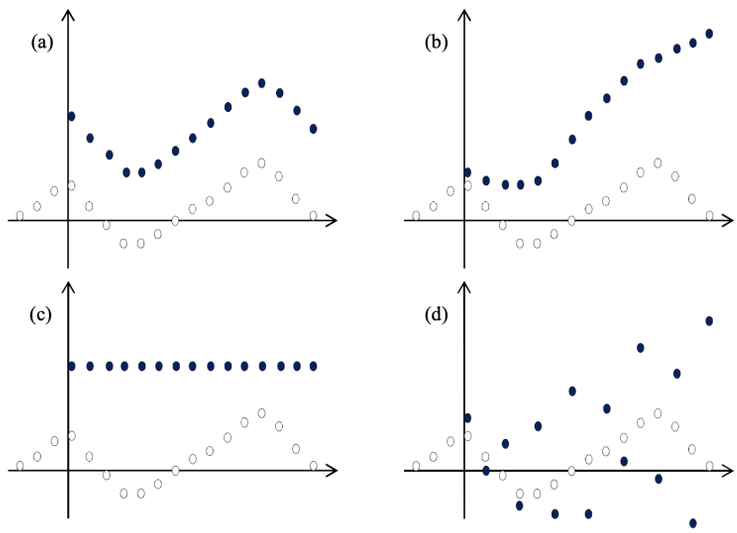

Figure 2.1: The four most common types of sensor faults are (a) bias, (b) drifting, (c) complete failure (freezing), and (d) precision degradation. Circle dots represent actual measurements, while filled dots indicate faulty sensor readings. Adapted from (Sharma et al., 2010).

Using these four fault types, we will create artificial sensor faults to test whether uncertainty can effectively detect changes in the input dataset. Therefore, we will summarize and provide mathematical formulas for each fault type based primarily on (Jan et al., 2021).

Suppose that $y(t) = x(t) + \epsilon$ is the expected output of a normally behaving sensor. With $x(t)$ being the input to the sensor at time t and $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$ being the noise with 0 mean and σ_ϵ standard deviation.

Drift Sensor is said to be drifted when its output increases continuously at a constant rate due to factors such as material corrosion, damage due to extreme environment condition. Mathematically sensor drift can be represented as:

$$y(t) = x(t) + \epsilon + d(t) \tag{2.1}$$

Where $d(t) = d(t-1) + C$ is a linearly increasing drift error and C is a constant.

Bias Sensor bias is one of the most common faults in sensors. Bias is an offset that shift normal sensor output by a constant value. We may mathematically express it as:

$$y(t) = x(t) + \epsilon + C, \quad \text{where} \quad C \quad \text{is a constant.} \tag{2.2}$$

Freezing A sensor is freezing when the output is a constant for a large number of successive samples. The constant output value is either very high or very low compared to normal sensor reading, and uncorrelated to underlying physical phenomena (Sharma et al., 2010). Mathematically:

$$y(t) = C, \quad where \quad C \quad \text{is a constant.} \tag{2.3}$$

Precision Degradation Precision degradation occur when the variance of sensor reading increases for a number of successive samples. It is mathematically modelled as:

$$y(t) = x(t) + \epsilon + \gamma \tag{2.4}$$

Where $\gamma \sim \mathcal{N}(0, \sigma_\gamma^2)$ is a normally distributed noisy with standard deviation $\sigma_\gamma \gg \sigma_\epsilon$.

2.3 Virtual Sensors

Virtual sensor is a term used to describe model/data driven systems for solving problems created by unavailability or limitation of physical sensors (Kadlec et al., 2009). They can be used independently as a low-cost alternative to expensive hardware sensors or in parallel with hardware sensors), thus allowing the realization of more reliable processes. The tasks fulfilled by virtual sensors is broad, but the most dominant application area is in the prediction manufacturing process parameters, which are often related to the quality of products Fortuna et al. (2007); Kadlec et al. (2009); Fortuna et al. (2005).

2.3.1 Neural Networks and Deep Learning

During the past few years, deep learning methods have significantly accelerated the development of virtual sensors in industrial fields. Therefore, we will first briefly introduce the theoretical concepts behind standard neural networks in this section.

Machine learning is a computing system that can learn by itself without having to be explicitly programmed by a human (Jordan and Mitchell, 2015). Neural networks, or artificial neural networks more specifically, are a class of machine learning that mimic the learning processes of the neural networks found in animal brains Goodfellow et al. (2016). The basic structure of a neural network consists of an input layer, hidden layers, and an output layer. In deep neural

networks, "deep" refers to the number of layers.

A neural network can learn in three different ways: supervised (Caruana and Niculescu-Mizil, 2006), unsupervised (Barlow, 1989) and reinforcement learning (Sutton and Barto, 2018). Supervised learning involves assigning labels to each training example $\mathbf{x}$ with ground-truth/label y. Deep learning or machine learning aims to find a mapping between $\mathbf{x}$ and y. Unsupervised learning occurs when there is no label for the training sample $\mathbf{x}$. Learning models are designed to uncover knowledge hidden in training examples $\mathbf{x}$. Data is not predefined in reinforcement learning. Instead, there are **agent** and **environment**. The agent gains knowledge by discovering its environment.

It's possible to classify neural networks based on the type of problem they address or dataset they use. The three main types of neural networks used in deep learning are:

- Feed-forward neural network.
- Recurrent neural network.
- Convolutional neural network.

Feed-forward Neural Network A feed-forward neural network is a neural network that associates input signal to output signal. Each layer consists of neurons, and the connections between layers do not form a loop (Svozil et al., 1997). Thus, signals can only travel one way, i.e., from input to output. Pattern recognition is a typical application of feed-forward neural networks.

Recurrent Neural Network The opposite of a feed-forward neural network is a recurrent neural network (RNN) (Mikolov et al., 2010). In a feed-forward neural network, all the inputs and outputs are independent.

In RNN, connections between neurons can form a loop. As such, signals are allowed to travel through time, i.e., RNN remembers each previous output of a layer. RNN deals with sequential data where one input follows another in time, such as time series. It achieves state of art performance on important tasks such as language modelling (Salovey and Mayer, 1990), speech recognition (Shannon et al., 1995),and machine translation (Bahdanau et al., 2014).

As with most machine learning models, RNN trains by computing gradients at each layer, that is, by using backpropagation learning. Since we need to learn dependencies over time, we compute gradients by back-propagating through layers and by back-propagating through time (Werbos, 1990). An RNN

is typically hard to train because gradients vanish or explode (Pascanu et al., 2013). Multiplications of long matrix variables in gradient computation are responsible for exploding gradients. The vanishing gradient problem is the reverse; the gradients decrease in magnitude in the long chain.

Long-short term memory (LSTM)s (Greff et al., 2016) are RNNs that deal with vanishing/exploding gradient problems. LSTM maintains and regulates information flow over time using a cell state (c_t) and three gates.Cell states store long-term information, and gates decide which information should be written,read and erased to/from the cell state.

For a given input $x(t)$ at a given time t: **Input gate** (i_t) determines what parts of the new data should be written to **The cell**. **The Forget Gate**(f_t) determines what information will be forgotten from time $t-1$ The **Output gate**(o_t) decides which information in the cell should be written to the hidden state. Equation (2.5) shows a formula for these three gates, cell state c_t, and hidden state h_t. W is a set of trainable parameters related to hidden states h, and U is a set of trainable parameters related to input x.

$$
\begin{aligned}
f_t &= \sigma\Big(W_f h_{t-1} + U_f x_t + b_f\Big) \\
i_t &= \sigma\Big(W_i h_{t-1} + U_i x_t + b_i\Big) \\
o_i &= \sigma\Big(W_o h_{t-1} + U_0 x(t) + b_o\Big) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tanh\left(W_c h_{t-1} + U_c x_t\right) \\
h_t &= o_t \odot \tanh\left(c_t\right)
\end{aligned}
\tag{2.5}
$$

Convolutional Neural Network Another type of deep learning is the convolutional neural network (CNN), which processes data with a grid-like pattern, like images, videos, sound clips, and time series (Albawi et al., 2017). CNN's designed to handle 2D data such as images and videos are commonly known as 2D CNNs. CNN's designed to handle sequential data such as time series are called 1D CNN (Kiranyaz et al., 2021) or temporal convolutional network(TCN) (Lea et al., 2017). A key aspect of CNN is convolution, which is a linear mathematical operation between matrices (Dumoulin and Visin, 2016)

The convolution operation takes two functions f and g as inputs and returns

$f * g$ as the output, which can be expressed as an equation in integral form(2.6).

$$(x * w)(t) = \int_{-\infty}^{\infty} x(t)w(t-\tau)d\tau \tag{2.6}$$

In 2D CNN, (2.6) is a dot product between matrices, while in 1D CNN, it is a dot product between vectors. The following equation describes convolution as a matrix product(2.7).

$$\textbf{Let} \quad x(t) = \begin{array}{|c|c|c|}\hline x_{11} & x_{12} & x_{13} \\ \hline x_{21} & x_{22} & x_{23} \\ \hline x_{31} & x_{32} & x_{33} \\ \hline \end{array}, \quad \textbf{and} \quad w(t) = \begin{array}{|c|c|}\hline w_{11} & w_{12} \\ \hline w_{21} & w_{22} \\ \hline \end{array}$$

be two matrices, the convolution operation between them is

$$(x * w)(t) = \begin{array}{|c|c|c|}\hline x_{11} & x_{12} & x_{13} \\ \hline x_{21} & x_{22} & x_{23} \\ \hline x_{31} & x_{32} & x_{33} \\ \hline \end{array} * \begin{array}{|c|c|}\hline w_{11} & w_{12} \\ \hline w_{21} & w_{22} \\ \hline \end{array} = \begin{array}{|c|c|}\hline a_{11} & a_{12} \\ \hline a_{21} & a_{22} \\ \hline \end{array} \tag{2.7}$$

$$\begin{aligned} a_{11} &= x_{11}w_{11} + x_{12}w_{12} + x_{21}w_{21} + x_{22}w_{22} \\ a_{12} &= x_{12}w_{11} + x_{13}w_{12} + x_{22}w_{21} + x_{23}w_{22} \\ a_{21} &= x_{21}w_{11} + x_{22}w_{12} + x_{31}w_{21} + x_{32}w_{22} \\ a_{22} &= x_{22}w_{11} + x_{23}w_{12} + x_{32}w_{21} + x_{22}w_{33} \end{aligned}$$

CNN uses the term "convolutional layer" for the layer that implements convolution. As an example, x(t) represents input data(e.g. images) or the output of other layers, whereas wt is a learnable parameter **kernel**.

During the forward pass, the kernel slides across the height and width of the input matrix to produce a latent representation of the original data **strides** is the sliding size. The output shape of a $N \times N$ input and a $F \times F$ kernel with stride S is defined as follows:

$$\text{Output shape} = \frac{N - F}{S} + 1$$

When we add P amount of padding, the output shape becomes:

$$\text{Output shape} = \frac{N - F + 2P}{S} + 1$$

Besides convolutional layers, people often use pooling layers as well (Sun et al., 2017). The pooling layer is a downsampling layer used to control the overfitting and location sensitivity of convolutional layers outputs. Pooling layers enable us to summarize the characteristics detected in the input by the convolutional

layer. A slight change in the feature's location in the input detected by the convolutional layer will result in a pooled feature map with the feature in the same place. The pooling process allows the learned representations to be invariant to small translations. Invariance to translation simply means that most of the pooled outputs remain the same if we translate the input by a small amount.

Two common pooling layers are **Average pooling** and **Max pooling**.
Max pooling refers to selecting the maximum element from the feature map region covered by the pooling filter, as shown in figure 2.2.
The average pooling method computes the average of the elements present within the region of a feature map that is covered by the pooling filter, as illustrated in Figure 2.3.

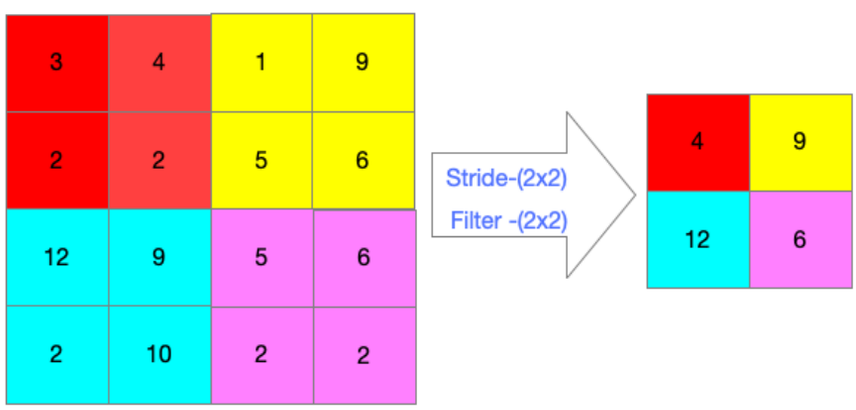

Figure 2.2: Example of 2D max-pooling: max-Pooling calculates maximum values for a region of a feature map.

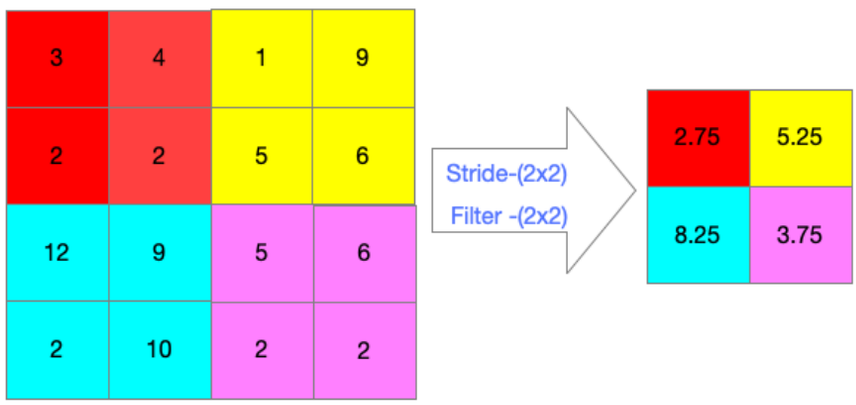

Figure 2.3: An example of 2D average-pooling: average-Pooling calculates the average value for a region of a feature map. pooling.

A temporal convolutional network (TCN) is a 1D version of the 2D CNN specialized for sequential data (Lea et al., 2017) . With TCNs, you get the advantages of both CNNs and RNNs. Like CNN, it can learn and recognize repeating patterns within input data. They can also process input sequences to produce output sequences similar to RNNs.

Two main principles distinguish TCN (Bai et al., 2018):

1. There is no information leakage between the future and the past as the convolutions are casual.

2. Any sequence can be mapped to a sequence of the same length by the architecture.

As a first principle, TCN applies casual convolutions, where output at a given time is convolved only with elements from that time frame and earlier layers. TCN uses a 1D fully-convolutional network (FCN) to accomplish the second principle. TCNs are thus a combination of causal convolution and FCNs. Simple casual convolutions have the disadvantage that they only look back on history, with the size linearly increasing as the depth of the network grows. In other words, their response field is increasing linearly as the number of layers increases. As a result, applying causal convolution to sequential tasks is difficult, especially when the history of the task is long. TCN uses dilated convolution

to solve this problem, which creates an exponentially large receptive field.

Autoencoders and Variational autoencoders In addition to the three main types of neural networks described above, autoencoders and variational autoencoders are commonly used to develop virtual sensors.
An autoencoder is a neural network that we can train without ground truth. We assume that for input $x \in R * N$, there is an unobserved **latent variable** $z \in R * d$ (d ** N) such that the mutual information between the input and the latent representation is maximum. Autoencoders rely on the **Encoder** and **Decoder** networks to map between input and latent representation. Encoder transforms input data $xinR * N$ into latent representations $zinR * d$, while Decoder transforms latent space into input data. To train the autoencoder, we minimize the reconstruction loss of the Decoder, i.e., make the output as similar to the input as possible.

Autoencoders that learn the probability distribution of data instead of mapping functions are called variational autoencoders (Kingma and Welling, 2013). Variational autoencoders assume the input data has a probability distribution (Gaussian) and then determine its parameters.

2.3.2 Review of Deep Learning Virtual Sensors

In industry 4.0 sensors, missing data is one of the most common problems. Therefore, in the spring semester of 2021, we developed variational autoencoder-based virtual sensors that estimate the missing value of physical sensors based on the information provided by the other physical sensors within the same process (Gutama, 2021).

The authors of (Guo et al., 2020) also used Variational autoencoder (VAE) to address missing data. Using variational autoencoders, they convert high-dimensional input sensors into relevant information and then estimate missing data based on the extracted data. First, variables are weighted by their correlation coefficient during VAE training. Then, to identify similarities between historical and deployment data, they use Kullback-Leibler symmetry. Then, they estimate the missing values by using EM algorithms.

One more VAE-based virtual sensor aimed at dealing with missing data is discussed in Xie et al. (2019). The virtual sensor framework is based on two VAE submodels. To begin, supervised deep variational autoencoders (SDVAE) learn latent distributions. Then, with the latent distribution learned from the first submodel, they construct a second submodel known as a modified deep variational autoencoder (MDVAE). Finally, The virtual sensor combines a

decoder from MDVAE with an encoder from SDVAE.

A virtual sensor called Dual Attention-Based Encoder-Decoder is proposed in (Feng et al., 2020). This virtual sensor uses Long Short-Term Memory (LSTM) networks and attention mechanisms to predict hard-to-measure process variables based on measurable variables. In particular, LSTM and attention can consider the spatiotemporal correlation between sensors.

The virtual sensor proposed in (Li et al., 2020) combines an LSTM network with normalized mutual information selection (NMIFS). LSTM is used in this model for handling highly non-linear dynamics of industrial time series. NMIFS is used to select the input variables for LSTM.

Jiang and Ge (2021) presents an augmented multidimensional convolutional neural network (CNN) virtual sensor that addresses unbalanced sampling, inaccurate matching, and partial missing industrial process data. The coarse-grained data is first generated by stitching process variables to fully retain industrial process information. Then, based on coarse-grained data, a CNN regression model is constructed.

Each of the virtual sensors approaches presented achieves high accuracy compared to different baseline models. However, They are poor at representing uncertainty. They are uninterpretable black-boxes, lacking in transparency, difficult to trust if they encounter an unexpected situation in input data.

2.4 Model Uncertainty

In machine learning, performance degradation most often results from a mismatch between training and deployment data (Chen et al., 2020). We use terms like Out-of-Distribution (OoD) samples, data shift, and distribution shift to describe mismatches between the training and deployment samples. The reasons behind Out-of-Distribution data are many. However, this thesis focuses on one type of such cause, sensor faults presented in previous section.

2.4.1 Types of Uncertainty

In machine learning, predictions will become uncertain when a model's input data differs during training and deployment. By uncertainty, we mean the confusion regarding the outcome of a decision. It is possible to distinguish between two kinds of uncertainty in deep learning models based on their sources- aleatoric and epistemic (Hüllermeier and Waegeman, 2021).

Aleatoric Uncertainty Aleatoric uncertainty (AU) from the Greek word "alea" meaning "to roll a dice" refers to the variability in the output of a model due to the stochasticity of input data (Tagasovska and Lopez-Paz, 2019). It measures the variance of the conditional distribution of the target variable based on input variables. Hidden variables or measurement errors cause this uncertainty, and collecting more data cannot reduce it.

Epistemic Uncertainty Epistemic uncertainty (EU) from the Greek word episteme, meaning "knowledge," describes errors caused by a lack of experience in our model at a particular region of feature space Tagasovska and Lopez-Paz (2019). The epistemic uncertainty is inversely proportional to the density of training examples and might be reduced by gathering data in low-density areas.

