A Comparative Study between Bayesian and Frequentist Neural Networks for Remaining Useful Life Estimation in Condition-Based Maintenance[☆]

Luca Della Libera[a,*]

[a]Department of Informatics, Technical University of Munich, Boltzmannstr. 3, 85748 Garching bei München, Germany

Abstract

In the last decade, deep learning (DL) has outperformed model-based and statistical approaches in predicting the remaining useful life (RUL) of machinery in the context of condition-based maintenance. One of the major drawbacks of DL is that it heavily depends on a large amount of labeled data, which are typically expensive and time-consuming to obtain, especially in industrial applications. Scarce training data lead to uncertain estimates of the model's parameters, which in turn result in poor prognostic performance. Quantifying this parameter uncertainty is important in order to determine how reliable the prediction is. Traditional DL techniques such as neural networks are incapable of capturing the uncertainty in the training data, thus they are overconfident about their estimates. On the contrary, Bayesian deep learning has recently emerged as a promising solution to account for uncertainty in the training process, achieving state-of-the-art performance in many classification and regression tasks. In this work Bayesian DL techniques such as Bayesian dense neural networks and Bayesian convolutional neural networks are applied to RUL estimation and compared to their frequentist counterparts from the literature. The effectiveness of the proposed models is verified on the popular C-MAPSS dataset. Furthermore, parameter uncertainty is quantified and used to gain additional insight into the data.

Keywords: Bayesian deep learning, C-MAPSS, Condition-based maintenance, Neural network, Remaining useful life, Uncertainty

1. Introduction

Condition-based maintenance (CBM) is a maintenance strategy that optimally plans maintenance operations based on real-time monitored health condition of machinery. In the recent years, it has become more and more popular thanks to its effectiveness in reducing unnecessary interventions and improving the reliability of machinery [1]. One of the most critical tasks of CBM is the estimation of the *remaining useful life* (RUL) of physical systems, i.e. "the available time before a failure occurs within a component or sub-component" [2].

Several methods exist to predict the RUL value. Traditionally, RUL prediction was accomplished through *model-based* and *statistical* approaches [3]. While the former describe the degradation process through mathematical models of the underlying failure mechanism, the latter fit a statistical model to available failure data, without relying on any physical principles. Recently, *AI-based* techniques have gained in popularity thanks to their ability to learn degradation patterns directly from observations [1]. In particular, *deep learning* (DL) has established itself as a powerful method to extract information from the data without any prior knowledge about the underlying phenomenon, at the cost of an increased computational power demand. DL techniques, such as neural networks (NNs) and their multiple variations, e.g. dense neural networks (DNNs), recurrent neural networks (RNNs), long short-term memory (LSTM) networks and convolutional neural networks (CNNs), have been successfully applied to RUL prediction, outperforming traditional prognostic algorithms in simulated turbofan engine degradation data benchmarks [4–7].

One of the major drawbacks of DL is that it heavily depends on the availability of a large *quantity* of labeled data [8], which are typically expensive and time-consuming to obtain, especially in industrial applications. The scarcity of data leads to uncertain estimates of the model's parameters, which in turn result in poor prognostic performance. This consideration suggests the need for a mechanism to quantify the uncertainty inherent in the training data, in order to determine how confident the output prediction is. Uncertainty modeling requires switching from a frequentist to a Bayesian perspective, i.e. assuming that the model parameters are random variables, whose prior distribution is updated according to Bayes' rule after encountering the data, to yield a posterior distribution [9]. While this is computationally affordable in model-based and statistical approaches, it is prohibitive in DL due to the large amount of parameters to update. Only recently, thanks to breakthroughs in the field of *Bayesian deep learning*, variational inference techniques have been proposed to approximate the intractable posterior distribution, making it possible to train Bayesian neural networks (BNNs) [10–13]. BNNs have been widely employed in computer vision, achieving state-of-the-art performance in many tasks [14], but, to the best of the author's knowledge, they have not been applied to RUL prediction yet.

The goal of this work is to test the effectiveness and investigate the advantages of BNNs in RUL estimation. In particular,

[☆]Sources of support: this research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

[*]Corresponding author.

Email address: luca.della@tum.de (L. Della Libera) the focus is on Bayesian dense neural networks (BDNNs) and Bayesian convolutional neural networks (BCNNs), and their performance is assessed on simulated run-to-failure turbofan engine degradation data.

The main contributions of this study are as follows:

- Bayesian DL techniques are successfully applied to the field of CBM, achieving competitive performance compared to the state-of-the-art.
- It is shown how the uncertainty provided by Bayesian deep models is useful to gain additional insight into the dataset for what concerns data quantity.
- The source code[1] is released to promote further research in the field.

The remainder of this paper is structured as follows. In Section 2 recent and related work on DL for RUL prediction is summarized . In Section 3 an overview of the theoretical foundations of BNNs is provided and the notion of uncertainty is formally defined. In Section 4 the regression models used in the experiments are introduced. In Section 5 the experimental setup is presented and the results are discussed. Finally, in Section 6 the conclusions are drawn and future work directions are outlined.

2. Related work

In this section recent DL approaches to estimate RUL from run-to-failure data are reviewed. The selected studies mainly focus on improving the predictive performance by means of more and more complex NN architectures. Additionally, two works that address the problem of data quantity in DL are mentioned.