2.4.2 Estimation of Uncertainty

In machine learning, out-of-distribution samples will have high epistemic uncertainty, and more noisy samples will have high aleatoric uncertainty. Therefore the total model uncertainty, also called predictive uncertainty (PU) in predictive models, is equal to the sum of EU and AU.

Ensemble learning is a popular method for dealing with neural network models' uncertainties (Zhou et al., 2002). Ensemble learning involves building a series of independently trained models, as shown in Figure 2.4. These models have the same architecture, and the only difference between them is their weights. Each model contributes to the ensemble prediction, which is the average of all the models. The standard deviation of ensemble predictions is used to calculate the predictive uncertainty of models. Besides providing uncertainty estimates, ensemble learning improves model prediction accuracy. Moreover, ensemble learning has been shown to work well in evaluating predictive uncertainty, especially under dataset shifts (Ovadia et al., 2019).

However, this method has disadvantages. Ensemble learning ignores local uncertainty since its based on averaging different models (Fort et al., 2019). Therefore, the ensemble method may lead to overconfident decisions. In addition, Since deep neural networks have millions or billions of parameters, training and testing ensemble learning means processing and storing several data samples and models (Souza et al., 2016).

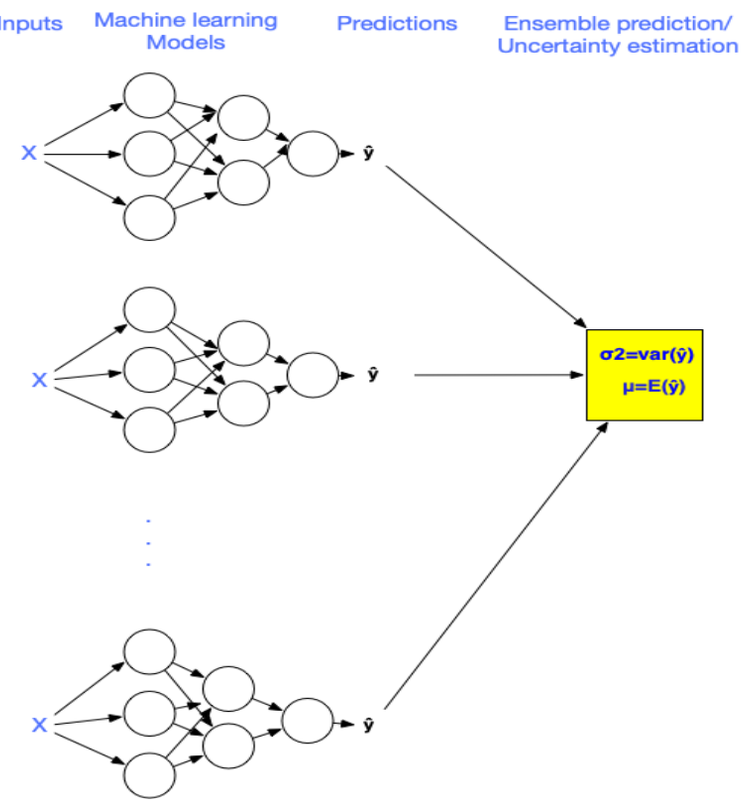

Figure 2.4: The assembly methods: Combine several models to achieve one optimal prediction model and uncertainty in the model.

Bayesian neural networks are an alternative to ensemble learning to estimate uncertainty. Bayesian neural networks combine deep learning with Bayesian probabilistic for proper uncertainty quantification. Probabilistic modeling offers a general framework for building systems that learn from data (Droguett and Mosleh, 2008).

Among the advantages of Bayesian neural networks are (Jospin et al., 2020):

- It is mathematically grounded and gives a better estimate of uncertainty.
- Preventing overfitting of the model. A Bayesian neural network is characterized by its weights expressed in the form of a probability distribution rather than as a deterministic value.
- It is capable of handling all types of uncertainty. The posterior distribution of weights can be learned by assuming a prior distribution for the network parameters. We can capture the epistemic uncertainty (uncertainty due

to model fitness) based on this distribution. By letting the model's output be a distribution, aleatoric uncertainty (the uncertainty due to noisy observations) can be captured.

- Having a single Bayesian neural network correspond to having infinitely many ensemble models.

The disadvantage of the Bayesian neural network is that it is more complex to implement, more expensive to compute, and more challenging to train than a standard neural network (Hernández-Lobato and Adams, 2015). Therefore, much of the current research directed at Bayesian neural networks is focused on determining techniques to make the training process easier.

2.5 Bayesian Neural Network

This thesis aims to explore the potential of the Bayesian neural network to generate uncertainty-aware virtual sensors for Industry 4.0. Therefore, we will cover the theoretical concepts behind standard Bayesian neural networks in this section.

Bayesian neural network (BNN) represents a probabilistic version of the standard neural network i.e., the weight and output of the neural network is considered random variables from a prior probability distribution, see Figure 2.5.

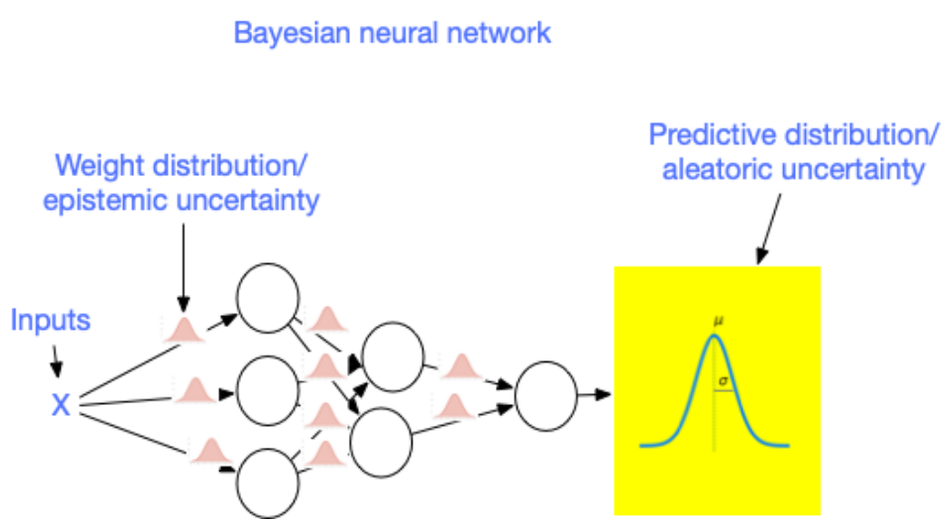

Figure 2.5: The Bayesian Network provides distributions of weights and outputs and explains the uncertainties associated with the model.

In formal terms, a Bayesian neural network (BNN) is a stochastic neural network trained using Bayesian inference (Jospin et al., 2020).

The Bayesian inference method uses Bayes' theorem to convert an existing **prior** belief into a **posterior** probability when more **evidence** or information becomes available. (Box and Tiao, 2011). Equation (2.8) and (2.9) gives Bayes' rule expressed differently.

$$\textbf{Posterior} = \frac{\textbf{Likelihood} \times \textbf{Prior}}{\textbf{Evidence}} \tag{2.8}$$

$$p(\textbf{hypothesis}|\textbf{data}) = \frac{p(\textbf{data}|\textbf{hypothesis})p(\textbf{hypothesis})}{p(\textbf{data})} \tag{2.9}$$

2.5.1 Approximating Parameters

A Bayesian neural network combines Bayesian inference with neural networks to derive the probability distribution of network weights,$\mathbf{w}$. Given the training data $\mathbf{D} = \{(\mathbf{x}^1, y^1), ...(\mathbf{x}^N, y^N)\}$. Direct application of (2.9), yields the posterior $p(\mathbf{w}|\mathbf{D})$:

$$p(\mathbf{w}|\mathbf{D}) = \frac{P(\mathbf{D}|\mathbf{w})p(\mathbf{w})}{p(\mathbf{D})} \tag{2.10}$$

In the numerator, $p(\mathbf{D}|\mathbf{w})$ represents the **likelihood** of the training data given a particular weight vector $\mathbf{w}$. The second term, $p(\mathbf{w})$, represents the prior distribution over the weights. It indicates our beliefs regarding weights before any evidence becomes available. The most commonly selected priors are Gaussians or uniform distributions. When each training data point is independent, $p(\mathbf{D}|\mathbf{w})$ becomes the product of the likelihood for each training point, as shown in (2.11).

$$p(\mathbf{D}|\mathbf{w}) = \prod_{i=1}^{N} p(y^i|\mathbf{w}, \mathbf{x}^i) \tag{2.11}$$

In the denominator of (2.10), $p(\mathbf{D})$ represents the distribution of the evidence (marginal distribution) and can be expressed as in (2.12)

$$p(\mathbf{D}) = \int p(\mathbf{D}|w)p(\mathbf{w})\, dw \tag{2.12}$$

In neural networks, the large number of weight parameters makes it computationally prohibitive to precisely calculate equation (2.12). Therefore, in most cases, the posterior functions $p([w])$ in equation (2.10) are intractable. To speed up and make the calculation more feasible, we should approximate the true posterior. There are two families of approaches to consider:

1. Sampling methods.

2. Variational inference.

Approximating Posterior Distribution of Parameters by Sampling Methods Markov Chain Monte Carlo (MCMC) is the most common sampling technique. Three most popular MCMC sampling methods are Metropolis-Hastings Algorithm (Chib and Greenberg, 1995), Hamiltonian Monte Carlo (Betancourt, 2017) and No-U-Turn Sampler (NUTS) Hoffman and Gelman (2014).

While sampling-based methods are guaranteed to find an asymptotically optimal solution, they have several crucial limitations (Blei et al., 2017).

1. They are computationally more expensive than variational inference when dealing with large datasets.

2. It is difficult to tell how close their solutions are to the actual distribution given a finite amount of time.

3. Sampling methods require selecting appropriate sampling techniques(e.g., proposal density in Metropolis-Hastings), so choosing sampling techniques represents a separate issue.

Variational inference is preferred for Bayesian neural network inference due to the shortcomings of MCMC.

Approximating Posterior Distribution Parameters by Variational Inference Variational inference approximates the intractable posterior by presenting the inference problem as an optimization problem (Hinton and Van Camp, 1993; Graves, 2011). To approximate the intractable posterior distribution, $p(\mathbf{w}|\mathbf{D})$ as close as possible, a simpler and tractable variational distribution is introduced, $q(\mathbf{w};\theta)$, parameterized by θ. The Kullback-Leibler divergence (KL) between p and q can be used to measure the closeness between p and q with respect to q (Van Erven and Harremos, 2014). The mathematical definition of KL divergence between p and q is given in (2.13).

$$KL(q||p) = E_q\left[\log \frac{q(\mathbf{w};\theta)}{p(\mathbf{w}|\mathbf{D})}\right] \quad \text{Where} \quad E_q \quad \text{is expectation with respect to q} \tag{2.13}$$

By applying the rule of conditional probability and logarithm, we can simplify equation (2.13) as equation (2.14).

$$KL(q||p) = \overbrace{(E_q[\log p(\mathbf{w},\mathbf{D})] - E_q[\log q(\mathbf{w};\theta)])}^{\text{ELBO}} + \overbrace{\log p(\mathbf{D})}^{\text{Constant wrt } E_q} \tag{2.14}$$

The objective is to make the variational distribution $q(\mathbf{w};\theta)$ as close to the target posterior distribution $p(\mathbf{w}|\mathbf{D})$ as possible by minimizing the KL divergence in equation (2.14). However, look at the last term of the equation (2.14). It is $\log p(\mathbf{D})$. The reason we are using variational inference is that we were unable to solve the intractable marginal of evidence, $p(\mathbf{D})$, which is given in equation (2.12). We are in the same situation again, and we cannot compute KL divergence directly. It turns out that KL divergence has two properties that work around this issue this time:

1. $KL(q||p) > 0, \forall\ q, p$

2. $KL(q||p) = 0$ if and only if $q = p$

Moreover, we see that $p(\mathbf{D})$ is constant with respect to variational probability $q(\mathbf{w};\theta)$ even if it's intractable. Rearranging equation (2.14) and using KL divergence properties, we obtain:

$$\begin{aligned}\log p(\mathbf{D}) &= \overbrace{(E_q[\log p(\mathbf{w},\mathbf{D})] - E_q[\log(\mathbf{w};\theta])}^{\text{ELBO}} - \overbrace{KL(q||p)}^{\text{KL divergence}} \\ &\geq \overbrace{(E_q[\log p(\mathbf{w},\mathbf{D})] - E_q[\log q])}^{\text{ELBO}}\end{aligned} \tag{2.15}$$

Minimizing the KL-divergence in equations (2.14) is equivalent to maximizing equations (2.15) also referred to as evidence lower bounds (ELBO).

2.5.2 Loss Function

In the same way as traditional neural networks, Bayesian neural networks are trained through stochastic gradient descent and backpropagation, that is, by selecting a loss function and optimizing weights. Equation (2.17), which is a rewrite of ELBO in (2.16), gives Bayesian neural networks' cost function :

$$\mathcal{J}(\mathbf{D}, \mathbf{w}) = -ELBO = E_q[\log q(\mathbf{w}; \theta)] - E_q[\log p(\mathbf{w}, \mathbf{D})] \tag{2.16}$$

This cost function is composed of two parts: First, $KL(q([w]\theta)p(w))$, which refers to as a prior dependent part, or the complexity loss. Its objective is to make the variational distribution diffuse (Mullachery et al., 2018). The second, $E_{q(\mathbf{w}|\theta)}[\log p(\mathbf{D})|\mathbf{w}]$, is a data dependant part, also referred to as the **likelihood** cost.

$$\mathcal{J}(\mathbf{D}, \mathbf{w}) = KL(q(\mathbf{w}|\theta)||p(w)) - E_{q(\mathbf{w}|\theta)}[\log p(\mathbf{D})|\mathbf{w}] \tag{2.17}$$

2.5.3 Backpropagation

It is computationally difficult to minimize the cost of (2.17). Thus, approximations are often made. Two general categories of the approximate method include:

1. Variational methods (Graves, 2011; Hinton and Van Camp, 1993).

2. Monte Carlo methods (Blundell et al., 2015).

Bayes-by-backprop (Blundell et al., 2015) is a popular method for getting a Monte Carlo estimate of a cost function in (2.16) using the usual backpropagation algorithm (Hecht-Nielsen, 1992). Essentially, the Bayesian neural network weights are random variables from the posterior distribution, $p(\mathbf{wD})$. Unfortunately, backpropagation does not function with random nodes. Bayes by Backprop uses the local reparameterization trick (Kingma et al., 2015) to resolve this problem. The key is to make the random variable input into our model, rather than something embedded inside it, so we never need to backpropagate through a random node. To accomplish this, we write the network weights in the following way.

$$\mathbf{w} = \mu + \sigma * \epsilon, \quad \text{where} \quad \epsilon \sim \mathcal{N}(0, 1). \tag{2.18}$$

In equation (2.18), $\theta = (\mu, \sigma)$ is variation parameter for posterior distribution of weights. We use (μ, σ) because variational distribution $q(\mathbf{w}|\theta)$ is often chosen to be gaussian. ϵ is a sample from standard gaussian distribution given to the

model as input.

Flipout Using this reparameterization method has the disadvantage that the sampled weight is the same for all training data because ϵ is sampled only once per batch for the sake of computational efficiency. This practice results in high variance of the gradient descent estimate and slows the convergence of the Bayes-by-Backprop method. There are several works proposed as variance reduction (Miller et al., 2017), but we are going to use the most recent work called Flipout (Wen et al., 2018). For this reason, we provide a brief description of the Flipout method in one paragraph.

Flipout is a general method of estimating gradients using weight perturbation. Therefore, It can be used with any method that utilizes weight perturbation. Using this method, it is possible to reduce the variance of gradient estimates by de-correlating the gradients between different training samples in the mini-batch. Let $\widehat{\Delta \mathbf{w}}$ represent the gradient estimates resulting from Bayes-by-Backprop, and let $\widehat{\Delta \mathbf{w}}$ be correlated within the mini-batch. First, Flipout utilizes a standard Gaussian distribution as its base noise. Randomly generated sign matrices are then multiplied by this base noise. For example, if r_n and s_n are two randomly drawn sign vectors, then the Flipout method can be written as follows:

$$\Delta \mathbf{W}_n = \widehat{\Delta \mathbf{w}} \circ (\mathbf{r}_n \mathbf{s}_n^T) \tag{2.19}$$

Here, n stands for a training example of the mini-batch. It has been shown that matrix multiplications with sign matrices decorrelate gradient estimates and decrease variance. These results contribute to an earlier convergence to optimal variational parameter values. Flipout achieves ideal linear variance reduction for fully connected, convolutional and recurrent neural network.

As a result, the Bayesian neural network weight can be expressed as the combination of mean weight $\overline{\mathbf{w}}$ plus the Flipout perturbation $\Delta \mathbf{w}$. Mathematically:

$$\mathbf{w} = \overline{\mathbf{w}} + \Delta \mathbf{w} \tag{2.20}$$

3

Methodology

This chapter describes methods and procedures for creating and evaluating uncertainty-aware virtual sensors. Section §3.1 will first provide an overview of our virtual sensor pipeline workflow and describe the tools used to manage it. Next, we will present different data transformation strategies to get data ready for use in section §3.2. We will then describe implementations and training methods for the virtual sensor in section §3.3. Next, in section §3.4 we will describe how we obtain an estimate of process variable and estimate of uncertainty from trained models. Finally, in section §3.5, we describe methods used to evaluate the predictive performance of the virtual sensor and the accuracy of uncertainty estimates made by the virtual sensor.

3.1 Workflow for building uncertainty-aware virtual sensors

We refer to the concept of uncertainty-aware virtual sensors as a model that estimates a process variable based on available hardware sensor measurements while also measuring its uncertainty.

The development of a virtual sensor begins with transforming raw data into a form suitable for modeling the relationship between the input and target. Next step is to develop a predictive model, train it, and evaluate it. In figure 3.1, we

visually represent our virtual sensor pipeline.

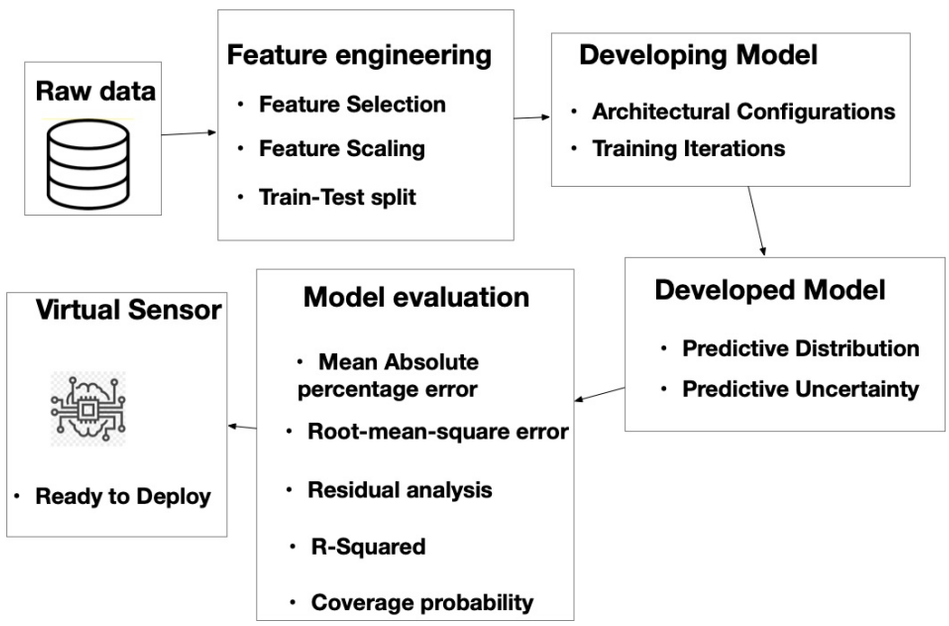

Figure 3.1: An overview of the workflow of the uncertainty-aware virtual sensor pipeline.

Initially, the raw dataset is preprocessed and made ready for use. The steps include selecting features, scaling, and splitting training and testing data. The "feature" is the term we use to describe a single sensor or dataset column. Thus, feature selection refers to the process of selecting sensors that provide the best estimates of the target variable. Scaling refers to the process of bringing all features to the same magnitude/range. Training test split is the process of splitting data into train and test to train the model on train data and test it on unseen test data. A detailed description of each feature engineering method will be presented in section §3.2.

Following feature engineering comes model development. During this phase, we will develop the virtual sensor models. During this step, the model's architecture is configured, the data is fed to the model in a certain way, training for a certain number of iterations, tuning the hyperparameters, and retraining again until a satisfactory result is achieved. We will describe this step in section §3.3.

Once a satisfactory result is obtained, we have a virtual sensor aware of uncertainty. In other words, the model that gives us an estimated distribution of the target variable, i.e., a predictive distribution, and an estimate of the

uncertainties in the prediction, i.e., a predictive uncertainty. The details of the predictive uncertainty and predictive distributions are explained in section §3.4.

Our final step of virtual sensor development is to evaluate the model against an unseen dataset. We will describe all methods of model evaluation in §3.5.

Programming languages and supporting software

Python We conduct all experiments and computations in Python (Van Rossum and Drake, 2009) in this thesis because:

- Python is an easy-to-learn, easy-to-use language that has straightforward syntax and code.
- Python is platform-independent, it can run on Windows, Linux, and macOS.
- Python is a highly stable, flexible language with many great tools.

The most popular deep learning tools in Python include:

- Pandas:Wes McKinney (2010) an advanced data structure and analysis tool.
- Tensorflow (Abadi et al., 2015), PyTorch (Paszke et al., 2019), Keras (Chollet et al., 2015),MXNet (Chen et al., 2015),and many more for building, training, evaluating, and deploying deep learning models.

Tensorflow We chose TensorFlow among all the available frameworks to build our model. Tensorflow is one of the leading deep learning models and modeling techniques developed by the Google Brain Team. TensorFlow's TensorFlow probability (Dillon et al., 2017) is one of the main reasons we chose TensorFlow. With TensorFlow probability, neural networks can make probabilistic decisions and perform statistical analysis. Using the TensorFlow probability library, we can build, train, and evaluate Bayesian neural networks similar to a standard neural network.

Data version control To develop a well-performing virtual sensor, we must construct multiple models with varying configurations, features, iterations, and

hyperparameter tuning until a satisfactory outcome is obtained. Additionally, we handle a large volume of data that goes through several preprocessing stages. We must track and measure all the changes we make in such a scenario to understand what has worked and what has not. Furthermore, it is necessary to go back to a specific version and view past results. We handle all of that with Data version control (DVC) (Kuprieiev et al., 2021). DVC allows us to track the different prepossessing stages of data, model hyper-parameter, and the model itself. DVC is an open-source command-line tool for managing datasets and machine learning models changes. DVC works alongside the Git repository and focuses on reproducible machine learning models.

The pipeline was developed for SINTEF Digital and in collaboration with them. We contributed to their pipeline called Erroneous data repair for Industry 4.0. It can be viewed under the bayesian branch here: https://github.com/SINTEF-9012/Erdre/tree/bayesian.

3.2 Data Feature Engineering

Feature engineeringTurner et al. (1999) is a data prepossessing technique to transform the raw data into some meaningful and understandable format to produce accurate prediction using a machine learning model.

3.2.1 Feature Selection

The data from Industry 4.0 processes contains a variety of variables. The previous section described two datasets, each consisting of over 47 variables. Not all variables will be helpful when generating virtual sensors for a given target variable based on other process variables. Certain variables are highly correlated with the target variable, while others are unimportant. Therefore, selecting the appropriate variables should be done carefully.

The issue of input selection in virtual sensor development has been widely explored, and several surveys on the topic have been conducted Curreri et al. (2020); May et al. (2011); Hira and Gillies (2015); Vergara and Estévez (2014). This study employs a simple and user-friendly approach of selecting inputs based on information generated by an open-source Python program called pandas. Pandas library offers a broad range of functionalities. The **Pandas profiling** function is one such function.

The Pandas profiling tool provides users with a descriptive statistical overview of all the features of the dataset and several features for generating reports.

These statistics are broken down into five categories to make them more understandable.

1. Overview The first category consists of three columns: Overview, Warnings, and Reproduction. The Overview column contains general information regarding variables, observations, missing cells, duplicate raws, duplicate raw percentages, total size, and different categories of variables (numerical, categorical). You will find warnings under the Warning column concerning missing values, correlation, and skewness of the variables. Finally, the reproduction column contains information related to report generation, such as the time taken to generate the report, when analysis began and ended, and the software version of pandas-profiling and the download option.

2. Variables The second category is variables, which presents a detailed analysis of each variable found in the dataset. These statistics include minimum and maximum values, percentiles, medians, ranges, and interquartile ranges. Also included in this category are descriptive statistics such as standard deviation, coefficient of variance, mean, skewness, and variance. A histogram that illustrates the frequency of variables is also provided.

3. Correlation Third, we have Correlation, which provides information regarding the degree to which variables are correlated using five different correlation coefficients. The correlation coefficients are **Pearsons's r**, **Spearman's**ρ, **Kendall's** τ , **Phik (**ϕ **k)** and **Creamer's V**.

4. Missing value The fourth category gives a visual representation of the missing values in the dataset.

4. Samples The last pandas profiling category is Sample, which displays the first and last ten rows from the dataset.

One of the most significant advantages of Panda's profiling report is that it is simple to use and easily integrated with other programs. We can save this report in JSON format, a standard data interchange format based on JavaScriptPezoa et al. (2016). JSON files are accessible for humans to read and write, and they are also accessible for machines to generate. Python comes with a built-in package called *json* that can automatically process datasets using pandas information.

We use Pearson's r, described in the pandas profiling report under category three, to determine the optimal input sensor. Pearson's coefficient Benesty et al. (2009) measures the statistical relationship between two random variables. Its value is between +1 and −1, where +1 indicates a positive relationship, −1

indicates a negative relationship, and 0 indicates no relationship. Pearson's correlation coefficient is independent of a unit of measurement; for example, if one variable is measured in kilometers and another in seconds, Pearson's correlation coefficient will not change. Moreover, Pearson's correlation coefficient is symmetric, meaning that the correlation between $X1$ and $X2$ or $X2$ and $X1$ is the same. We utilize Python scripts that return the correlation coefficients between our target variable and other variables from the pandas profiling report automatically.

3.2.2 Feature Scaling

A crucial part of data preprocessing is feature scaling. (Bhanja and Das, 2018). Scaling is a method to normalize the range of input features or target variable to a similar range. The effectiveness of any learning algorithm depends largely on the scaling method employed. Most deep learning models algorithms values features based on their similarity. This similarity is computed using what is called euclidean distance. The algorithm only takes in the magnitude of the features neglecting units. So For for example using a temperature value of 32 °C or 305 K have different effect. the feature with high magnitudes weight alot in the similarly calculation than features with low magnitude. Therefore we need to bring all features to the same level of magnitude to avoid these effects.

In our experiment, we use two popular scaler methods from scikit-learn, namely the Min-max Scaler and the Standard Scaler (Pedregosa et al., 2011) .

Min-Max Scaler A Min-Max Scaler is a scaling technique that shifts values between a given minimum and maximum value, often between 0 and 1. Equation (3.1) provides an expression for [0,1] scaling.

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{3.1}$$

The letter X represents a dataset feature, and the strings X_{min} and X_{max} represent that feature's minimum and maximum values, respectively.

Standard Scaler Standard scalers center values around the mean with a unit standard deviation. If $\mu = E[X], \sigma^2 = Var(X)$,then the formula for standard scaler becomes:

$$X_{scaled} = \frac{X - \mu}{\sigma} \tag{3.2}$$

$E[X]$ and $Var(X)$ describe the mean and variance of the feature X. We apply the Min-Max scaling method to the input features and the Standard scaling method to the target variables.

3.2.3 Data Split

Data splitting is a technique used to evaluate the performance of a machine learning algorithm. Data can either be divided into train-test data or train-validation-test data. A machine learning model is fitted based on training data and evaluated using test data. As the learning process proceeds, validation data validate the model. Figure 3.2 shows two methods for splitting data for a machine learning model. Pictured at the top is the train-test split. Seventy-five percent are used for training and 25 percent for testing. By splitting the data this way, one can specify how much of the 75 percent should be used for validation during training. For instance, if we have eight-hour time series data, we will use 6 for training + validation and 2 for testing. Alternatively, we can split data into training tests, and we can explicitly feed validation during this stage (see bottom Figure in 3.2) and validation data into the model during training. We use the train-test split for our experiments since the TensorFlow library takes validation data from training data, so we only need to specify the validation percentage.

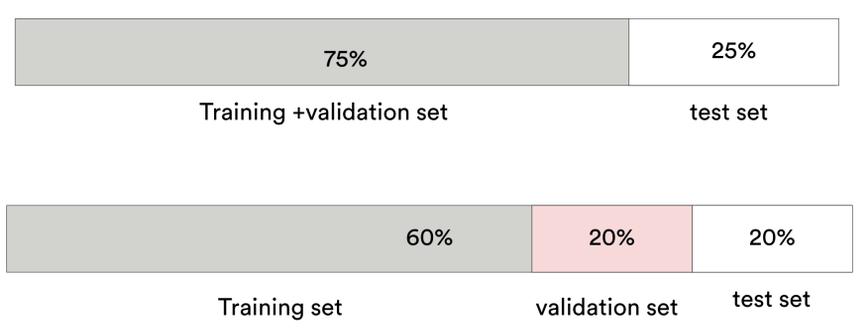

Figure 3.2: The different ways of splitting data for the purpose of training and evaluating models. Adapted from (Gutama, 2021).

3.3 Developing virtual sensors

Firstly, before defining the structure of our model, we offer a brief description of the task of building virtual sensors.

Sequence Modeling Most sensors deliver data as time series, which is sequential. We, therefore, need to use sequence modeling. Suppose we have $\mathbb{S} = s_1, s_2, ...s_N$ an array of sensors observing the same process parameters. Where N represents the total number of sensors. In each sensor, there is a sequence of time series of $s_i = x_1, x_2, ..., x_T$. Basically, the objective of virtual sensors is to predict some sensor output $s_j = y_1, y_2, ..., y_T$ from $\mathbb{S}$.

For long, recurrent neural networks (RNNs) have been the preferred method for modeling sequences. Our uncertainty-aware virtual sensors are developed using a convolutional neural network (1D CNN) more specifically Bayesian Temporal convolutional neural network. Here are the reasons why we chose 1D CNN rather than RNN, which has traditionally been used for sequence modeling.

Traditionally, recurrent neural network (RNN) is the default choice for sequence modelling. However, We develop Bayesian deep neural network using 1D convolutional neural network (1D CNN) implemented using tensorflow-probability library of tensorflow (Abadi et al., 2015). The reason we choose 1D CNN network rather than RNN which traditionally is the default choice for sequence modeling are:

1. RNNs learn only temporal feature dependencies, whereas 1D CNNs can extract both spatial and temporal feature dependencies.

2. Models of 1D CNN exhibit longer memory than LSTMs because they do not use the gating mechanism as RNNs, especially when applied to very long sequences.

3. The 1D CNN models do not have recurrent connections and are faster to train than RNN models (Oord et al., 2016).

4. Recently, a comparison of 1D CNN and RNN models demonstrated the superior performance of the 1D CNN model across a variety of sequence modeling tasks (Bai et al., 2018).

3.3.1 Architectural Configuration

Our model is implemented using tensorflow-probability library of tensorflow. Model architecture is illustrated in 3.3. We use Conv1D/Convolution1D interchangeably with 1D CNN because TensorFlow commonly refers to it as Conv1D/Convolution1D.

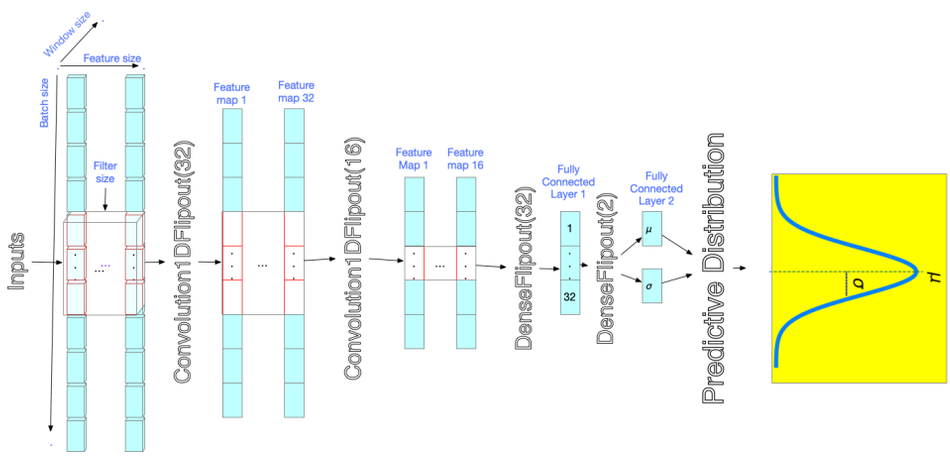

Figure 3.3: The Bayesian Temporal Convolutional Neural Network-based Virtual Sensor Model. Inside the parenthesis are the filter size of the convolutional layer and the number of nodes for the fully connected layer.

The model receives 3D input data in the input layer. The first dimension represents the number of batches, which is the number of sample processed before the model is updated, and the second represents the window size, which indicates how much history is to be used to predict the next target's distribution. The final dimension is the feature size, representing the number of input sensors. This 3D input goes through two Bayesian Conv1D layers, each with a Flipout estimate (as explained in §2.5.3). Two fully connected layers follow the two Conv1D layers. The final fully connected layers represent the mean and standard deviation of the predictive distribution. A "layer" in deep learning is a container that receives a weighted input, transforms it with a transformation function, and passes the result on to the next layer. Neural networks are made up of layers.

3.3.2 Training Iterations

Training refers to the process whereby the model gradually learns to accomplish the task to which it is assigned, which in our case is to learn to estimate unavailable process variables based on available sensor measurements.

Training iterations consist of a forward pass and a backward pass, which use Bayes-by-Backprop§2.5.3 and Flipout§2.5.3 to estimate the gradient. Based on a set of training data $D = \{\mathbf{x}_i, y_i\}_{i=1}^{N}$, the objective is to determine the minimum loss function. The loss function $\mathcal{J}(.)$ we minimize is the ELBO function described in detail in Section §2.5.2 and given by equation (2.17). The Monte Carlo method is used to approximate this cost function, as discussed in §2.5.3. Flipout is utilized to generate Monte Carlo samples, avoiding backpropagation through random nodes, as well as reducing variance.

In practice, thanks to Tensorflow, we don't need to make any mathematical calculations or approximations. So rather than dealing with mathematical details of implementation, we focus on tuning parameters and hyperparameters. Model parameters refer to configuration variables that can be estimated from data and are internal to the model. On the other hand, hyperparameters are configuration variables external to the model, and we cannot estimate their values from data. In our case, the only parameters are weights in the network; we explained how they are assessed in the background chapter. There are many types of hyperparameters, and here is a list of parameters we can adjust to training our model.

1D CNN Related Hyperparameters

Kernel Size Kernel Size is a number that determines the size of the kernel or the array of weights in a 1D convolutional neural network. For example, with a kernel size of 5, you will get a weight array of 5 values.

Filter Size Filter size determine the how many kernels to be used. One kernel learns specific feature in dataset.

Pooling Size Pooling size determines the region of feature space that will be covered by a pooling filter.

Optimization Related Hyperparameters

Loss function In training, we must calculate the difference between the predictions and the ground truth, called the model's loss. Consequently, we must choose a suitable loss function. The goal is to minimize losses.

Optimizer algorithms We also need to choose an optimizer algorithm that applies the gradient to the models' weights such that it leads to the minimization of the loss function. Again, TensorFlow has many algorithms available to choose from, including Adam, Stochastic gradient descent, and Adagrad.

Learning rate One can think of loss function as a curved surface and optimization problem as walking around that surface to find the lowest point. The learning rate determines the step size for each iteration to find the lowest point. Too big step size leads to local minima, which means the model can't find the lowest point in the loss surface. On the other hand, too small learning-rate will slow down the learning process. Therefore it is vital to choose the learning rate correctly.

Batch size Batch size is a hyperparameter that specifies how many data samples to compute before the loss is calculated. It is typically between 1 and the size of the training data.

Epoch Epoch hyperparameters specify how many times the learning algorithm will run through a set of training data.

When all hyperparameters are set, the model is ready for training. The following steps summarize the training step:

Step 1: Training data (features and ground truth label) is fed into the model.

Step 2: Every batch will be iterated over in a single epoch.

Step 3: The model predicts and compares it with the ground truth label, then calculates and reports the loss.

Step 4: The model uses the optimization algorithm to update the models' parameters(weights).

Step 5: Then repeat the steps for all epochs and report the final loss.

If we are not satisfied with the result of the trained model, we change some of the hyperparameters and retain them until we are happy.

3.4 Developed Virtual Sensors

Once the model has been trained and satisfied with the performance on training datasets, the next step is to predict an unseen dataset. We also use our model to estimate the prediction uncertainty since our model is uncertainty-aware. As our model is probabilistic, we call it the predictive distribution and the predictive uncertainty. The following paragraphs will describe how we get predictive distributions and uncertainty.

3.4.1 Predictive Distribution

Our models' output is not a single prediction of the ground-truth label; instead, it is a prediction/estimate of the probability distribution of the ground truth label. The mean of this distribution should approximately be the same as our ground truth sample. Therefore we obtain the prediction/estimate of the process variable by taking a sample from this distribution.

3.4.2 Predictive Uncertainty

A Bayesian neural network has the advantage of estimating uncertainty in a simple way and managing all kinds of uncertainties. Our model output is based on the Gaussian distribution. Accordingly, the $\mathcal{N}(\mu, \sigma)$ is the predictive distribution for our model. μ is a model's mean prediction, and σ may be viewed as an estimate of aleatoric uncertainty. We can calculate epistemic uncertainty by averaging the results of N forward passes of the Bayesian neural network. Instead of taking a single σ as an estimate of aleatoric uncertainty, we compute the aleatoric uncertainty (AU) and epistemic uncertainty (EP) simultaneously by averaging the output of the models. Let $(\mu_i, \sigma_i)_{i=1}^{N}$ be models' output at forward-pass i.

$$\mathbf{EU} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(\mu_i - \overline{\mu})^2} \quad \text{where} \quad \overline{\mu} = \frac{1}{n}\sum_{i=1}^{N}\mu_i \tag{3.3}$$

The formula (3.3) may seem complected, but it is straightforward. We take many samples from our model distribution and compute the standard deviation for these samples. This method is similar to ensemble ways of computing uncertainty. The only difference is we are using only a single model in contrast to ensemble methods. Using this approach would be equivalent to using an

ensemble of an uncountable infinite number of neural networks (Blundell et al., 2015). We compute aleatoric uncertainty by simply averaging overall σ's.

$$\mathbf{AU} = \frac{1}{N}\sum_{i=1}^{N}\sigma_i \tag{3.4}$$

The total predictive uncertainty (PU) can be accurately predicted by the superposition of these to uncertainties:

$$\mathbf{PU} = EU + AU \tag{3.5}$$

3.5 Model Evaluation

The final step in developing an uncertainty-aware virtual sensor model is to evaluate and validate the predictive distribution and predictive uncertainty accuracy. To validate the generated virtual sensor, we will perform a residual analysis. For evaluation of predictive model performance, we use three different metrics, namely root mean square error (RMSE), mean absolute percentage error (MAPE), and R-squared (R2). These are all standard metrics in statistics and machine learning. Considering that sensor data (time series data) exhibit a variety of cycle, trend, and seasonal characteristics, we utilize three metrics. Different metrics may be appropriate for different behaviors. We believe that three metrics taken together can provide a comprehensive picture of the quality of a model. Therefore, we use all three metrics. Finally, we measure coverage probability to assess the accuracy of our prediction uncertainties. In the following paragraphs, we describe residual analysis, three performance evaluation metrics, and coverage probabilities.