In the last decade, considerable effort has been invested to advance the state-of-the-art of RUL prediction. In [15], Tian developed a DNN model to predict the RUL of a group of pumps based on vibration data collected from bearings. First, each condition monitoring measurement series is fitted by a generalization of the *Weibull distribution failure rate function* [16], in order to reduce the noise inherent in the data. Then, the age and the fitted condition monitoring measurement values at the present and previous observations are given as inputs to the network, which returns the life percentage as an output. The closer the life percentage is to 100%, the closer the pump is to its end-of-life. Experimental results showed that the proposed method achieved relatively low average prediction error rates, outperforming other techniques such as the modified version of the method by Wu et al. [17]. A major drawback of DNNs is that they have no built-in mechanism to capture *temporal* and *spatial* relationships, thus additional preprocessing (such as fitting the raw measurements using the generalized Weibull distribution failure rate function) is required to extract meaningful information from the data.

The issue of modeling temporal dependencies is solved, in principle, by RNNs. In [6], Gugulothu et al. proposed a novel RUL estimation approach – *Embed-RUL* – that uses a sequence-to-sequence RNN-based model to generate embeddings for multivariate time series. These embeddings tend to differ for normal and defective machines, thus they successfully capture the degradation pattern in the time series. Experiments performed on NASA's *International Conference on Prognostics and Health Management* (PHM08) prognostic data challenge dataset [18] and on a proprietary real-world pump dataset showed that *Embed-RUL* outperformed the previous state-of-the-art. Further improvements over RNN's capability of modeling temporal relationships are offered by LSTM networks, which capture not only short-term dependencies but also long-term ones. In [5], Zheng et al. proposed a DL architecture with 2 LSTM layers followed by 2 dense layers and a final output neuron. Extensive experiments on NASA's *Commercial Modular Aero-Propulsion System Simulation* (C-MAPSS) [19, 20], PHM08 and a milling dataset [21] demonstrated that LSTM networks were capable of revealing hidden patterns in the data, significantly outperforming former RUL estimation approaches.

The problem of capturing spatial dependencies among different sensor measurements is addressed by CNNs. In [4], Babu et al. proposed a novel CNN architecture for RUL prediction, which consists of 2 convolutional and average pooling layers followed by a final output neuron. This method improved the accuracy of the RUL predictions on the C-MAPSS dataset compared to a multilayer perceptron, a support vector machine, and a relevance vector machine. More recently, in [7], Li et al. achieved even better performance compared to both the LSTM network in [5] and the CNN in [4]. They employed a deep CNN architecture with 5 convolutional layers followed by 1 dense layer and a final output neuron. Additionally, they used *dropout* [22] as a regularization technique and they trained the network with the adaptive learning rate method *Adam* [23]. In both studies, a sliding window approach is adopted to take into account the temporal relationships within the time series, which would be otherwise ignored since no recurrent layer was used.

An attempt to combine the ability of LSTM networks to capture temporal dependencies with the ability of CNNs to extract hierarchical spatial features was made by Jayasinghe et al. in [24], with promising results for datasets obtained from complex environments.

All these studies rely on completely labeled run-to-failure training series. However, in real-life scenarios, most data are unlabeled from the beginning and the labeling process is expensive, time-consuming and error-prone. In order to reduce the amount of labeled data needed to train the model, semi-supervised DL techniques based on generative models and restricted Boltzmann machines have recently been applied with success by Yoon et al. [25] and Ellefsen et al. [2], respectively. Differently from them, in this work the focus is not on proposing alternative solutions to the issue of data labeling, but on determining whether enough labeled data were provided to effectively train the model.

[1] See https://github.com/luca310795/bayesian-deep-rul

3. Theoretical background

In this section the theoretical foundations of BNNs are introduced and the notion of uncertainty is formally defined. For a more rigorous mathematical discussion see [10, 13, 14].

3.1. Bayesian neural networks

In standard NNs uncertainty is completely neglected. Weights assume deterministic values and output predictions are point estimates [10]. On the contrary, in BNNs weights are represented by probability distributions over possible values and output predictions are random variables. If we know the posterior distribution of the weights given the training data, $p(\mathbf{w} \mid \mathcal{D})$, the predictive distribution of a label $\hat{y}$ of an unseen test sample $\hat{\mathbf{x}}$ is calculated as $p(\hat{y} \mid \hat{\mathbf{x}}) = \mathbb{E}_{p(\mathbf{w}|\mathcal{D})}[p(\hat{y} \mid \hat{\mathbf{x}}, \mathbf{w})]$, i.e. the expected prediction under each possible configuration of the weights given $\hat{\mathbf{x}}$, which is equivalent to using an ensemble of an uncountably infinite number of NNs [10].