3.5.1 Residual Analysis

A residual is simply the difference between the ground truth labels and the predicted values (Cook and Weisberg, 1982). Analysis of residuals plays a key role in validating a predictive model since it provides information about its quality. A residual with an expected value that is not near zero implies that the model is asymptotically biased. In situations where the residual contains patterns, the model is inconsistent, i.e., it does not explain some of the data

relationships. Whenever the residual variance is not constant, it indicates that a particular part of the model has a different predictive power. A good predictive model should therefore possess the following characteristics (Verran and Ferketich, 1984):

- The expected value of the residual should be zero.
- The variance of the residual should be constant.
- Residuals should not be auto-correlated.
- Residuals should be normally distributed.

3.5.2 Root Mean Squared Error

The root mean square error (RMSE) is defined as the square root of the mean square error (MSE), as shown in the equation (3.6). In the case of unbiased and normally distributed errors, RMSE is more appropriate than MSE (Chai and Draxler, 2014). The other advantage of RMSE over MSE is that RMSE does not use absolute values, which are often undesirable in mathematical calculations. The disadvantage of RMSE is that it is sensitive to outliers. The RMSE tends to become larger than MSE with $n^{\frac{1}{2}}$ as distribution error magnitudes become more variable (Willmott and Matsuura, 2005).

$$RMSE(y, \bar{y}) = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}} \tag{3.6}$$

3.5.3 Mean Absolute Percentage Error

The mean absolute percentage error (MAPE) is the difference between actual value and model prediction, divided by the absolute true value and summed for each prediction point as shown in equation (3.7)). It has become increasingly popular due to its intuitive interpretation of relative error (De Myttenaere et al., 2016). Additionally, it is sensitive to relative error and does not change when actual values are scaled. It does, however, have a few disadvantages, including:

- An actual value of y with a single zero value will lead to a division-by-zero problem. A standard solution to this problem is to add a small but strictly positive number ϵ to the numerator to avoid undefined results when y is zero (Pedregosa et al., 2011).

MAPE expression becomes $\frac{\sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{\max(\epsilon, |y_i|)})}{n}$.

- The MAPE has no upper limit, i.e., MAPE can exceed 100 percent. Unfortunately, many people misunderstand this concept (Tayman and Swanson, 1999).

$$MAPE(y, \bar{y}) = \frac{\sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{|y_i|}}{n} \tag{3.7}$$

3.5.4 R-squared

R-squared (R2) is defined as one minus the sum of squares residual divided by the total sum of squares in statistical analysis (Figueiredo Filho et al., 2011). This measure represents the percentage of variance explained by independent variables with respect to the dependent variable. The equation (3.8) expresses R2 mathematically. In cases where the model predictions $\hat{y}$ exactly match the true value y, then $R2 = 1$ since the sum of squares of residuals $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = 0$. $R2 = 0$ if the model predictions $\hat{y}$ match the mean of values $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ exactly, since $\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2} = 1$. In the case where the model predictions $\hat{y}$ are worse than $\bar{y}$, then $R2$ will have a negative value.

$$R2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2} \tag{3.8}$$

Where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$

3.5.5 Coverage Probability

The coverage probability measures the accuracy of a confidence/prediction interval. In theory, future observations $y*$ are expected to lie within a given confidence interval with a prescribed probability$(100(1-\alpha)\%)$ (Khosravi et al., 2010). The α represents a significance level (Salkind, 2006). Generally, coverage probability for prediction intervals approaches a nominal confidence level. Thus, if our uncertainty estimates are accurate, we would expect precisely 95% of our test data to fall within the predictive distribution's inner 95% confidence/prediction interval. The coverage probability could either be less than or greater than the nominal confidence level if the uncertainty estimate is not perfect. An interval with a greater coverage probability than the nominal confi-

dence level is considered conservative, while one with less than the prediction interval is deemed to be anti-conservative (Olive, 2007). Conservative intervals are under-confident of their prediction, while anti-conservative intervals are overconfident.

We determine the coverage probability of our uncertainty interval using the simulation method. This method involves the following three steps:

Step 1: We simulate n samples based on the predictive distribution. For instance, $n = 1000$.

Step 2: Based on a given confidence level, we compute the lower and upper percentiles.

Step 3: We compute the proportion of ground truth labels that fall within the interval between the lower and upper percentiles of samples.

4

Experiments and Results

This thesis's predefined objective is to investigate the following research questions.

RQ1 How can we build and validate uncertainty-aware virtual sensors using a Bayesian neural network?

RQ2 How can we evaluate the accuracy of the uncertainty estimated by the Bayesian neural network?

RQ3 How well is uncertainty estimation able to detect single sensor faults?

a) **Faults due to drift?**

b) **Faults due to bias?**

c) **Faults due to freezing?**

d) **Faults due to precision degradation?**

Creating a robust and uncertainty-aware virtual sensor is the first step toward answering these equations. Identifying the input and target variables, training the Bayesian convolutional neural network, and evaluating the model's performance is part of this step. The second step involves estimating model un-

certainty and measuring accuracy of the method used to estimate uncertainty. An experiment based on fault injection analysis is the final step toward addressing the research questions. Therefore, in this chapter ,first we describe dataset we use to d to generate virtual sensors and conduct experiments §4.1, then we will, we will experimentally perform the three mentioned steps, present results, provide statistical data and discussion to support our results in §4.2 and §4.3. In section §4.4, we will provide a comprehensive summary of all results.

4.1 Dataset Description

This thesis uses two distinct datasets from the industrial manufacturing process. First, we have the CNC Mill Tool Wear dataset from the Manufacturing and Automation Research Testbed at Michigan. Second, we have the Tennessee Eastman Process Simulation Dataset, the Eastman Chemical Company's realistic industrial process data. The following sections provide a summary of them.

4.1.1 CNC Mill Tool Wear Dataset

The CNC Mill Tool Wear Data (CNC) is a series of time-series data obtained from CNC milling machines installed at the Systems-level Manufacturing and Automation Research Testbed (SMART) at the University of Michigan. This dataset is currently available on Kaggle (Sun, 2021). This dataset consists of experiments using two-by-two-by-five-inch wax blocks with varying tool conditions, feed rates, and clamping pressures. In each experiment, a wax part was made with an S-shaped top face, as shown in Figure 4.1. Each experiment's metadata is summarized in Table4.1.The feed rate variable indicates the relative velocity of the cutting tool. The clamping_pressure refers to the force used to hold the workpiece in a vice. Machining_finalized indicates whether machining was completed without removing the workpiece from the pneumatic vise. The dataset contains 48 different sensors. Adopted from (Sehee_Lee, 2021), Table 4.2 shows list of sensors in this dataset, and their description and unit of measurement.

Figure 4.1: A wax block that has an S-shaped curve to it. The wax block is a part of the CNC Mill Tool Wear dataset.Sun (2021)

Table 4.1: Overview of the metadata collected for the eighteen CNC mill tool wear experiments.

Experiment number	Number of row	feedrate (mm/s)	clamp _pressure (bar)	tool _condition	machining _finalized
1	1056	6	4	unworn	yes
2	1669	20	4	unworn	yes
3	1522	6	3	unworn	yes
4	533	6	2.5	unworn	no
5	463	20	3	unworn	no
6	1297	6	4	worn	yes
7	566	20	4	worn	no
8	606	20	4	worn	yes
9	741	15	4	worn	yes
10	1302	12	4	worn	yes
11	2315	3	4	unworn	yes
12	2276	3	3	unworn	yes
13	2234	3	3	worn	yes
14	2333	3	3	worn	yes
15	1382	6	3	worn	yes
16	603	20	3	worn	no
17	2151	3	2.5	unworn	yes
18	2254	3	2.5	worn	yes

Table 4.2: Available variables of CNC tool wear dataset, their definitions and unit of measurements

Variables	Definition	Unit
X1_ActualPosition	actual x position of part	mm
X1_ActualVelocity	actual x velocity of part	mm/s
X1_ActualAcceleration	actual x acceleration of part	mm/s^2
X1_CommandPosition	reference x position of part	mm
X1_CommandVelocity	reference x velocity of part	mm/s
X1_CommandAcceleration	reference x acceleration of part	mm/s^2
X1_CurrentFeedback	current	A
X1_DCBusVoltage	voltage	V
X1_OutputCurrent	current	A
X1_OutputVoltage	Voltage	V
X1_OutputPower	Power	kW
Y1_ActualPosition	actual y position of part	mm
Y1_ActualVelocity	actual y velocity of part	mm/s
Y1_ActualAcceleration	actual y acceleration of part	mm/s^2
Y1_CommandPosition	reference y position of part	mm
Y1_CommandVelocity	reference y velocity of part	mm/s
Y1_CommandAcceleration	reference y acceleration of part	mm/s^2
Y1_CurrentFeedback	current	A
Y1_DCBusVoltage	voltage	V
Y1_OutputCurrent	current	A
Y1_OutputVoltage	voltage	V
Y1_OutputPower	power	kW
Z1_ActualPosition	actual z position of part	mm
Z1_ActualVelocity	actual z velocity of part	mm/s
Z1_ActualAcceleration	actual z acceleration of part	mm/s^2
Z1_CommandPosition	reference z position of part	mm
Z1_CommandVelocity	reference z velocity of part	mm/s
Z1_CommandAcceleration	reference z acceleration of part	mm/s^2
S1_ActualPosition	a actual position of spindle	mm
S1_ActualVelocity	actual velocity of spindle	mm/s
S1_ActualAcceleration	actual acceleration of spindle	mm/s^2
S1_CommandPosition	reference position of spindle	mm
S1_CommandVelocity	reference velocity of spindle	mm/s
S1_CommandAcceleration	reference acceleration of spindle	mm/s^2
S1_CurrentFeedback	spindle current	A
S1_DCBusVoltage	spindle voltage	V
S1_OutputCurrent	spindle output current	A
S1_OutputVoltage	spindle output voltage	V
S1_OutputPower	spindle power	A
S1_SystemIntertia	torque intertia	kgm^2
M1_CURRENT_PROGRAM_NUMBER	Number the program is listed under on the CNC	float64
M1_Sequence_number	Line of G-code being executed	float64
M1_CURRENT_FEEDRATE	instantaneous feed reate of spindle	float64
Machining_Process	The current machining stage being performed. Including preparation, tracing up and down the "S" curve involving different layers	object

4.1.2 Tennessee Eastman Process Simulation Dataset

Tennessee Eastman Process Simulation Dataset (TEP) is a well-known public dataset based on realistic industrial processes provided by Eastman Chemical Company for process monitoring and control studies Downs and Vogel (1993).

This process creates two liquid products from four gaseous components: A, C, D, and E, as well as an inert B and byproduct F Reinartz et al. (2021). Chemical reactions occur in five different units (reactor, condenser, compressor, separator, and stripper). In the reactor, gaseous components (A, C, D, and E) are transformed into liquid products (G and H). Condensers cool gaseous products coming out of reactors. Separators separate gas and liquid components of products. Compressors recirculate gas streams into the reactor. It is the stripper's responsibility to separate the two products (G and H) from any unrelated feed components. Figure 4.2 shows a high-level view of the process.

Data is collected using over 40 sensors to monitor flow rates, pressure, temperature, mole fractions, compressor power, etc. Moreover, operators manipulate more than ten quality variables to ensure that the chemical process runs smoothly. In the dataset which is available at Rieth et al. (2017) are 52 variables, of which 41 are sensor measurements(XMEAS_1 - XMEAS_41) and 11 are manipulated quality variables(XMV_1 - XMV_11). A description for manipulated variables are provided by Table 4.3

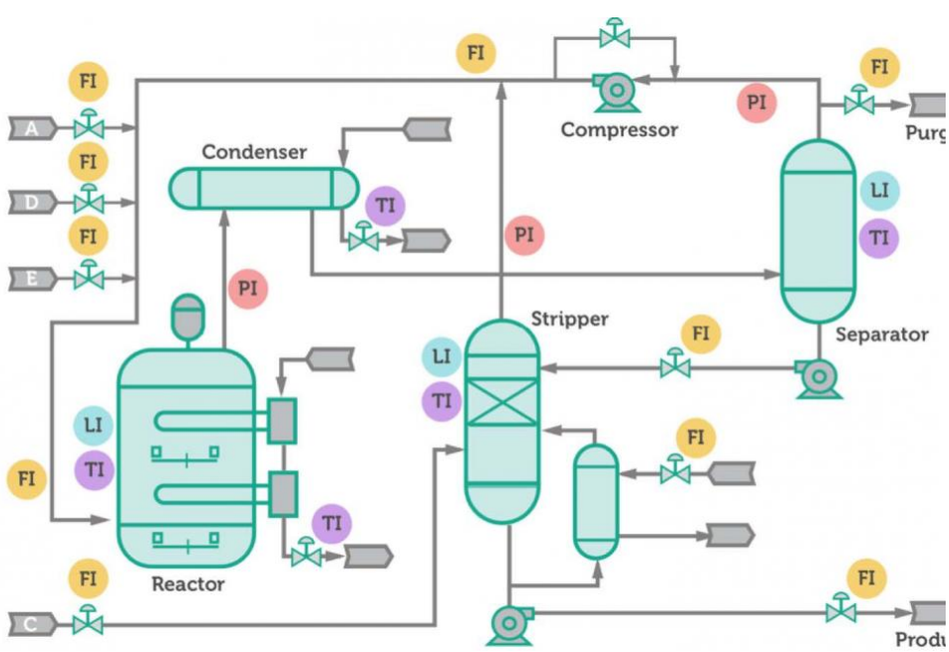

Figure 4.2: Tenessee Eastman (TE) process flow-sheet. Adopted from Chen (2019)

Table 4.3: Manipulated variables

Variable	Description
XMV_1	D Feed flow(stream 2)
XMV_2	E Feed flow (stream 3)
XMV_3	A Feed flow (stream 1)
XMV_4	A and C Feed flow (stream 4)
XMV_5	Compressor recycle valves
XMV_6	Purge valve (stream 9)
XMV_7	Seperator pot liquid flow (stream 10)
XMV_8	Stripper liquid product Flow
XMV_9	Stripper Steam Valve
XMV_10	Reactor cooling water flow
XMV_11	Condenser cooling water flow

4.2 Experiment 1: Building Uncertainty-aware Virtual Sensors

This section provides an answer to our first and second research questions:
RQ1: How can we build uncertainty-aware virtual sensors?
RQ2: How can we evaluate the accuracy of the uncertainty estimated by the Bayesian neural network?

We answer **RQ1** thought through three experimental steps, which we will present its result in the following three subsections of this section. We answer **RQ2** by experimentally evaluating predictive uncertainty. We will present the result of our experiment in the fourth subsection of this section.

4.2.1 Input selecting

As described in §4.1, we use CNC and TEP data to generate virtual sensors. The first step and a critical issue in virtual sensor building concern selecting input variables among those available in a dataset.

Although we have trained models with different target data, we will only present one target variable per data since we have only performed fault injection analysis using these particular target variables. In CNC data, we use the X1_ActualPosition variable as the target variable, as it is the first variable in the dataset and contains many correlated variables. We use the method described in §3.2.1 to determine the most correlated features. The threshold of absolute

Pearson's coefficient was set at 0.5, which indicates that for a correlation between the target variable and other variables to be accepted, the correlation should be > 0.5 or < -0.5. Fourteen variables met this requirement for the target variable X1_ActualPosition, as shown in Figure 4.3

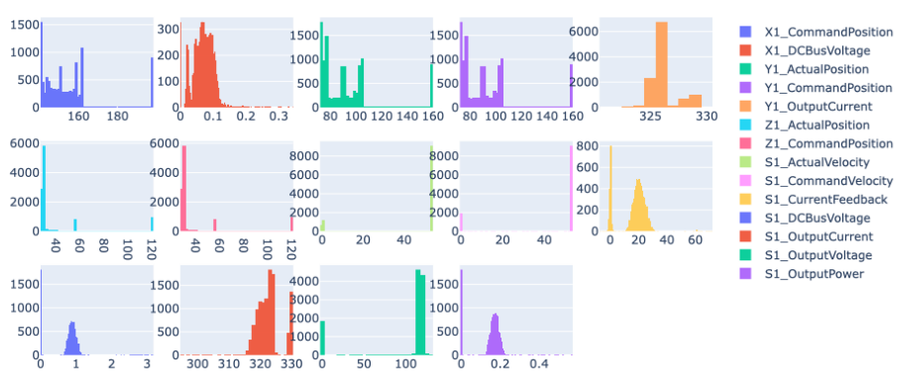

Figure 4.3: Distributions of selected input features for the CNC dataset. All of these features meet our requirement of at least 50 percent correlation to target variables.

For TEP data, we chose XMV_9 as our target variable. It is not a sensor measurement but rather a manipulated quality variable corresponding to the Steam Valve of the Stripper. Only four sensors met the requirement of a correlation coefficient of at least 0.5 for this target variable, as shown in Figure 4.4

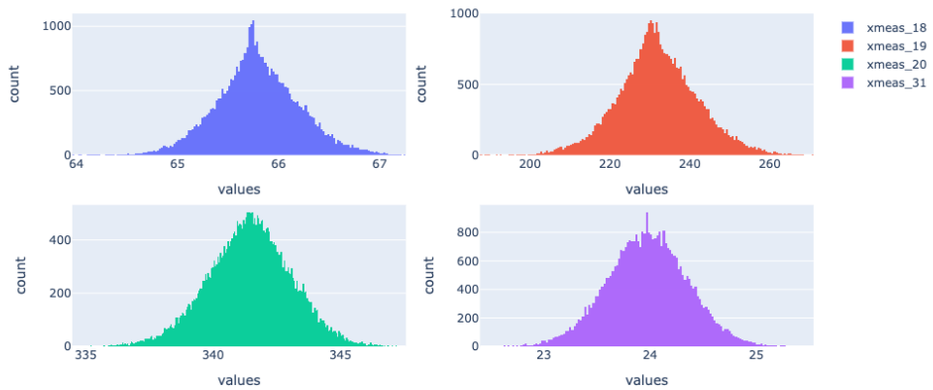

Figure 4.4: Distribution of selected input features based on correlation to target data for TEP dataset. All of these features have a correlation to target data above 50%.

4.2.2 Model Training

For our virtual sensor, we divided the dataset into train and test. Five of the CNC dataset's 18 experimental datasets are considered test datasets. The remaining thirteen are divided into 75 percent training and 25 percent validation datasets. Validation data is used after each training epoch, whereas test data is used only once after the final models have been developed.

In total, the TEP dataset contains 124999 individual data-points. For training and validation, we use around 87500, and for the test, 37499. The training-validation ratio is similar to the CNC dataset.

Hyperparameters such as learning rate, history size of the input dataset, batch size of mini-batch, and kernel size represent the weight in a convolutional neural network that influences the network's learning ability. In this regard, selecting the hyperparameters of a model is a critical step.

We train the model using the ADAM optimizer with mini-batch gradient descent. We found the optimum batch size to be between 30 and 50. A smaller batch size is computationally slower, while a larger one is quicker, resulting in less accurate models. Furthermore, through trial and error, we discovered that 0.001 is a reasonable learning rate. This parameter determines how fast the model will learn. We can evaluate how much-apportioned error is included in the model's weights when they are updated by controlling this parameter. The

model learns faster with a high learning rate but results in suboptimal weights. The model may learn more optimal weights or even globally optimal weights with a slower learning rate, but training will be significantly longer.

Loss functions are more challenging hyperparameters in Bayesian neural networks because we have to write custom losses. In §3.3.2, we discussed that we have two types of loss functions: a data-dependent loss and a prior-dependent loss. As the prior dependent loss, we use Kullback-Leibler divergence. Using this loss function, we seek to regularize the difference between prior weight beliefs and estimated weight posteriors. Since this loss is not dependent on the data, we evaluate it per layer. Even with mini-batch training, we found that this loss must be scaled by a factor of 1/size of training data. A learning process may become unstable if not scaled this way due to the significant variance in weight. We use the Negative Log-Likelihood as the data-dependent loss function, which minimizes the difference between the ground truth distribution and the predicted distribution. Because this loss is dependent on the data, it is evaluated after each forward pass. All hidden layers use ReLU, and the output layer uses linear activation. We used a kernel size of 5 for all convolutional layers.

4.2.3 Evaluating Predictive Distribution

To evaluate the goodness-of-fit of our Bayesian convolutional neural network-based virtual sensor, we want to use different evaluation metrics.

Firstly, we evaluated the accuracy of our Bayesian CNN model using R-Squared, RMSE, and MAPE. In addition to the Bayesian CNN model, we also trained the non-Bayesian CNN model and the LSTM network to compare our model with non-Bayesian models. Non-Bayesian models have the same parameters as Bayesian models.

Compared to non-Bayesian models, the Bayesian convolutional neural network-based virtual sensor produces reasonably accurate and reasonable results. Table 4.4 presents the complete comparison between Bayesian and non-Bayesian performance. As a reminder, R2 scores range from zero to one, one being the best and zero being the worst, whereas the other two matrices are inverse, zero being the best and one being the worst.

All models achieve R2 values greater than 0.90 in both datasets. The Bayesian model has the highest R2 score among all the models, with R2 = 0.99 for CNC data and R2 = 0.98 for TEP data. On CNC data, the LSTM model has the lowest R2 of 0.90. Bayesian models have the worst RMSE, with RMSE = 0.40 on CNC data and RMSE = 0.33 on TEP data. Both non-Bayesian models achieve a good

RMSE score on CNC data but worse on TEP data. MAPE scores for all models are similar and small. We can see that relying on a single metric may lead to an inaccurate assessment of reality.

Table 4.4: The predictive performance of virtual sensors based on Bayesian convolutional neural networks and virtual sensors based on non-Bayesian convolutional neural networks.

Network	Dataset	R2	RMSE	MAPE
Bayesian CNN	CNC data	0.9992	0.40	0.0017
	TEP Data	0.9839	0.33	0.005
CNN	CNC data	0.961	0.0027	0.03
	TEP Data	97.3	0.426	0.0073
LSTM	CNC data	0.99	0.0016	0.05
	TEP Data	0.901	0.52	0.0144

To understand why the performance on evaluation matrices is so different, we should plot and visualize the ground-truth value against the predictive distribution. To reduce the number of visualizations, we only provide visualization for Bayesian models. We first draw a sample from the predictive distribution and plot it with the distribution of the ground truth label.

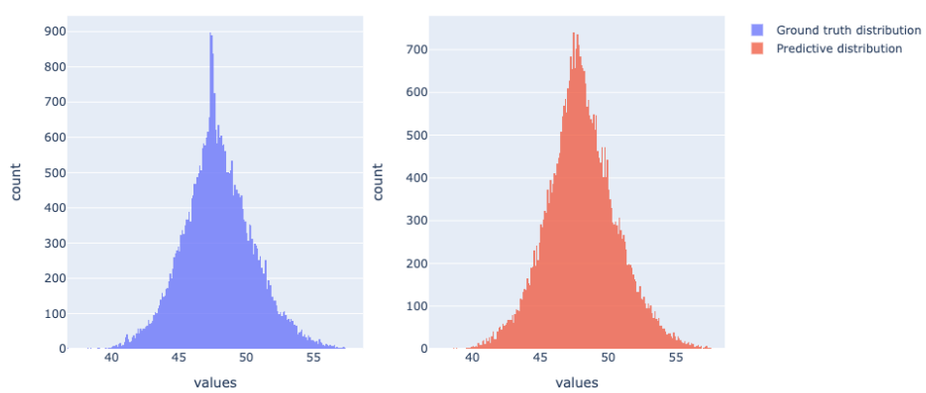

Figure 4.5: The distribution of ground truth labels vs. mean predicted values for a model trained on TEP dataset

The Figure 4.5 presents the distribution of predictive values on TEP data in conjunction with ground truth values. We can see that the distributions are close to identical; however, the predicted distribution has a bit more variance.

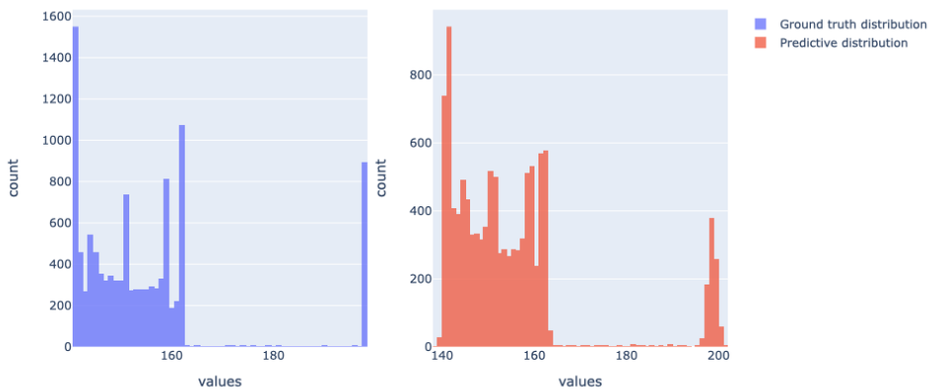

Figure 4.6: The distribution of ground truth labels vs. mean predicted values for a model trained on CNC dataset

In Figure 4.6, we can see predictive distribution on CNC data together with the ground-truth label distribution. Although the distributions appear similar, the variance in the predictions is more evident in this instance. To some extent, this Figure can explain why the RMSE score on this dataset is much higher than the RMSE score on the TEP dataset.

Lastly, we evaluated the model using residual analysis. Recall that the residual of a good model has the following characteristics:

- The expected value of the residual should be zero.
- The variance of the residual should be constant.
- Residuals should not be auto-correlated.
- Residuals should be normally distributed.

Figure 4.7 a shows the residual of the model trained on TEP data. In this residual, we observe that all four characteristics are present. The Figure 4.8 shows the residual for the model trained on CNC data. We can see that the residual is normally distributed, with zero expected value; however, the residual

variance is not constant, indicating that a particular segment of the model has a different predictive capability. This residual may help explain the high RMSE score.

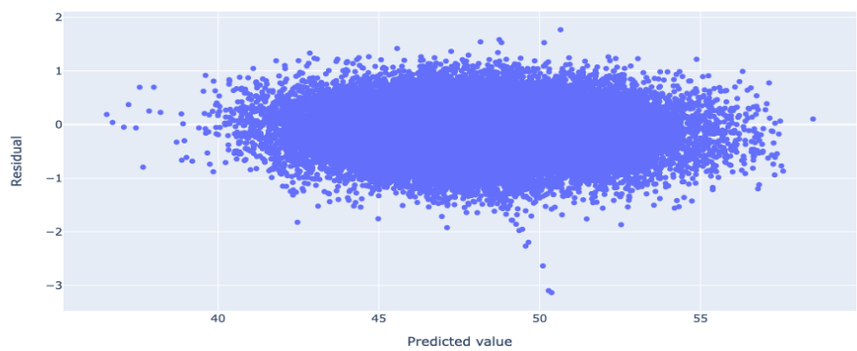

Figure 4.7: The residual plot for residual vs predicted values using the TEP dataset.

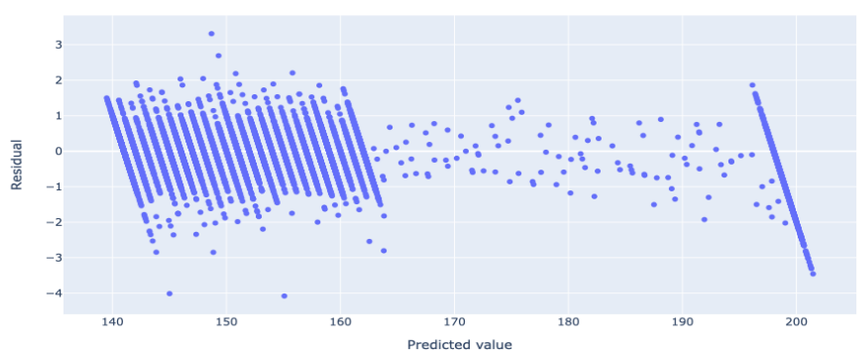

Figure 4.8: The residual plot for residual vs predicted values using the TEP dataset.

Based on the different evaluation matrices we have used, we conclude that the Bayesian neural network is not perfect but it performs at least as well as non-Bayesian neural networks for prediction.

4.2.4 Evaluating Predictive Uncertainty

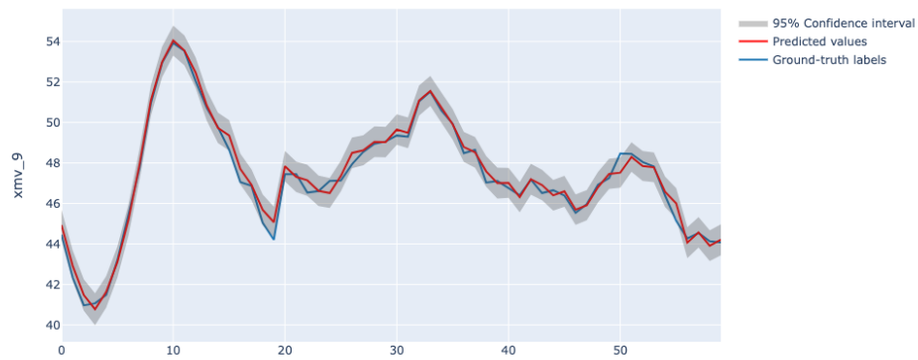

Figure 4.9: A visualization of ninety-five percent prediction interval for a model trained TEP dataset.

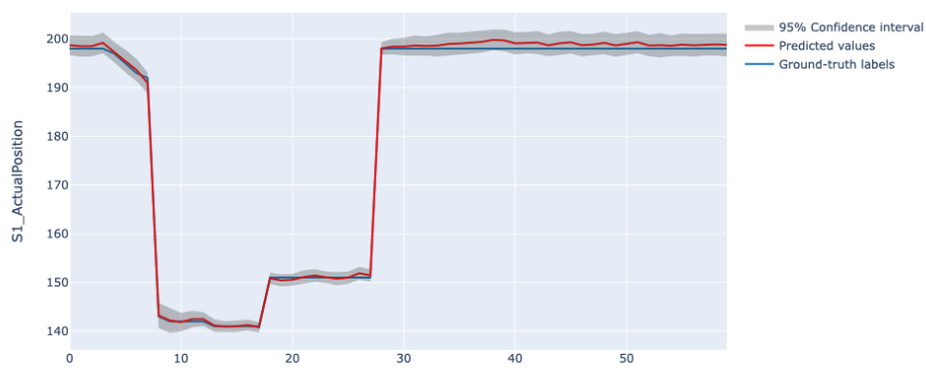

Figure 4.10: A visualization of ninety-five percent prediction interval for a model trained CNC dataset.

The Bayesian neural network has the advantage that we can quantify its predictive uncertainty using a relatively simple method. Our method for quantifying uncertainty can be found in §3.4.2. By making a confidence interval, we visualize the uncertainty. In 4.9, the models prediction and ground truth are shown along with ninety-five percent confidence region on TEP dataset, while 4.10

shows a similar plot on CNC dataset. The interval grows as the predicted value deviates from the ground truth, especially with CNC data.

We computed the coverage probability to demonstrate the effectiveness of Bayesian neural networks in quantifying uncertainty. Generally, a good uncertainty estimation method has equal coverage probability and confidence level, as described in §3.5.5. However, we can expect approximately equal outcomes even if we cannot expect equal coverage and confidence levels. To obtain the coverage probability, we collected one thousand samples from the predictive distribution and then computed the probability using three steps described in §3.5.5. Table 4.5 summarizes the coverage probability for four typical confidence levels for both TEP and CNC data. These results indicate that coverage probability is not significantly different from confidence levels. Most of them display a slight conservatism, which means low confidence in their predictions. Moreover, we observe that coverage probability is more conservative on CNC data than TEP data; This makes sense if we recall our model performance evaluation. CNC had a higher RMSE score, and residual analysis showed non-constant variance. From the industry's perspective, it's better to overestimate the uncertainty than underestimate it. A slight underconfidence in a model that is not perfect is, therefore, intuitively a good quality.

Dataset	Confidence level %	Coverage probability %
TEP data	90.0	91.42
	95.0	95.65
	97.0	97.49
	99.0	99.11
CNC data	90.0	93.66
	95.0	96.51
	97.0	97.31
	99.0	98.47

Table 4.5: Coverage probability for different confidence levels using two different datasets.

4.3 Experiment 2: Fault injection analysis

The experiment in this section answers our third research question. To begin with, we will describe our experimental method and experimental setup, and

then we will present the results of our experiment.

RQ3 How well is uncertainty estimation able to detect single sensor faults?

a) **Faults due to drift**

b) **Faults due to bias**

c) **Faults due to freezing**

d) **Faults due to precision degradation**

A fault injection test is an experiment-based approach used primarily in software and hardware engineering to verify that a system performs as intended (Hsueh et al., 1997). Faults are deliberately introduced during fault injection to test the system's robustness and error handling capabilities. The objective of fault injection is to generate more faults in a system and then collect and evaluate statistical data relating to their effects. We use this methodology to test the Bayesian neural network-based virtual sensor's fault detection capability. To compare and evaluate the collected statistics, we use box plots to visualize and interpret data distribution. Therefore, the following paragraphs will describe fault injection analysis and box plots.

We simulate four artificial faults described in §2.2.1 using their mathematical formulas. These faults are injected into our test dataset at varying lengths. The length varies between 5% and 25% of the length of the test data set. Each experiment generates one thousand experimental data points for each fault type. A fault injection test consists of the following steps.

Step 1: We copy the test dataset and inject a randomly chosen length of artificially generated fault into the data..

Step 2: We store results in a **experimental data list**

Step 3: We repeat the above steps one thousand times.

For each dataset in **experimental data list**.

Step 1: First, we compute a prediction of corresponding ground truth using the model trained on training data

Step 2: Then, we compute predictive uncertainty for each prediction point.

Step 3: Then, we compute the mean of predictive uncertainty.

Step 4: Finally, we compare the distribution of computed uncertainties for different lengths of injected fault.

Box plots (Williamson et al., 1989) are a standardized method for visualizing and interpreting the distribution of data. Box plots provide a high-level summary of data and are well suited to comparing distributions between groups. A box plot provides the following information. Refer to Figure 4.11 for an example of a box plot for a normal distribution.

- **Minimum** $Q1 - 1.5IQR$.
- **First quartile** The first quartile (Q1) is the number that lies in the middle between the smallest (not minimum) and the median. It is also known as (*25th* percentile).
- **Median** The median is the middle value of the data and may also be referred to as the second quantile (Q2).
- **Third quartile** The Third Quartile (Q3) is the number in the middle between the medians and the highest values. It is also known as *75th* percentile.
- **Boxes** Boxes, also known as interquartile ranges (IQR), show the area between Q1 and Q3. It identifies the range of central 50% of the data, with the central line representing the median.
- **Maximum** $Q3 + 1.5IQR$.
- **The whiskers** The whisker is a line that extends from the minimum to the maximum, extending from the box.
- **Outliers** Outliers are lines that are outside the whiskers; they are often represented by dots.

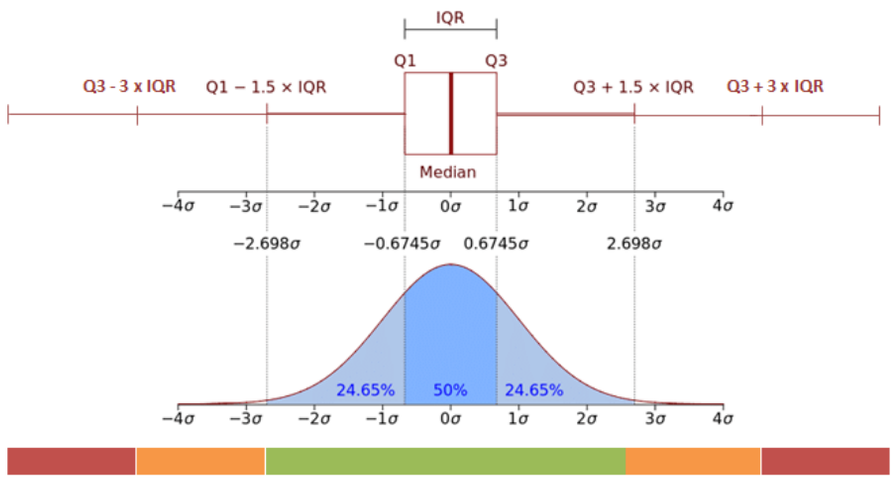

Figure 4.11: Box plot for a normal distribution Adopted from (Olano et al., 2018).

We interpret boxes as follows when comparing the distribution of uncertainties for different fault lengths. For example, please refer to Figure 4.11, where each box represents the predictive uncertainty of a given fault length. For example, the first box could indicate predictive uncertainty when 5 percent of the data is faulty, the second 10 percent of the data, and so on. Among the numerous box-plot statistics, we consider the median value and the spread of the box to be most important.

- **Median** The median of box plots is used to compare the uncertainties for different fault lengths. For example, in Figure 4.11 box 1 and 2 would be interpreted as equal because they represent equal have same Median value. While box 3 will be interpreted as higher than box 2 because it has a higher median value. Similarly, box 4 is higher than all other three boxes.

- **The box and The whiskers** A relatively short box indicates that uncertainty estimates derived from different experiments are highly similar (see box 1 and 4 in Figure 4.12). A relatively tall box indicates that uncertainty estimates derived from different experiments do not have a high degree of similarity (see box 2 in Figure 4.12). If uncertainty varies from iteration to iteration, then its accuracy is questionable.
  The whiskers are interpreted similarly to boxes. Having tall whiskers indicates poor accuracy of uncertainty, while having short whiskers indicates good accuracy of uncertainty.