Theoretically, $p(\mathbf{w} \mid \mathcal{D})$ can be obtained through exact Bayesian inference, but practically it is infeasible as the number of weights in an NN is very large and its functional form is not suitable for integration [10]. A possible solution is to approximate the posterior distribution of the weights through variational inference, a technique that determines the optimal parameters $\boldsymbol{\theta}^\star$ of a surrogate distribution $q(\mathbf{w} \mid \boldsymbol{\theta})$ (which is often assumed to be a diagonal Gaussian) that minimizes the Kullback-Leibler (KL) divergence with the true posterior of the weights [10]:

$$\begin{aligned}\boldsymbol{\theta}^\star &= \arg\min_{\boldsymbol{\theta}} \mathrm{KL}[q(\mathbf{w} \mid \boldsymbol{\theta}) \parallel p(\mathbf{w} \mid \mathcal{D})] \\ &= \arg\min_{\boldsymbol{\theta}} \int q(\mathbf{w} \mid \boldsymbol{\theta}) \log \frac{q(\mathbf{w} \mid \boldsymbol{\theta})}{p(\mathbf{w})p(\mathcal{D} \mid \mathbf{w})} \mathrm{d}\mathbf{w} \\ &= \arg\min_{\boldsymbol{\theta}} \mathrm{KL}\left[q(\mathbf{w} \mid \boldsymbol{\theta}) \parallel p(\mathbf{w})\right] - \mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})}\left[\log p(\mathcal{D} \mid \mathbf{w})\right],\end{aligned} \tag{1}$$

where $p(\mathbf{w})$ is the prior distribution of the weights. The resulting cost function is known as the evidence lower bound (ELBO) or the variational free energy (VFE) and it explicitly represents the trade-off between satisfying the simplicity of the prior (complexity cost) and the complexity of the data (likelihood cost) [10]. It is denoted as

$$\mathcal{F}(\mathcal{D}, \boldsymbol{\theta}) = \underbrace{\mathrm{KL}\left[q(\mathbf{w} \mid \boldsymbol{\theta}) \parallel p(\mathbf{w})\right]}_{\text{complexity cost}} - \underbrace{\mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})}\left[\log p(\mathcal{D} \mid \mathbf{w})\right]}_{\text{likelihood cost}}. \tag{2}$$

Exactly minimizing the ELBO is computationally intractable in many cases. Instead, various approximations are adopted. In [10], a backpropagation-like algorithm is proposed for Bayesian variational inference in NNs – *Bayes by Backprop* – which uses Monte Carlo sampling to draw weights from the surrogate distribution $q(\mathbf{w} \mid \boldsymbol{\theta})$ and obtain unbiased estimates of gradients of the cost function in Eq. 2. The exact ELBO is approximated as [10]:

$$\mathcal{F}(\mathcal{D}, \boldsymbol{\theta}) \approx \sum_{i=1}^{T} \log q(\mathbf{w}^{(i)} \mid \boldsymbol{\theta}) - \log p(\mathbf{w}^{(i)}) - \log p(\mathcal{D} \mid \mathbf{w}^{(i)}), \tag{3}$$

where $\mathbf{w}^{(i)}$ denotes the i-th Monte Carlo sample drawn from the surrogate distribution. Gradient-based optimization can be used to minimize the approximated ELBO and learn the optimal parameters of the surrogate distribution $q(\mathbf{w} \mid \boldsymbol{\theta})$ of the weights in an NN.

Bayes by Backprop can be applied to CNNs too, in order to approximate the intractable posterior distribution of the weights in each convolutional filter. In [13], Shridhar et al. assume a diagonal Gaussian surrogate distribution, i.e. $w_{i,j,h,w} \sim \mathcal{N}(\mu_{i,j,h,w}, \sigma^2_{i,j,h,w})$, where i and j are the indices of the input and output layers, respectively, h and w the height and width of any given filter, respectively, $\mu_{i,j,h,w}$ and $\sigma^2_{i,j,h,w}$ the mean and variance of the distribution, respectively, which need to be learned by the network. In particular, to facilitate the learning process, they define $\sigma^2_{i,j,h,w} = \alpha_{i,j,h,w}\mu^2_{i,j,h,w}$, where $\alpha_{i,j,h,w}$ is a learnable parameter that intuitively represents the deviation from the mean.

Differently from [10], they utilize the *local reparameterization trick* [26] to sample the layer activation b instead of the weights for a higher computational efficiency. The mean and variance of each weight are learned independently through two convolutional operations. In the first, the output b (which is a function of both the mean and the variance) is interpreted as the mean $\mu_{i,j,h,w}$ and optimized towards a single point estimate. In the second, the variance $\alpha_{i,j,h,w}\mu^2_{i,j,h,w}$ is learned. Since this formulation of the variance includes the mean $\mu_{i,j,h,w}$, only $\alpha_{i,j,h,w}$ must be learned in the second convolution, hence only one parameter is updated per convolutional operation. This procedure can be applied to dense layers too [13].

Adopting a Bayesian perspective in DL leads to the following advantages:

- Regularization is naturally provided, since a prior is placed upon the weights.
- Predictive uncertainty can be quantified, as the output value is not a point estimate but a probability distribution.
- An infinite ensemble of networks is trained implicitly.

However, these advantages come at a cost: if we assume a diagonal Gaussian surrogate distribution, the number of weights in a BNN is doubled compared to a frequentist NN of the same size [10], and training takes longer since at each step two operations are performed instead of one and the expensive Monte Carlo sampling is applied to approximate the ELBO.

3.2. Uncertainty in Bayesian deep learning

In Bayesian modeling there are two main types of uncertainty: *epistemic* and *aleatoric* [14]. Epistemic uncertainty represents the uncertainty in the model's parameters. It is important when the data are scarce, since it can be explained away only if enough training samples are provided. Aleatoric uncertainty captures noise inherent in the observations such as sensor or motion noise and cannot be reduced even if more data are available. It is important in large data situations, where epistemic uncertainty vanishes.