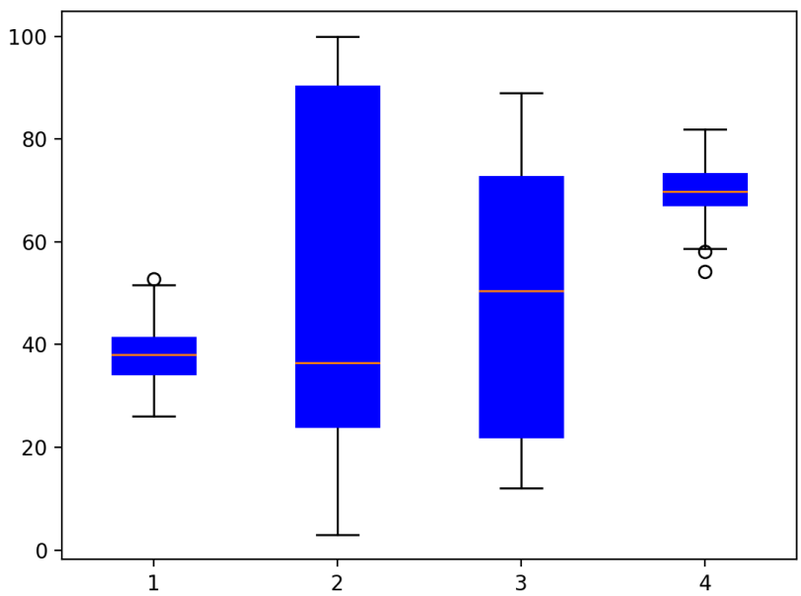

Figure 4.12: Various shapes of box plots for illustration purposes.

4.3.1 Experimental setup

We used the fault injection method described above to conduct four sets of experiments to see how well uncertainty estimated by Bayesian neural network-based virtual sensor can detect changes in data due to sensor drift, bias, freezing, and performance degradation. We used two datasets with four input sensors in one dataset and fourteen input sensors in the other dataset, thus 18 input sensors. We mutated each sensor 1000 times, resulting in 72000 data mutations in total. Each time, we randomly selected the percentage of faults from [5%, 10%, 15%, 20%, 25%]: We also used data with 0% fault as a baseline for comparison.

In the following sections, we will present and discuss the results of these experiments.

4.3.2 Faults due to drift ?

We simulated artificial drift faults according to its mathematical formula in equation (2.1). We defined the Range of the drift is set to 2 times the Range

of the data. Our concept of range refers to a numerical representation of the observation span, that is, the distance between the maximum and minimum values. We varied the percentage of injected fault between 5% and 25% of the total test dataset.

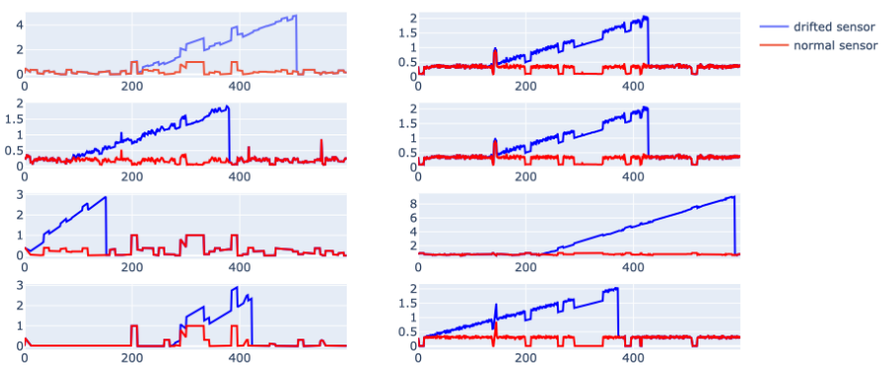

Figure 4.13: An illustration of drift faults injected into different sensors. For this example, the percentage faults have been chosen at random.

In our analysis, we found that the ability of predictive uncertainty to detect faults caused by drift depends both on the input sensor values and the degree of drift. Even the same drift fault can have different effects on different sensors of the input system. Figure 4.14 illustrates how predictive uncertainties can arise when two different sensors are injected with the same type of drift fault. We used green and blue colors to distinguish the TEP and CNC datasets.

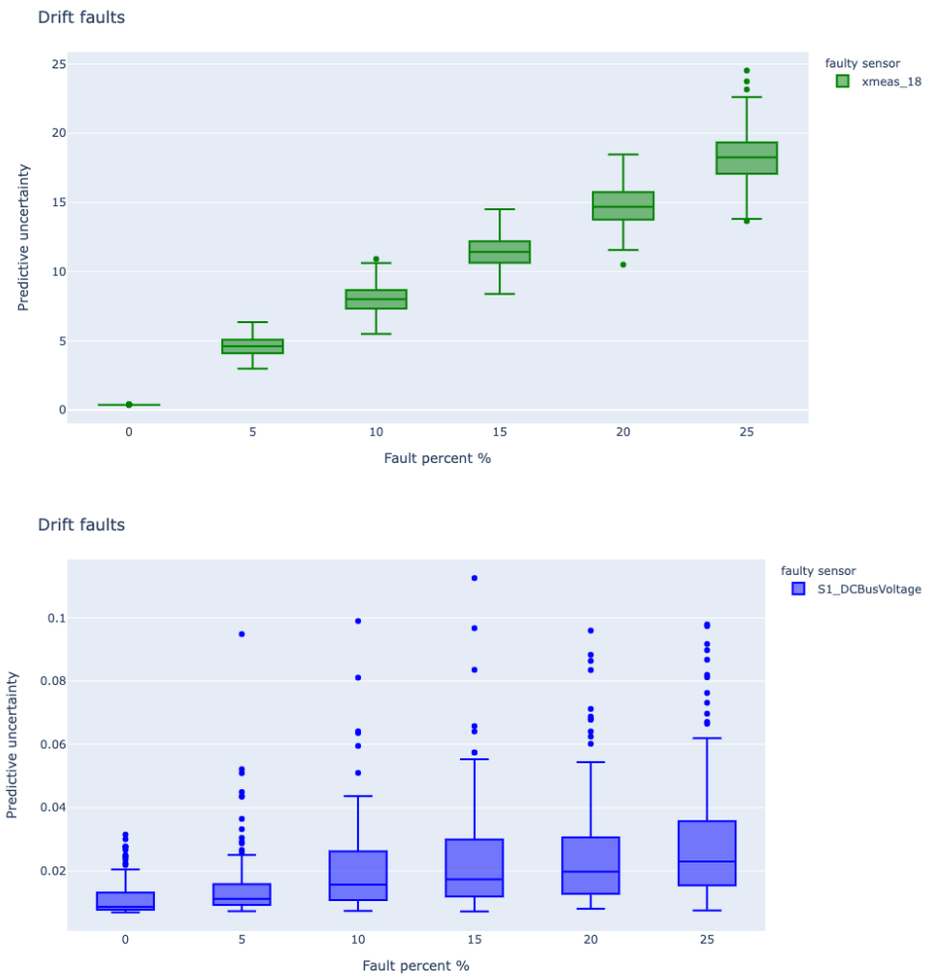

Figure 4.14: Comparison of the predictive uncertainty for different drift levels injected. Different levels of injected drift are represented on the x-axis, while predictive uncertainty is shown on the y-axis. Xmeas_18 is a variable in the TEP dataset, and here it shows the variable or sensor injected with the drift fault.

In the Figure 4.14 illustrated in green, the median predictive uncertainty increases linearly between 0 and 18 as the fault percentages increase from 0 to 25 percent. In contrast, in the blue Figure, the range only lies between 0 and 0.04. In green and blue cases, the boxes tend to increase in size, suggesting that the uncertainty estimation varies more with increasing drift faults. Additionally, we can see that sometimes we get many outliers, as in the blue Figure. For example, uncertainty for 15 percent drift is higher than uncertainty for 20 and 25 percent drift. By presenting higher uncertainty than is appropriate, these

outliers can lead to false alarms.

Our research has discovered more extreme results where the degree of uncertainty doesn't seem to increase with the percentage of faults within a system. Here is an example provided by Figure 4.15. In this illustration, the median predictive uncertainty does not appear to increase as the fault percentage increases. On the other hand, the range outlier (represented by dots) seems to grow linearly as the drift percentage increases.

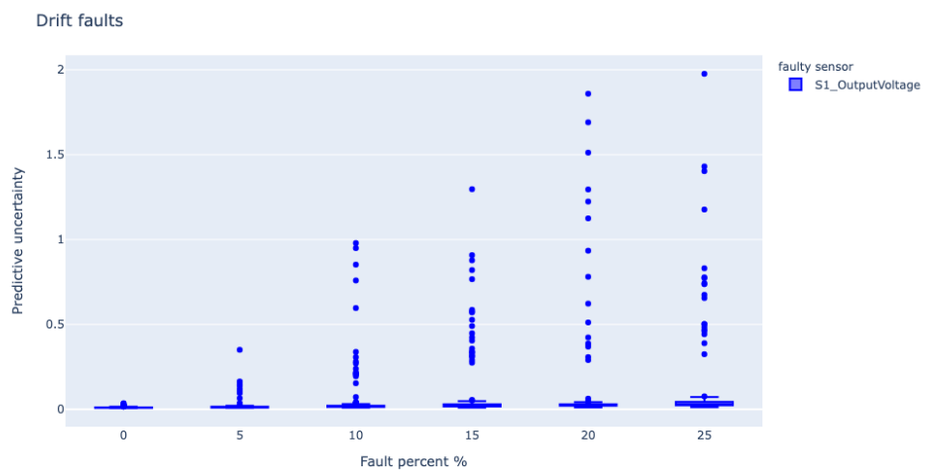

Figure 4.15: Comparison of the predictive uncertainty for different drift levels injected. Different levels of injected drift are represented on the x-axis, while predictive uncertainty is shown on the y-axis. S1_OutputVoltage is a variable in the CNC dataset, and here it shows the variable or sensor injected with the drift fault.

In reality, Fault detection is based more on visualizing predictive uncertainty than analyzing box plots. Thus, we visualized predictive uncertainty as confidence intervals to see how well it could detect drift fault. For example, in Figure 4.16, we have the confidence interval for a prediction when the S1_OutputPower sensor is drifted by 5 and 10 percent. S1_OutputPower is the same sensor as in Figure 4.15, which shows its result using the fault injection analysis. Even though it was difficult to see the increase in predictive uncertainty on box plots, it is evident from the visualizations that the uncertainty increases, and the drift fault can be detected.

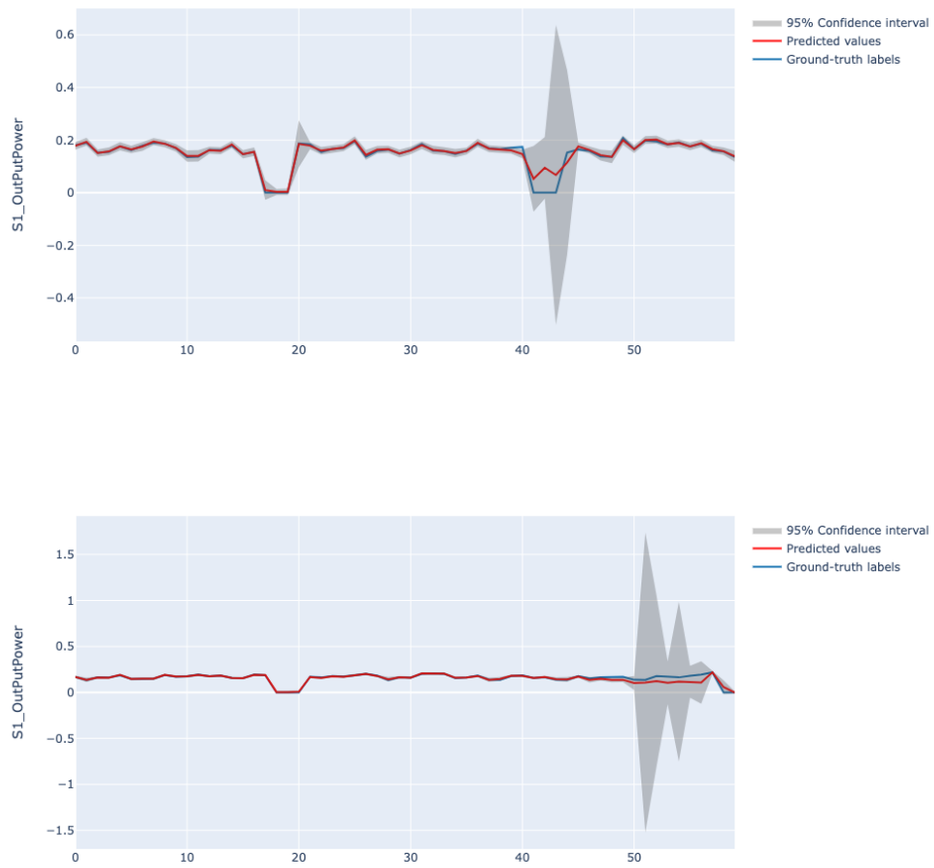

Figure 4.16: Confidence interval for a prediction when S1_OutputVoltage(spindle voltage) is injected with 5(top) and 10(bottom) drift. Ninety-five percent confidence intervals correspond to plus/minus two times the estimated uncertainty.

4.3.3 Faults due to bias?

A bias is when a measurement suffers a constant shift. We simulated artificial bias faults using equation(2.2) in the background section. An illustration of bias faults injected into different sensors is shown in Figure 4.17. All experiments used the same bias constant $C = 100$. Following this, we performed fault injection analysis similar to drift fault analysis.

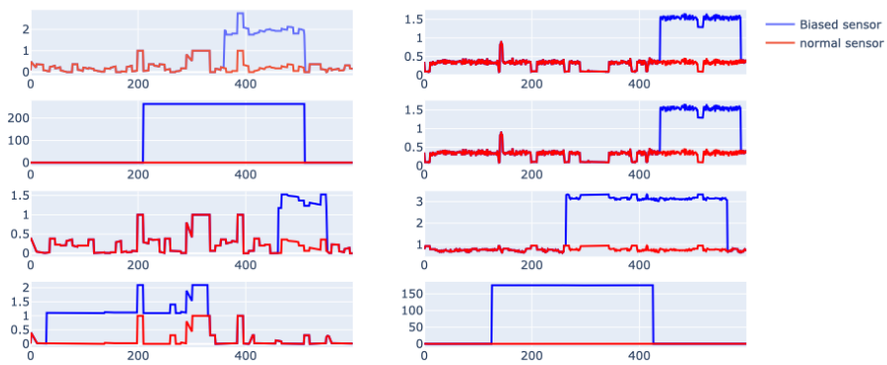

Figure 4.17: An illustration of bias faults injected into different sensors. For this example, the percentage faults have been chosen at random.

Our results showed that estimates of uncertainty were able to detect data bias. As with drift, different sensors exhibit different effects of bias. Figure 4.18 and 4.19 provides a visual representation of these results. The size of boxes and whiskers increase as the fault percentage increases, indicating a decreasing degree of accuracy in uncertainty.

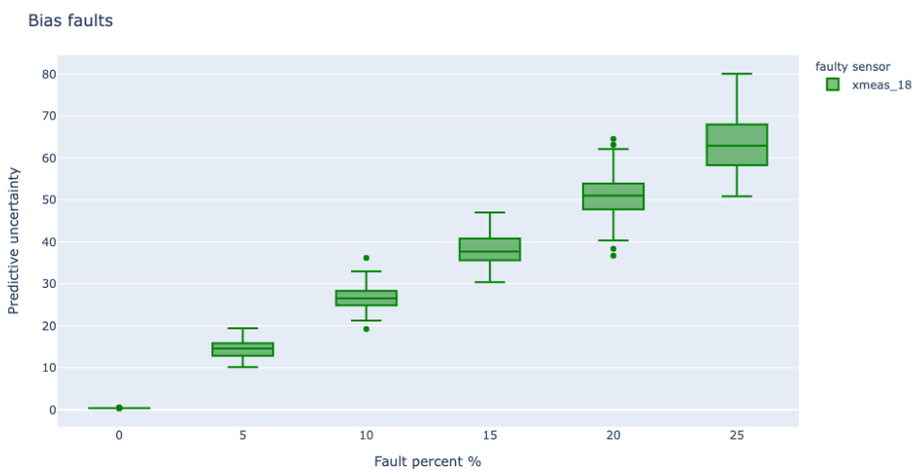

Figure 4.18: Comparison of the predictive uncertainty for different levels of bias injected. Different levels of injected drift are represented on the x-axis, while predictive uncertainty is shown on the y-axis. Xmeas_18 is variable in the TEP C dataset, and here they show the variables or sensors injected with the drift fault.

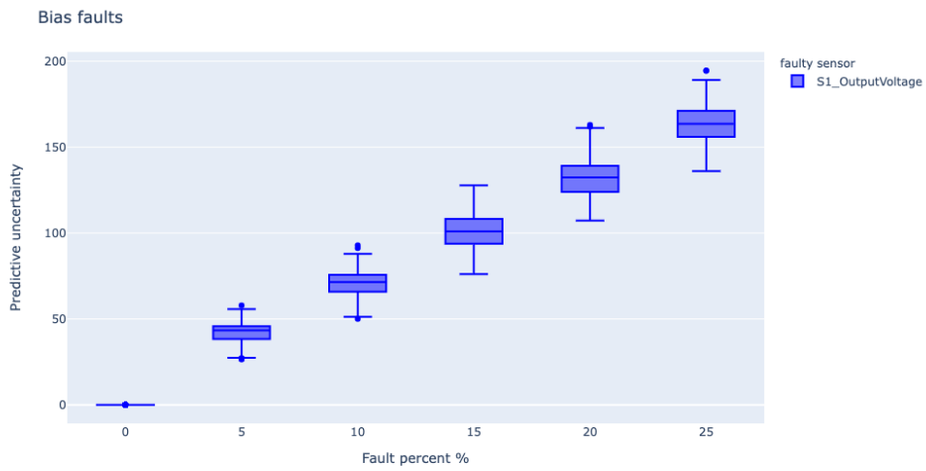

Figure 4.19: Comparison of the predictive uncertainty for different levels of bias injected. Different levels of injected drift are represented on the x-axis, while predictive uncertainty is shown on the y-axis. S1_OutputVoltage is variables in the CNC dataset, and here they show the variables or sensors injected with the drift fault.

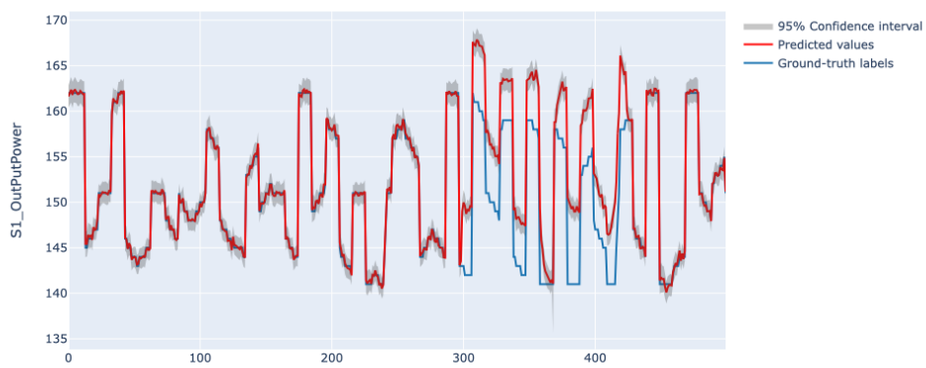

Figure 4.20: The visualization of predictive uncertainty when a bias is present in the input sensor. It shows the prediction with a ninety-five percent confidence intervals correspond to plus/minus two times the estimated uncertainty.

Visualizing the uncertainty interval on a prediction makes it more evident how the uncertainty's accuracy decreases. For example, see Figure 4.20, where we plot a prediction with a ninety-five percent confidence interval. The Figure

illustrates that the model gave an incorrect prediction for the observation between 300 and 400. But we cannot determine this without having a ground truth label since uncertainty does not increase. Figure 4.21 shows the uncertainty associated with the same prediction. Here, we see that the uncertainty increased in that area, but only a single spike.

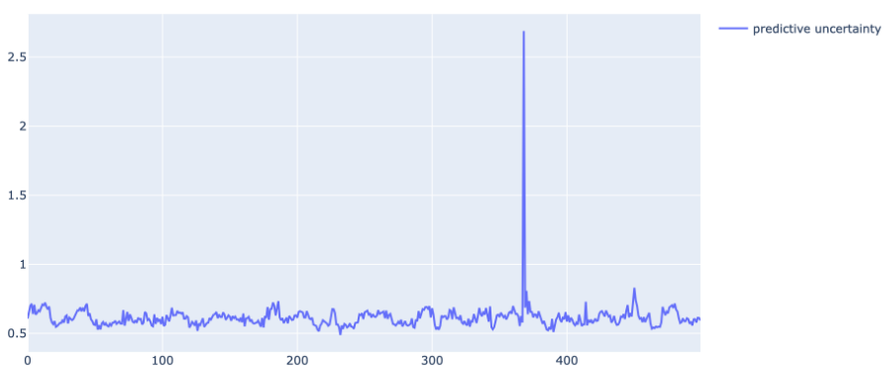

Figure 4.21: The visualization of predictive uncertainty when a bias is present in the input sensor.

4.3.4 Faults due to freezing?

Sensor freezing is the third type of fault we performed fault injection analysis on. An example of a freezing fault is a sensor that produces completely abnormal results. According to the equation(2.3) formula, freezing is generated by using a constant value $C = 999$ to represent an abnormal reading. Please refer to Figure 4.22 for an example of injected freezing faults into different parts of sensors in the CNC dataset.

Our fault injection analysis results show also a linear increase in uncertainty as the proportion of freezing faults increases. We present the results of our analysis in Figure 4.23 and 4.24

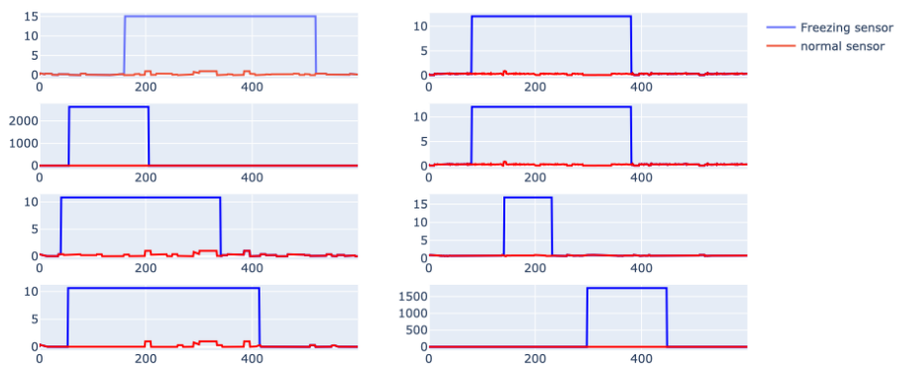

Figure 4.22: Various input sensors were injected with freezing faults in the experimental test dataset. The fault length in this figure is randomly generated since it is an illustration.

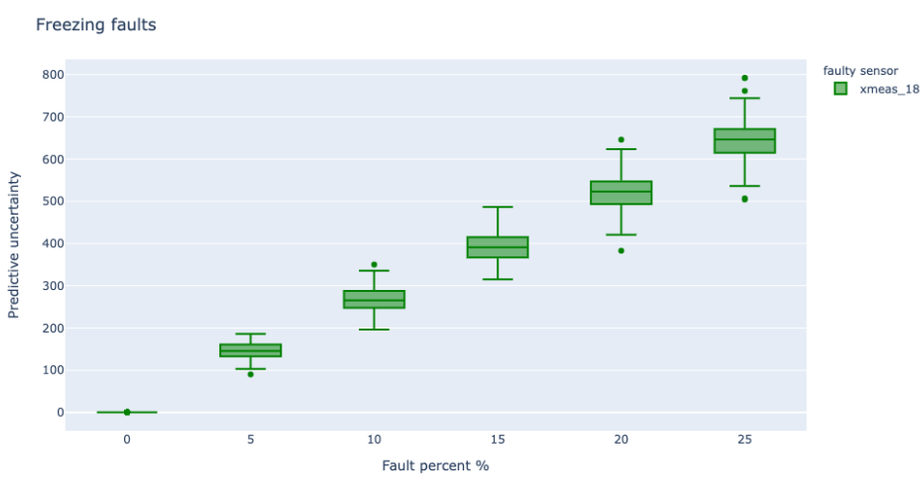

Figure 4.23: This figure compares the predictive uncertainty for different input sensors and different levels of freezing faults injected on TEP data.

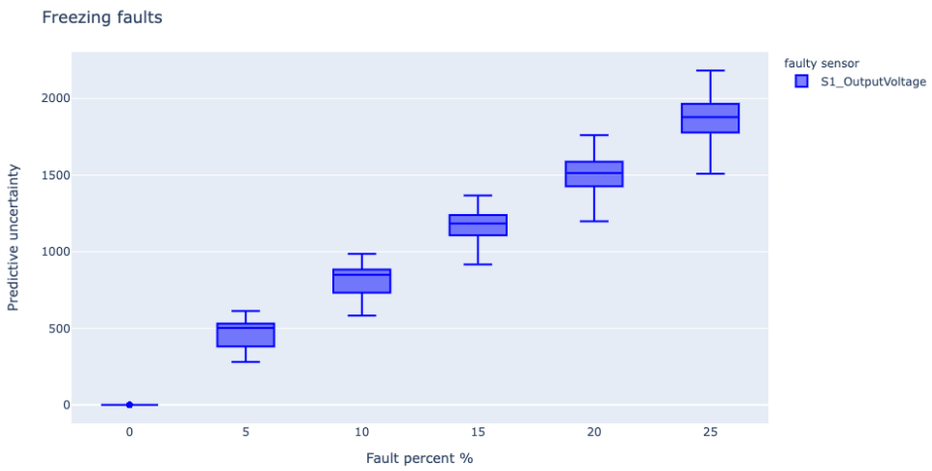

Figure 4.24: This figure compares the predictive uncertainty for different input sensors and different levels of freezing faults injected on CNC data.

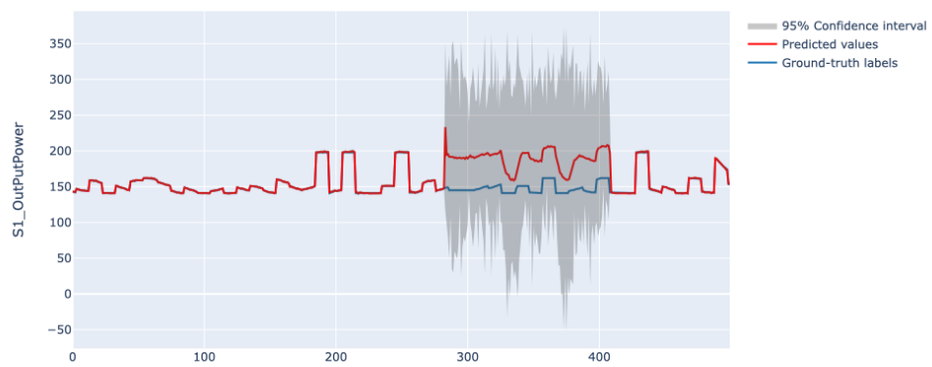

Figure 4.25: An example of how predictive uncertainty can be used to detect changes freezing faults in input dataset. It shows the prediction with a ninety-five percent confidence interval. Ninety-five percent confidence intervals correspond to plus/minus two times the estimated uncertainty.

We can visually change in prediction, as in Figure 4.25 and 4.26. However, As the uncertainty estimates became more inaccurate, the confidence interval visualization sometimes provided unexpected results when the input sensors were frozen. Figure 4.26 displays predictive uncertainty as a ninety-five percent confidence interval; In this Figure, we see an intuitive visualization. However, if

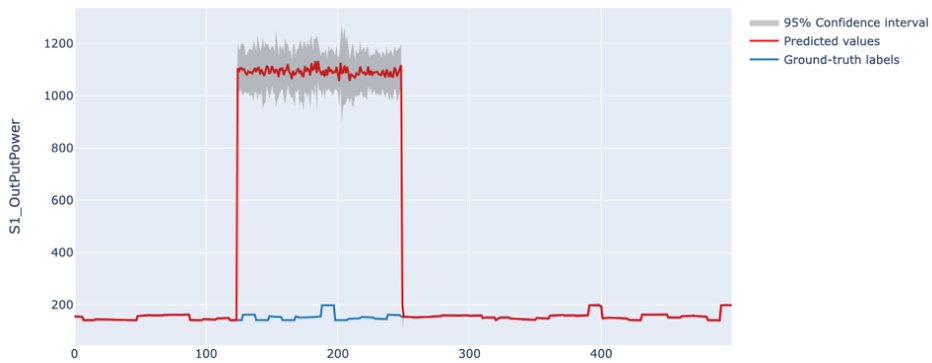

Figure 4.26: An example of how predictive uncertainty can be used to detect changes freezing faults in input dataset. It shows the prediction with a ninety-five percent confidence interval. Ninety-five percent confidence intervals correspond to plus/minus two times the estimated uncertainty.

we see another similar visualization in 4.26, we see that the ninety-five percent interval is not intuitive.

4.3.5 Faults due to performance degradation?

For the final analysis, we simulated performance degradation using (2.4 and analyzed it using fault injection analysis. Figure 4.27 illustrates an example of a degraded sensor that we injected into different sensors of our experimental data.

Experimental results reported a linear increase in predictive uncertainty with increasing percentages of degradation fault. This is similar to the other three fault types. In Figure 4.28 and 4.29 you can see the results of analyzing performance degradation faults.

However, similar to all three other results, the accuracy of uncertainty decreases as the fault due to performance degradation increase. In fact, as we can see from Figures 4.28 and 4.29, the size of the boxes and whiskers increase more(compared to the other results) as the percentage of faults increases.

Performance degradation in the model input leads to a noisy prediction and noisy prediction uncertainty. Figures 4.30 illustrate how uncertainty can be

visualized in terms of confidence interval.

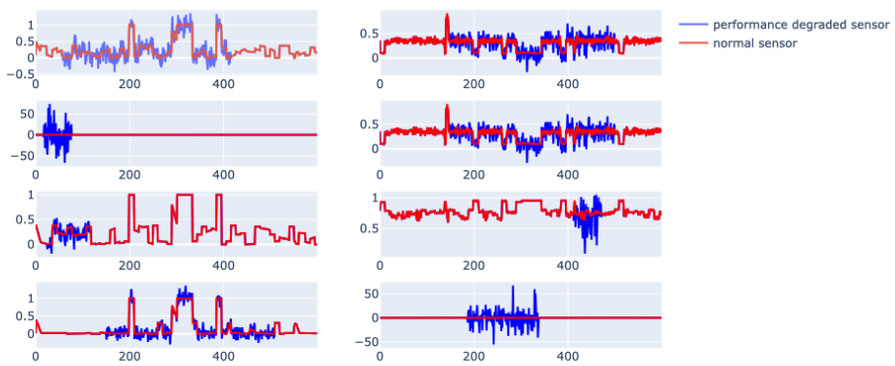

Figure 4.27: An illustration of a noisy/performance degraded sensor injected into our experimental test dataset.

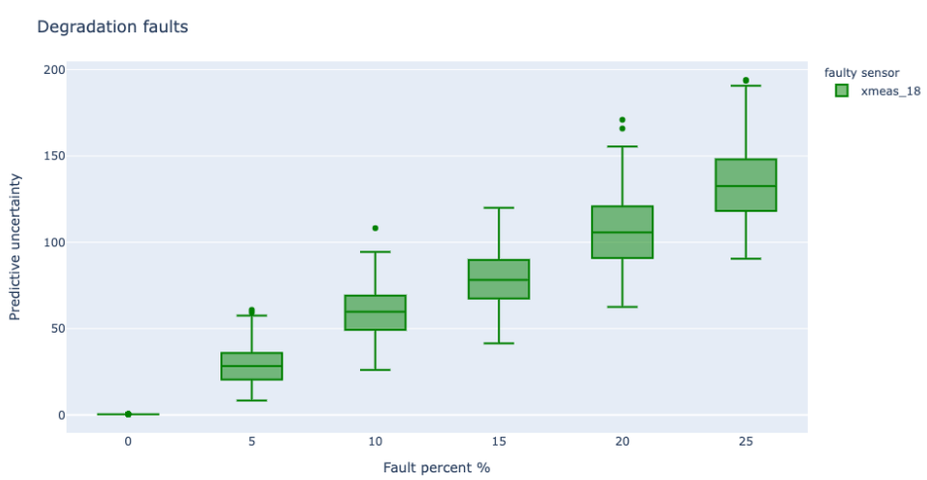

Figure 4.28: Comparing the predictive uncertainty for different input sensors and different levels of performance degradation faults injected on TEP data.

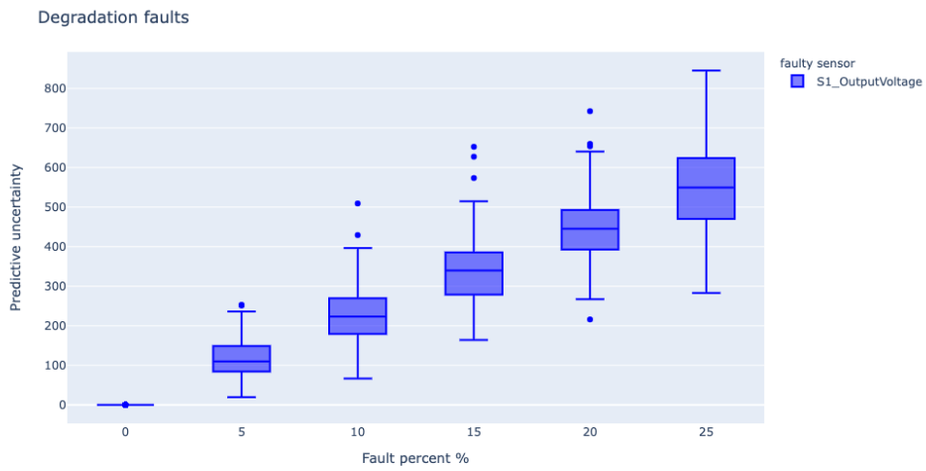

Figure 4.29: Comparing the predictive uncertainty for different input sensors and different levels of performance degradation faults injected on CNC data.

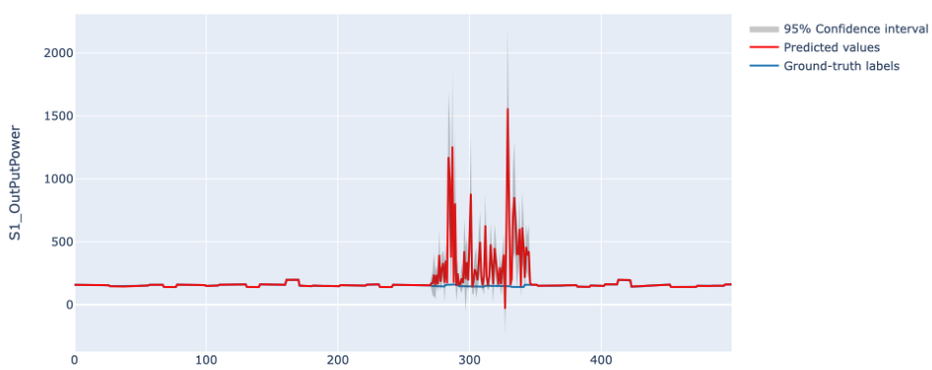

Figure 4.30: An example of how predictive uncertainty can be used to detect changes degradation faults in input dataset. It shows the prediction with a ninety-five percent confidence interval. Ninety-five percent confidence intervals correspond to plus/minus two times the estimated uncertainty.

4.4 Result Summary

In the first section of this chapter, we addressed the research questions **RQ1** and **RQ2**: **How can we build and validate uncertainty-aware virtual sensors**

using a Bayesian neural network, and How can we evaluate the accuracy of the uncertainty estimated by the Bayesian neural network?

In the first step , we build uncertainty-aware virtual sensors using two industrial manufacturing datasets based on Bayesian convolutional neural networks. We chose input sensors based on the Pearson correlation coefficients provided by Pandas' profiling report. We then trained the model by tuning its hyperparameters according to the trial-and-error method.

Then, we evaluate the predictive performance of the virtual sensor using the R2, RMSE, and MAPE scores. We compare our results to non-Bayesian CNNs and LSTM based virtual sensors. Our Bayesian-based virtual sensor had the best R2 and MAPE scores, with R2 = 0.99 on CNC data and R2 = 0.98 on TEP data, and MAPE = 0.0017 on CNC and MAPE = 0.005 on TEP data. Our model had the worst RMSE value of 0.40, CNN and LSTM models had RMSE values of 0.0027 and 0.0016, respectively, on CNC data. that

To investigate the performance of our virtual sensor further, we used residual analysis. We found that residual variance for models trained on CNC data is not constant, indicating that a given segment has a different predictive ability. We concluded that the high RMSE score in our model may be due to this non-constant variance.

We assessed the accuracy of the estimated uncertainty by comparing the coverage probability and confidence levels. The coverage probability is almost identical to the confidence level, with a small sign of conservatively, indicating a reasonable uncertainty estimate.

The second chapter discussed our last research question **RQ3**: **How well is uncertainty estimation able to detect single sensor faults due to drift, due to bias,due to freezing, and due to performance degradation?** According to our findings, the uncertainty increased in each of the four cases as the fault percentage increased, indicating the uncertainty can detect changes in data distribution. However the accuracy of the uncertainty in all four experiments declined as fault length increased. In addition, we found that the ability of predictive uncertainty to detect faults in input data depends both on the input sensor values and the degree of faults. Even the same fault can have different effects on different sensors of the input system.

5

Conclusion and Future work

This chapter will discuss future work and conclude with a few closing remarks.

5.1 Conclusion

In this thesis, we explored the potential of the Bayesian convolutional neural network as an uncertainty-aware virtual sensor for Industry 4.0.

We have created an uncertainty-aware virtual sensor by utilizing the most recent Bayesian method for estimating distribution over the weight parameter. In order to evaluate models' predictive performance, we used four different metrics, including root mean square error (RMSE), mean absolute percentage error(MAPE), and R-squared (R2) and Residual analysis. These are all standard metrics in statistics and machine learning. In addition, we calculated coverage probability to assess the method's capability of estimating uncertainty. Finally, we also tested how estimated uncertainty can detect changes in the input dataset by performing fault injection analysis.

Based on our results, Bayesian convolutional neural network-based uncertainty-aware virtual sensors perform as well as any deep learning model when it comes

to prediction accuracy. We found that our BCNN virtual sensor had the highest R-squared scores, with $R2 = 0.99$ on CNC and $R2 = 0.98$ on TEP data. While the RMSE for our model was 0.40, CNN and LSTM models had RMSE values of 0.0027 and 0.0016, respectively, on CNC data. The result of the coverage probability score indicates a reasonably good uncertainty estimate.

Our result of fault injection analysis showed that the uncertainty increased in each of the four cases as the fault percentage increased, indicating the uncertainty can detect changes in data distribution. However the accuracy of the uncertainty in all four experiments declined as fault length increased. In addition, we found that the ability of predictive uncertainty to detect faults in input data depends both on the input sensor values and the degree of faults. Even the same fault can have different effects on different sensors of the input system.