In regression tasks, the uncertainty of a Bayesian deep model (e.g. a BNN) with respect to the random output $y = f^{\mathbf{w}}(\mathbf{x})$ (where $\mathbf{x}$ is the input sample and $f^{\mathbf{w}} : \mathrm{X} \to \mathrm{Y}$ is the mapping between input space and output space defined by the model's weights $\mathbf{w}$) is quantified by the predictive variance, which can be approximated as [14]:

$$\mathrm{Var}(y) \approx \sigma^2 + \frac{1}{T}\sum_{i=1}^{T}(f^{\mathbf{w}^{(i)}}(\mathbf{x}) - \mathbb{E}[y])^2, \qquad (4)$$

with predictions obtained by approximating the predictive mean $\mathbb{E}[y] \approx \frac{1}{T}\sum_{i=1}^{T} f^{\mathbf{w}^{(i)}}(\mathbf{x})$, where $\mathbf{w}^{(i)}$ is the i-th Monte Carlo sample drawn from the posterior distribution of the weights. The first term in the predictive variance, σ^2, corresponds to the aleatoric uncertainty, i.e. the amount of noise inherent in the data; the second is the epistemic uncertainty, which measures how confident the model is about its predictions. This term vanishes when the parameter uncertainty is zero (i.e. all draws $\mathbf{w}^{(i)}$ assume the same constant value). As mentioned in Section 1, estimating the RUL in industrial applications is often problematic due to the scarcity of labeled data, thus it is more useful to quantify epistemic rather than aleatoric uncertainty. For this reason, in this work the focus is on the epistemic uncertainty, and the aleatoric one is neglected (i.e. $\sigma^2 = 0$ in Eq. 4).

4. Regression models

In this section the proposed regression models for RUL prediction are introduced. As mentioned in Section 1, the goal of this work is to test the effectiveness of BDNNs and BCNNs in RUL estimation and investigate their advantages over standard NNs. For the sake of a fair comparison, three frequentist DL models are considered as a baseline and compared to their Bayesian counterparts, for a total of six models. Frequentist models are trained through backpropagation with mean square error (MSE) as a loss function, Bayesian models through Bayes by Backprop with ELBO (see Eq. 3). Prior and surrogate distributions of the weights are initialized as diagonal Gaussians $\mathcal{N}(0, 0.05)$ and $\mathcal{N}(\mu_q, \alpha\mu_q^2)$, respectively, where the learnable parameters μ_q and α are initialized through Xavier uniform initializer [27]. The input is assumed to be an $N_t \times N_f$ matrix, where N_t represents the time sequence dimension and N_f the number of features. The procedure to prepare the data in 2D format is discussed in detail in Section 5.2.3.

The models are as follows:

- *Frequentist-Dense3* (F-D3): standard DNN architecture (see Fig. A.6a). The input matrix is flattened into a 1D vector, which is fed to 3 consecutive 100 neurons dense layers. After each dense layer, the sigmoid activation is applied. A final output neuron returns the predicted RUL value.

- *Bayesian-Dense3* (B-D3): Bayesian counterpart of F-D3, in which standard dense layers are replaced by Bayesian ones (see Fig. A.6b). The softplus activation (whose output is strictly positive) is applied to the final output neuron, in order to ensure a positive non-zero variance [13].

- *Frequentist-Conv2Pool2* (F-C2P2): CNN architecture proposed in [4] by Babu et al. (see Fig. A.6c). The input matrix is fed to a 5×14 convolutional layer with 8 channels followed by a 2×1 average pooling layer. A second 2×1 convolution with 14 channels is applied, followed by another 2×1 average pooling. The output is flattened into a 1D vector and a final output neuron returns the predicted RUL value. After each convolutional layer, the sigmoid activation is applied.

- *Bayesian-Conv2Pool2* (B-C2P2): Bayesian counterpart of F-C2P2, in which standard convolutional and dense layers are replaced by Bayesian ones (see Fig. A.6d). As in B-D3, the softplus activation is applied to the final output neuron.

- *Frequentist-Conv5Dense1* (F-C5D1): CNN architecture proposed in [7] by Li et al. (see Fig. A.6e). The input matrix is fed to 4 consecutive 10×1 convolutional layers with 10 channels followed by a 3×1 convolution with 1 channel. The output is flattened into a 1D vector and a 100 neurons dense layer is applied, followed by a final output neuron that returns the RUL value. After each convolutional layer, the tanh activation is used. The input is zero-padded before each convolution to keep the dimensions unaltered. Dropout with $p = 0.5$ is applied to the flattened 1D vector to reduce overfitting. Xavier normal initializer [27] is used to initialize the weights. In [7], an additional tanh is applied to the output of the 100 neurons dense layer; it had to be removed because in the proposed implementation it led to gradient saturation, preventing the network from training properly.

- *Bayesian-Conv5Dense1* (B-C5D1): Bayesian counterpart of F-C5D1, in which standard convolutional and dense layers are replaced by Bayesian ones (see Fig. A.6f). To ensure a positive non-zero variance, the 5 tanh activations are replaced by 4 sigmoids (whose output is strictly positive) and 1 softplus, which is also applied to the 100 neurons dense layer and to the final output neuron. Sigmoid instead of softplus is used in the first 4 convolutions since a performance improvement was empirically observed. No more than 4 sigmoids are applied since experiments showed that exceeding that threshold leads to gradient saturation. Dropout is removed due to the natural regularization provided by the Bayesian approach.