We can conclude that the Bayesian neural network can be used as a uncertainty-aware virtual sensor, However, more research is needed to find out why the accuracy of the uncertainty estimate decreased as the length of the fault increased.

5.2 Future Work

Further experimentation with fault injection We analyzed only four faults in the fault injection analysis: drift, bias, freezing, and performance degradation. However, there may be other sensor faults such as calibration errors, scaling errors, etc. In addition, we only considered a single sensor fault, whereas it is likely that multiple sensors would fail simultaneously, either with the same fault or with different faults. Moreover, we have not been able to determine why the accuracy of the uncertainty decreased with fault length. Therefore, further experiments with fault injection could have benefited this thesis.

Explainable Artificial Intelligence Visualization uncertainty can alert us when a model mistake has occurred, but it does not tell us the cause of the mistake, what kind of change in the dataset has occurred, or what input has been changed. Thus, it is also essential to identify the type of fault in the dataset and which sensor/feature variables are most helpful in identifying faults. Is explainable artificial intelligence useful in identifying faults and input features that lead to faults accurately and timely? Several model-agnostic

explanations of AI are available here (Molnar, 2020): The two most popular are LIME (Ribeiro et al., 2016) and SHAP (Lundberg and Lee, 2017)]. Most of them, however, work with frequentist models. There is a Bayesian extension of LIME called Baylime (Zhao et al., 2021) that currently does not support 3D input. Therefore, we can not conduct any experiments with explainable AI and leave it to future research.

Prior in Bayesian Neural Network The main focus of the Bayesian neural network (Fortuin, 2021)has been estimating the posterior distribution of parameters and identifying properties of predictive uncertainty; however, choosing the prior is one of the most critical and crucial parts. A standard Gaussian prior over the parameters is often considered sufficient (Wilson and Izmailov, 2020). This is supported by the central limit theorem (Dudley, 2014), in which samples taken from data will asymptotically approach a normal distribution regardless of their underlying distribution. Recent evidence, however, has questioned the validity of Gaussian priors in Bayesian neural network (Wenzel et al., 2020). Since the objective of our thesis was to validate an existing Bayesian neural network in the context of virtual sensors, for this project, we also used a standard Gaussian prior. Further research is necessary to replace the standard Gaussian prior.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Albawi, S., Mohammed, T. A., and Al-Zawi, S. (2017). Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. Ieee.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bai, S., Kolter, J. Z., and Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*.

Barlow, H. B. (1989). Unsupervised learning. *Neural computation*, 1(3):295–311.

Benesty, J., Chen, J., Huang, Y., and Cohen, I. (2009). Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer.

Betancourt, M. (2017). A conceptual introduction to hamiltonian monte carlo. *arXiv preprint arXiv:1701.02434*.

Bhanja, S. and Das, A. (2018). Impact of data normalization on deep neural network for time series forecasting. *arXiv preprint arXiv:1812.05519*.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622. PMLR.

Box, G. E. and Tiao, G. C. (2011). *Bayesian inference in statistical analysis*, volume 40. John Wiley & Sons.

Brugman, S. (2020). pandas-profiling : Exploratory data analysis reports in python.

Caruana, R. and Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168.

Chai, T. and Draxler, R. R. (2014). Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature. *Geoscientific model development*, 7(3):1247–1250.

Chen, C., Yuan, J., Lu, Y., Liu, Y., Su, H., Yuan, S., and Liu, S. (2020). Oodanalyzer: Interactive analysis of out-of-distribution samples. *IEEE transactions on visualization and computer graphics*, 27(7):3335–3349.

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., and Zhang, Z. (2015). Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*.

Chen, X. (2019). Tennessee eastman simulation dataset.

Chib, S. and Greenberg, E. (1995). Understanding the metropolis-hastings algorithm. *The american statistician*, 49(4):327–335.

Childs, P. R., Greenwood, J., and Long, C. (2000). Review of temperature measurement. *Review of scientific instruments*, 71(8):2959–2978.

Chollet, F. et al. (2015). Keras.

Cook, R. D. and Weisberg, S. (1982). *Residuals and influence in regression*. New York: Chapman and Hall.

Curreri, F., Fiumara, G., and Xibilia, M. G. (2020). Input selection methods for soft sensor design: a survey. *Future Internet*, 12(6):97.

De Myttenaere, A., Golden, B., Le Grand, B., and Rossi, F. (2016). Mean absolute percentage error for regression models. *Neurocomputing*, 192:38–48.

Der Kiureghian, A. and Ditlevsen, O. (2009). Aleatory or epistemic? does it matter? *Structural safety*, 31(2):105–112.

Dillon, J. V., Langmore, I., Tran, D., Brevdo, E., Vasudevan, S., Moore, D., Patton, B., Alemi, A., Hoffman, M., and Saurous, R. A. (2017). Tensorflow distributions. *arXiv preprint arXiv:1711.10604*.

Downs, J. J. and Vogel, E. F. (1993). A plant-wide industrial process control problem. *Computers & chemical engineering*, 17(3):245–255.

Droguett, E. L. and Mosleh, A. (2008). Bayesian methodology for model uncertainty using model performance data. *Risk Analysis: An International Journal*, 28(5):1457–1476.

Dudley, R. M. (2014). *Uniform central limit theorems*, volume 142. Cambridge university press.

Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

Ehmann, K., Kapoor, S., DeVor, R., and Lazoglu, I. (1997). Machining process modeling: a review. *Journal of Manufacturing Science and Engineering, Transactions of the ASME*, 119(4B):655–663.

Eiliat, H. and Urbanic, J. (2018). Visualizing, analyzing, and managing voids in the material extrusion process. *The International Journal of Advanced Manufacturing Technology*, 96(9):4095–4109.

Fan, Y., Tao, B., Zheng, Y., and Jang, S.-S. (2019). A data-driven soft sensor based on multilayer perceptron neural network with a double lasso approach. *IEEE Transactions on Instrumentation and Measurement*, 69(7):3972–3979.

Feng, L., Zhao, C., and Sun, Y. (2020). Dual attention-based encoder-decoder: A customized sequence-to-sequence learning for soft sensor development. *IEEE Transactions on Neural Networks and Learning Systems*.

Figueiredo Filho, D. B., Júnior, J. A. S., and Rocha, E. C. (2011). What is r2 all about? *Leviathan (São Paulo)*, (3):60–68.

Fort, S., Hu, H., and Lakshminarayanan, B. (2019). Deep ensembles: A loss landscape perspective. *arXiv preprint arXiv:1912.02757*.

Fortuin, V. (2021). Priors in bayesian deep learning: A review. *arXiv preprint arXiv:2105.06868*.

Fortuna, L., Graziani, S., Rizzo, A., and Xibilia, M. G. (2007). *Soft sensors for monitoring and control of industrial processes*, volume 22. Springer.

Fortuna, L., Graziani, S., and Xibilia, M. G. (2005). Soft sensors for product quality monitoring in debutanizer distillation columns. *Control Engineering Practice*, 13(4):499–508.

Ge, Z., Huang, B., and Song, Z. (2014). Mixture semisupervised principal component regression model and soft sensor application. *AIChE Journal*, 60(2):533–545.

Gibson, I., Rosen, D., Stucker, B., and Khorasani, M. (2014). *Additive manufacturing technologies*, volume 17. Springer.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Graves, A. (2011). Practical variational inference for neural networks. *Advances in neural information processing systems*, 24.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2016). Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232.

Guo, F., Bai, W., and Huang, B. (2020). Output-relevant variational autoencoder for just-in-time soft sensor modeling with missing data. *Journal of Process Control*, 92:90–97.

Gutama, I. M. (2021). Virtual sensor to repair erroneous sensor data in manufacturing. project thesis.

Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier.

Herceg, S., Andrijić, Ž. U., and Bolf, N. (2019). Development of soft sensors for isomerization process based on support vector machine regression and dynamic polynomial models. *Chemical Engineering Research and Design*, 149:95–103.

Hernández-Lobato, J. M. and Adams, R. (2015). Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International conference on machine learning*, pages 1861–1869. PMLR.

Hinton, G. E. and Van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the*

sixth annual conference on Computational learning theory, pages 5–13.

Hira, Z. M. and Gillies, D. F. (2015). A review of feature selection and feature extraction methods applied on microarray data. *Advances in bioinformatics*, 2015.

Hoffman, M. D. and Gelman, A. (2014). The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623.

Hsueh, M.-C., Tsai, T. K., and Iyer, R. K. (1997). Fault injection techniques and tools. *Computer*, 30(4):75–82.

Hüllermeier, E. and Waegeman, W. (2021). Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine Learning*, 110(3):457–506.

Isermann, R. (2005). *Fault-diagnosis systems: an introduction from fault detection to fault tolerance*. Springer Science & Business Media.

Jan, S. U., Lee, Y. D., and Koo, I. S. (2021). A distributed sensor-fault detection and diagnosis framework using machine learning. *Information Sciences*, 547:777–796.

Jiang, X. and Ge, Z. (2021). Augmented multidimensional convolutional neural network for industrial soft sensing. *IEEE Transactions on Instrumentation and Measurement*, 70:1–10.

Jordan, M. I. and Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260.

Jospin, L. V., Buntine, W., Boussaid, F., Laga, H., and Bennamoun, M. (2020). Hands-on bayesian neural networks–a tutorial for deep learning users. *arXiv preprint arXiv:2007.06823*.

Kadlec, P., Gabrys, B., and Strandt, S. (2009). Data-driven soft sensors in the process industry. *Computers & chemical engineering*, 33(4):795–814.

Khosravi, A., Nahavandi, S., Creighton, D., and Atiya, A. F. (2010). Lower upper bound estimation method for construction of neural network-based prediction intervals. *IEEE transactions on neural networks*, 22(3):337–346.

Kingma, D. P., Salimans, T., and Welling, M. (2015). Variational dropout and the local reparameterization trick. *Advances in neural information processing*

systems, 28:2575–2583.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kiranyaz, S., Avci, O., Abdeljaber, O., Ince, T., Gabbouj, M., and Inman, D. J. (2021). 1d convolutional neural networks and applications: A survey. *Mechanical systems and signal processing*, 151:107398.

Kok, Y., Tan, X. P., Wang, P., Nai, M., Loh, N. H., Liu, E., and Tor, S. B. (2018). Anisotropy and heterogeneity of microstructure and mechanical properties in metal additive manufacturing: A critical review. *Materials & Design*, 139:565–586.

Kristiadi, A., Hein, M., and Hennig, P. (2020). Being bayesian, even just a bit, fixes overconfidence in relu networks. In *International Conference on Machine Learning*, pages 5436–5446. PMLR.

Kuprieiev, R., Petrov, D., Pachhai, S., Redzynski, P., da Costa-Luis, C., Rowlands, P., Schepanovski, A., Shcheklein, I., Taskaya, B., Orpinel, J., et al. (2021). Dvc: Data version control—git for data & models.

Lakara, K., Bhandari, A., Seth, P., and Verma, U. (2021). Evaluating predictive uncertainty and robustness to distributional shift using real world data. *arXiv preprint arXiv:2111.04665*.

Lasi, H., Fettke, P., Kemper, H.-G., Feld, T., and Hoffmann, M. (2014). Industry 4.0. *Business & information systems engineering*, 6(4):239–242.

Lea, C., Flynn, M. D., Vidal, R., Reiter, A., and Hager, G. D. (2017). Temporal convolutional networks for action segmentation and detection. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 156–165.

Li, D., Li, Z., and Sun, K. (2020). Development of a novel soft sensor with long short-term memory network and normalized mutual information feature selection. *Mathematical Problems in Engineering*, 2020.

Li, L., Brookfield, D., and Steen, W. (1996). Plasma charge sensor for in-process, non-contact monitoring of the laser welding process. *Measurement Science and Technology*, 7(4):615.

Li, X., Djordjevich, A., and Venuvinod, P. K. (2000). Current-sensor-based feed cutting force intelligent estimation and tool wear condition monitoring. *IEEE*

Transactions on Industrial Electronics, 47(3):697–702.

Lin, C.-Y., Wang, A. X., Lee, B. S., Zhang, X., and Chen, R. T. (2011). High dynamic range electric field sensor for electromagnetic pulse detection. *Optics express*, 19(18):17372–17377.

Liu, H., Huang, M., Janghorban, I., Ghorbannezhad, P., and Yoo, C. (2011). Faulty sensor detection, identification and reconstruction of indoor air quality measurements in a subway station. In *2011 11th International Conference on Control, Automation and Systems*, pages 323–328. IEEE.

Lu, Y. and Wang, Y. (2018). Monitoring temperature in additive manufacturing with physics-based compressive sensing. *Journal of manufacturing systems*, 48:60–70.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc.

May, R., Dandy, G., and Maier, H. (2011). Review of input variable selection methods for artificial neural networks. *Artificial neural networks-methodological advances and biomedical applications*, 10:16004.

Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048. Makuhari.

Miller, A. C., Foti, N. J., D'Amour, A., and Adams, R. P. (2017). Reducing reparameterization gradient variance. *arXiv preprint arXiv:1705.07880*.

Molnar, C. (2020). *Interpretable machine learning*. Lulu. com.

Moriwaki, T. (2008). Multi-functional machine tool. *CIRP annals*, 57(2):736–749.

Mullachery, V., Khera, A., and Husain, A. (2018). Bayesian neural networks. *arXiv preprint arXiv:1801.07710*.

Olano, X., de Jalón, A. G., Pérez, D., Barberena, J. G., López, J., and Gastón, M. (2018). Outcomes and features of the inspection of receiver tubes (itr) system for improved o&m in parabolic trough plants. In *AIP Conference Proceedings*, volume 2033, page 030011. AIP Publishing LLC.

Olive, D. J. (2007). Prediction intervals for regression models. *Computational statistics & data analysis*, 51(6):3115–3122.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., Dillon, J. V., Lakshminarayanan, B., and Snoek, J. (2019). Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *arXiv preprint arXiv:1906.02530*.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. PMLR.

Pasquale, M. (2003). Mechanical sensors and actuators. *Sensors and Actuators A: Physical*, 106(1-3):142–148.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Pérez, A. T. and Hadfield, M. (2011). Low-cost oil quality sensor based on changes in complex permittivity. *Sensors*, 11(11):10675–10690.

Pezoa, F., Reutter, J. L., Suarez, F., Ugarte, M., and Vrgoč, D. (2016). Foundations of json schema. In *Proceedings of the 25th International Conference on World Wide Web*, pages 263–273.

Reinartz, C., Kulahci, M., and Ravn, O. (2021). An extended tennessee eastman simulation dataset for fault-detection and decision support systems. *Computers & Chemical Engineering*, 149:107281.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). " why should i trust you?"

explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144.

Rieth, C. A., Amsel, B. D., Tran, R., and Cook, M. B. (2017). Additional Tennessee Eastman Process Simulation Data for Anomaly Detection Evaluation.

Salkind, N. J. (2006). *Encyclopedia of measurement and statistics*. SAGE publications.

Salovey, P. and Mayer, J. D. (1990). Emotional intelligence. *Imagination, cognition and personality*, 9(3):185–211.

Sehee_Lee (2021). awesome-industrial-machine-datasets.

Shannon, R. V., Zeng, F.-G., Kamath, V., Wygonski, J., and Ekelid, M. (1995). Speech recognition with primarily temporal cues. *Science*, 270(5234):303–304.

Sharma, A. B., Golubchik, L., and Govindan, R. (2010). Sensor faults: Detection methods and prevalence in real-world datasets. *ACM Transactions on Sensor Networks (TOSN)*, 6(3):1–39.

Souza, F. A., Araújo, R., and Mendes, J. (2016). Review of soft sensor methods for regression applications. *Chemometrics and Intelligent Laboratory Systems*, 152:69–79.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Sun, M., Song, Z., Jiang, X., Pan, J., and Pang, Y. (2017). Learning pooling for convolutional neural network. *Neurocomputing*, 224:96–104.

Sun, Q. and Ge, Z. (2021). A survey on deep learning for data-driven soft sensors. *IEEE Transactions on Industrial Informatics*.

Sun, S. (2021). Cnc mill tool wear.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Svozil, D., Kvasnicka, V., and Pospichal, J. (1997). Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory sys-*

tems, 39(1):43–62.

Tagasovska, N. and Lopez-Paz, D. (2019). Single-model uncertainties for deep learning. *Advances in Neural Information Processing Systems*, 32:6417–6428.

Tayman, J. and Swanson, D. A. (1999). On the validity of mape as a measure of population forecast accuracy. *Population Research and Policy Review*, 18(4):299–322.

Thyer, G. (2014). *Computer numerical control of machine tools*. Elsevier.

Tönshoff, H. K. and Inasaki, I. (2001). *Sensors in manufacturing*, volume 236. Wiley Online Library.

Turner, C. R., Fuggetta, A., Lavazza, L., and Wolf, A. L. (1999). A conceptual basis for feature engineering. *Journal of Systems and Software*, 49(1):3–15.

Van Erven, T. and Harremos, P. (2014). Rényi divergence and kullback-leibler divergence. *IEEE Transactions on Information Theory*, 60(7):3797–3820.

Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

Vergara, J. R. and Estévez, P. A. (2014). A review of feature selection methods based on mutual information. *Neural computing and applications*, 24(1):175–186.

Verran, J. A. and Ferketich, S. L. (1984). Residual analysis for statistical assumptions of regression equations. *Western Journal of Nursing Research*, 6(1):27–40.

Wang, K.-S. (2013). Towards zero-defect manufacturing (zdm)—a data mining approach. *Advances in Manufacturing*, 1(1):62–74.

Wen, Y., Vicol, P., Ba, J., Tran, D., and Grosse, R. (2018). Flipout: Efficient pseudo-independent weight perturbations on mini-batches. *arXiv preprint arXiv:1803.04386*.

Wenzel, F., Roth, K., Veeling, B. S., Świątkowski, J., Tran, L., Mandt, S., Snoek, J., Salimans, T., Jenatton, R., and Nowozin, S. (2020). How good is the bayes posterior in deep neural networks really? *arXiv preprint arXiv:2002.02405*.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

Wes McKinney (2010). Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61.

Williamson, D. F., Parker, R. A., and Kendrick, J. S. (1989). The box plot: a simple visual method to interpret data. *Annals of internal medicine*, 110(11):916–921.

Willmott, C. J. and Matsuura, K. (2005). Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1):79–82.

Wilson, A. G. and Izmailov, P. (2020). Bayesian deep learning and a probabilistic perspective of generalization. *arXiv preprint arXiv:2002.08791*.

Wong, K. V. and Hernandez, A. (2012). A review of additive manufacturing. *International scholarly research notices*, 2012.

Xie, R., Jan, N. M., Hao, K., Chen, L., and Huang, B. (2019). Supervised variational autoencoders for soft sensor modeling with missing data. *IEEE Transactions on Industrial Informatics*, 16(4):2820–2828.

Yaghobi, H. (2016). Out-of-step protection of generator using analysis of angular velocity and acceleration data measured from magnetic flux. *Electric Power Systems Research*, 132:9–21.

Youssef, H. A. and El-Hofy, H. (2008). *Machining technology: machine tools and operations*. CRC Press.

Zhao, X., Huang, W., Huang, X., Robu, V., and Flynn, D. (2021). Baylime: Bayesian local interpretable model-agnostic explanations. In *Uncertainty in Artificial Intelligence*, pages 887–896. PMLR.

Zheng, J. and Song, Z. (2018). Semisupervised learning for probabilistic partial least squares regression model and soft sensor application. *Journal of process control*, 64:123–131.

Zhou, Z.-H., Wu, J., and Tang, W. (2002). Ensembling neural networks: many could be better than all. *Artificial intelligence*, 137(1-2):239–263.