The proposed models were implemented in PyTorch [28], using the open source implementation of Bayesian layers[2] provided by Shridhar et al. along with [13], and the data loader[3] provided by Qi et al. along with [29]. Note that, since the current implementation of Bayesian layers does not support bias, in order to enable a fair comparison it is not used in frequentist layers either.

[2]See https://github.com/kumar-shridhar/PyTorch-BayesianCNN, last accessed on March 14, 2019.

[3]See https://github.com/charlesq34/pointnet2, last accessed on March 14, 2019.

5. Experimental study

In this section the dataset is introduced, the data preprocessing phase is presented in detail, the performance metrics considered in the experiments are defined, the prognostic procedure is described and the experimental results are discussed.

5.1. Dataset

The proposed models are tested on the simulated turbofan engine degradation data of the publicly available C-MAPSS dataset, which includes four subsets of multivariate temporal data obtained from 21 sensors. Each subset contains a training and a test set. The training set includes run-to-failure sensor records of multiple engines collected under different operating conditions and fault modes. At the beginning, the engines are assumed to be healthy but their initial wear and manufacturing variation is unknown. As time progresses, they start to degrade until a system failure occurs. The last data entry corresponds to the time cycle at which the unit is declared defective. On the contrary, the sensor records in the test set end at some point before the failure. The goal is to predict the RUL of each engine in the test set. For verification, the true RUL values of the test engines are provided. Each subset includes 26 columns: engine number, time cycle, 3 operational settings (that determine the operating condition) and 21 sensor measurements. The detailed information of the four subsets, denoted as FD001, FD002, FD003 and FD004, is presented in Table 1.

5.2. Data preprocessing

A preprocessing phase is required before feeding the data to the model. It consists of four steps: *feature selection*, *normalization*, *sliding window segmentation* and *label rectification*. The parameters of the preprocessing phase are reported in Table 1. The resulting specifics of the proposed regression models are reported in Table 2.

5.2.1. Feature selection

In FD001 and FD003, sensors 1, 5, 6, 10, 16, 18, and 19 exhibit constant measurements throughout the whole engine's lifetime, thus they do not provide any useful information to predict the RUL [2]. Moreover, FD001 and FD003 are subjected to a single operating condition. For these reasons, only 14 out of 21 sensors are used as the input features, whose indices are 2, 3, 4, 7, 8, 9, 11, 12, 13, 14, 15, 17, 20 and 21, and the 3 operational settings are discarded. On the contrary, in FD002 and FD004 there are no constant measurements, and the presence of 6 operating conditions makes it harder to detect degradation patterns [2]. Therefore, all 21 sensors and 3 operational settings are used as the input features.

5.2.2. Normalization

The collected sensor data are normalized to be within the range $[-1, 1]$ by means of *min-max* normalization [30]:

$$x^{i,j}_{norm} = \frac{2(x^{i,j} - x^{j}_{min})}{x^{j}_{max} - x^{j}_{min}} - 1, \quad \forall\, i, j, \tag{5}$$

Table 1
The C-MAPSS dataset and the parameters of the preprocessing phase.

Information	C-MAPSS			
	FD001	FD002	FD003	FD004
Training series	100	260	100	249
Test series	100	259	100	248
Operating conditions	1	6	1	6
Fault modes	1	1	2	2
Parameter				
N_t	30	20	30	15
N_f	14	24	14	24
Training samples	17,731	48,819	21,820	57,763
Test samples	100	259	100	248
R_{early}	125	125	125	125

Table 2
Number of trainable layers and weights in each model.

Model	Layers	Weights			
		FD001	FD002	FD003	FD004
F-D3	4	62,100	68,100	62,100	56,100
B-D3	4	124,200	136,200	124,200	112,200
F-C2P2	3	868	1,246	868	1,092
B-C2P2	3	1,736	2,492	1,736	2,184
F-C5D1	7	45,230	51,230	45,230	39,230
B-C5D1	7	90,460	102,460	90,460	78,460

where $x^{i,j}$ denotes the original i-th data point of the j-th sensor, $x^{i,j}_{norm}$ the normalized value of $x^{i,j}$, x^{j}_{max} the maximum value of the original measurement data from the j-th sensor and x^{j}_{min} the minimum value.

5.2.3. Sliding window segmentation

In multivariate time series-based problems, we can generally extract more information by analyzing the temporal sequence data, rather than just considering the single data points sampled at each time step independently. In order to preserve the degradation patterns hidden in the temporal dimension, the sliding window approach of Li et al. [7] is adopted to divide the time series into overlapping segments of fixed size. Let N_t denote the size of the time window and N_f the number of features (i.e. sensors). At each time step, all the future sensor records within the time window are collected into an $N_t \times N_f$ matrix (see Fig. 1). In particular, given a time series of length L, we can extract exactly $L - N_t + 1$ segments. If $L < N_t$, the time series is discarded. The obtained segments are then labeled with the RUL of the last data point in the time window and used as input samples to feed the model.

5.2.4. Label rectification

A common approach in the literature [4, 5, 7, 31] is to adopt a piece-wise linear degradation model, i.e. to assume that the RUL target function is constant until a threshold value, R_{early},

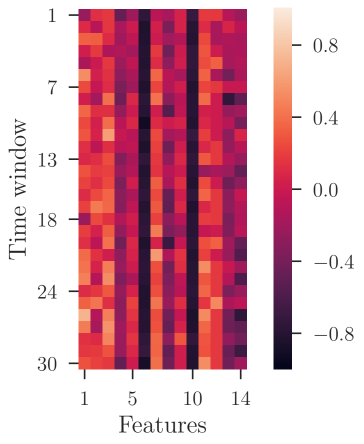


Fig. 1. Normalized FD001 test sample from the 14 selected sensors within a time window of size 30.

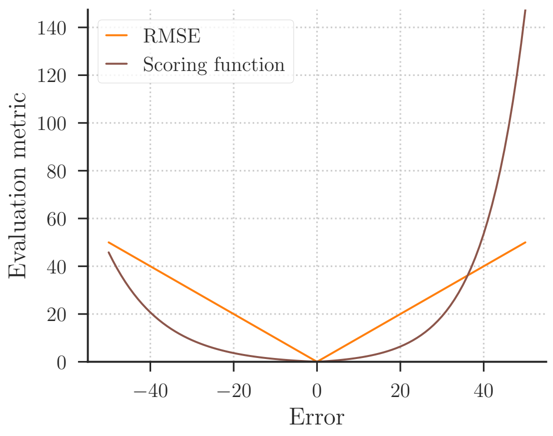


Fig. 2. Comparison between the RMSE and the scoring function with respect to different error values.

after which it linearly decreases to 0. From an implementation perspective, this translates into rectifying the labels, i.e. to artificially set the RUL to R_{early} for all those samples whose RUL is strictly greater than R_{early}. The rationale behind it is that a system typically works normally in the early age and starts to degrade only after a certain degree of wear. However, the value of R_{early} varies across the literature and label rectification is often applied to both the training and the test set (e.g. in [7]), making it difficult to compare the prognostic performance of different methods. Modifying the test labels according to R_{early} is indeed equivalent to creating an ad hoc version of the test set on which the performance metrics are biased towards over-optimistic estimates. In order to facilitate the comparison with current and future studies, the results both with and without rectified test labels are reported.

5.3. *Performance metrics*

In this study three metrics were considered for evaluating the performance of the proposed regression models, i.e. root mean square error (RMSE), mean absolute error (MAE) and the scoring function (S) proposed in [20] by Saxena et al. and also used in the PHM08 prognostic data challenge dataset [18], which are formulated as follows:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} d_i^2}, \tag{6}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |d_i|, \tag{7}$$

$$\text{S} = \sum_{i=1}^{n} e^{s_i} - 1, \quad \text{where} \quad s_i = \begin{cases} -\frac{d_i}{13}, & \text{for } d_i < 0 \\ \frac{d_i}{10}, & \text{for } d_i \geq 0 \end{cases}, \tag{8}$$

where n is the total number of test samples and d_i is the error (i.e. the difference) between the estimated RUL value and the true RUL value for the i-th test sample. Good prognostic methods should obtain relatively low values on all the three metrics.

As shown in Fig. 2, the RMSE equally penalizes early and late predictions, while the asymmetric scoring function penalizes late predictions more than early ones. Minimizing S is important because late predictions typically lead to more serious consequences than early predictions, as the maintenance will be scheduled too late. However, in order to correctly evaluate the performance, it is useful to monitor RMSE and MAE too, since S is extremely sensitive to outliers due to the exponentiation.

In addition to the three evaluation metrics, in Bayesian models the epistemic uncertainty (see Section 3.2), averaged over all the test samples, is considered too.

5.4. *Prognostic procedure*

The proposed prognostic procedure is based on [7] and consists of the following steps (see Fig. 3):

- The dataset is preprocessed as described in Section 5.2. Note that the data prepared in 2D format are fed directly to the model, without the need of any prior feature engineering step.
- Based on the specific dataset information, the regression model is initialized. For each epoch, the training samples and their corresponding RUL labels are randomly divided into multiple mini-batches of size 512 and fed to the model. The model's weights are updated according to the backpropagated gradients of the loss function computed batchwise. The Adam [23] optimization algorithm is used to train the model for 250 epochs with a base learning rate of 0.001. After 200 epochs, the learning rate is reduced by a factor of 10 for fine-tuning. In Bayesian models, only 1 Monte Carlo sample is drawn from the posterior distribution to compute the loss[4].

[4]Drawing more than 1 sample at training time is too computationally expensive on the hardware used in this work.

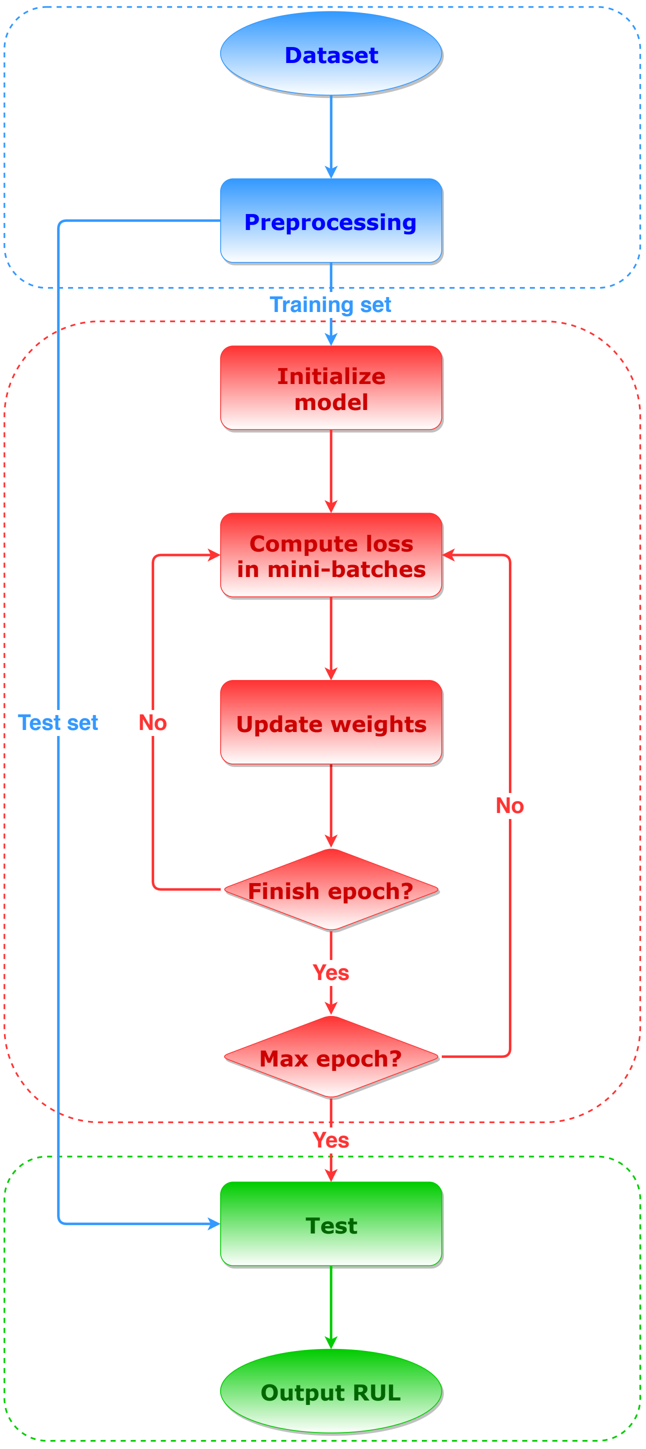


Fig. 3. Flowchart of the proposed prognostic procedure (adapted from [7]).

- The test samples are fed to the trained model to predict the RUL values and calculate the performance metrics. In Bayesian models, 150 Monte Carlo samples are drawn from the posterior distribution to calculate the performance metrics[5].

[5]Since testing is a one-time operation, we can afford to draw many samples to calculate the performance metrics with higher precision.

Table 3
Hyperparameters of the proposed prognostic procedure.

Hyperparameter	Value
Mini-batch size	512
Max epoch	250
Base learning rate	0.001
Learning rate decay factor	10
Learning rate decay epoch	200
Monte Carlo samples for training	1
Monte Carlo samples for testing	150

The hyperparameters of the proposed prognostic procedure are reported in Table 3.

5.5. *Experimental results and discussion*

In this section the prognostic performance of the proposed regression models is evaluated and the importance of epistemic uncertainty in assessing the quantity of training data is discussed. The experimental results are averaged over 10 trials to reduce the effect of statistical fluctuations, and mean and standard deviation values are reported. All the experiments were run on a publicly available Google Compute Engine Deep Learning VM instance with 2 vCPUs, 13 GB RAM, 1 NVIDIA Tesla K80 GPU and *PyTorch 1.2 + fast.ai 1.0 (CUDA 10.0)* framework.

5.5.1. *Performance evaluation*

As shown in Table 4, in most of the cases Bayesian models perform better (with respect to RMSE, MAE and score) than their frequentist counterparts. This is probably due to the natural regularization effect provided by the Bayesian approach, which helps to reduce the generalization error. In particular, B-C5D1 achieves the best RMSE in FD001, FD002 and FD004 (as expected, since its frequentist counterpart accomplishes state-of-the-art performance [7]), and in FD002 and FD004 it even improves the rectified state-of-the-art metrics reported in [2]. B-D3, despite its simplicity, achieves the best RMSE in FD003. Since B-D3 has more parameters than B-C5D1 but no convolutional layers, we conclude that, comparing to the other subsets, the temporal dimension of the data in FD003 is less informative, i.e. no clear degradation pattern can be detected. This leads to the conclusion that on FD003 larger dense models are preferable to smaller models that use convolutional layers in the attempt to extract useful information from the run-to-failure series. More generally, the performance improvement deriving from extracting the temporal information (through the convolutional layers in combination with the sliding window segmentation approach, see Section 5.2.3) is moderate in the other subsets too. This is a factor to take into consideration, especially when the computational power is limited, since the time required for training B-C5D1 is enormously larger than B-D3 (see Fig. 4), and large training times make it difficult to properly tune the hyperparameters.

Table 4
Performance evaluation results averaged over 10 trials. Metrics followed by "∗" are computed on the rectified test data. Mean and standard deviation (std) values are reported. The best values of the metrics are highlighted in bold. Values of the rectified metrics that improve the state-of-the-art in [2] are $\boxed{\text{framed}}$.

Subset	Metric	F-D3		B-D3		F-C2P2		B-C2P2		F-C5D1		B-C5D1		[2]
		mean	std	mean	std	mean	std	mean	std	mean	std	mean	std	mean
FD001	RMSE	14.86	0.27	15.53	0.49	27.79	0.01	23.06	1.07	14.51	0.16	**14.48**	0.21	–
	MAE	10.89	0.30	11.30	0.36	22.72	0.01	17.61	1.25	11.23	0.17	**10.83**	0.10	–
	Score $\cdot 10^{-2}$	4.43	0.31	6.43	0.58	45.70	0.41	28.12	7.33	**3.14**	0.09	3.46	0.28	–
	RMSE*	13.83	0.28	14.66	0.52	26.72	0.01	22.02	1.08	13.13	0.15	13.07	0.22	**12.56**
	MAE*	9.82	0.30	10.23	0.36	21.65	0.01	16.54	1.25	10.17	0.17	**9.80**	0.10	–
	Score$^* \cdot 10^{-2}$	4.16	0.31	6.18	0.55	44.98	0.41	27.62	7.31	2.75	0.10	2.99	0.25	**2.31**
	Epistemic	0.00	0.00	8.37	0.98	0.00	0.00	2.96	1.26	0.00	0.00	11.24	1.78	–
FD002	RMSE	32.04	0.59	32.33	0.61	31.43	0.23	31.34	0.23	35.29	0.57	**29.87**	0.12	–
	MAE	22.80	0.46	23.34	0.55	22.95	0.29	23.06	0.23	26.65	0.50	**21.42**	0.14	–
	Score $\cdot 10^{-2}$	336.42	140.99	371.89	147.80	168.40	11.92	**161.40**	8.46	286.60	24.08	172.26	10.58	–
	RMSE*	21.10	0.67	21.91	0.72	19.69	0.22	19.70	0.32	23.38	0.44	$\boxed{\mathbf{18.06}}$	0.11	22.36
	MAE*	15.30	0.46	15.85	0.55	15.72	0.27	15.76	0.30	19.31	0.44	**14.43**	0.15	–
	Score$^* \cdot 10^{-2}$	95.01	103.98	90.34	43.58	22.47	1.39	26.03	2.72	36.63	5.00	$\boxed{\mathbf{18.62}}$	1.41	33.66
	Epistemic	0.00	0.00	0.37	0.06	0.00	0.00	0.13	0.01	0.00	0.00	0.72	0.07	–
FD003	RMSE	14.34	0.32	**14.10**	0.33	28.86	0.02	24.23	1.09	15.20	0.36	14.70	0.48	–
	MAE	10.46	0.30	**10.05**	0.32	24.14	0.01	19.15	1.29	11.89	0.40	11.02	0.37	–
	Score $\cdot 10^{-2}$	4.26	0.28	4.16	0.43	36.30	0.13	26.48	1.85	**3.56**	0.23	4.15	0.51	–
	RMSE*	13.05	0.37	12.72	0.35	27.62	0.02	23.07	1.05	13.59	0.42	13.28	0.58	**12.10**
	MAE*	8.93	0.30	**8.49**	0.32	22.58	0.01	17.59	1.29	10.49	0.43	9.51	0.39	–
	Score$^* \cdot 10^{-2}$	3.93	0.28	3.81	0.42	35.47	0.13	25.86	1.81	3.09	0.24	3.79	0.53	**2.51**
	Epistemic	0.00	0.00	4.39	0.78	0.00	0.00	1.41	0.57	0.00	0.00	9.18	2.50	–
FD004	RMSE	33.02	0.62	33.64	0.49	32.97	0.36	33.13	0.52	36.21	1.71	**31.35**	0.15	–
	MAE	25.23	0.36	25.72	0.50	25.64	0.34	25.79	0.62	29.38	1.57	**24.35**	0.23	–
	Score $\cdot 10^{-2}$	188.94	41.97	232.27	41.43	121.98	8.84	118.64	8.21	180.73	51.06	**85.78**	2.94	–
	RMSE*	22.80	0.63	23.51	0.60	22.30	0.43	22.44	0.59	25.64	1.76	$\boxed{\mathbf{20.34}}$	0.23	22.66
	MAE*	16.53	0.36	17.03	0.50	17.05	0.33	17.29	0.63	20.93	1.57	**15.73**	0.25	–
	Score$^* \cdot 10^{-2}$	104.50	33.78	122.13	43.22	57.12	5.63	54.47	7.46	72.54	30.37	30.15	2.36	**28.40**
	Epistemic	0.00	0.00	0.26	0.06	0.00	0.00	0.08	0.01	0.00	0.00	0.49	0.05	–

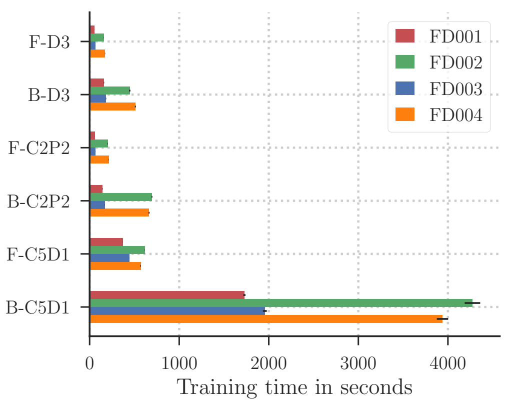


Fig. 4. Average training times on the C-MAPSS dataset.

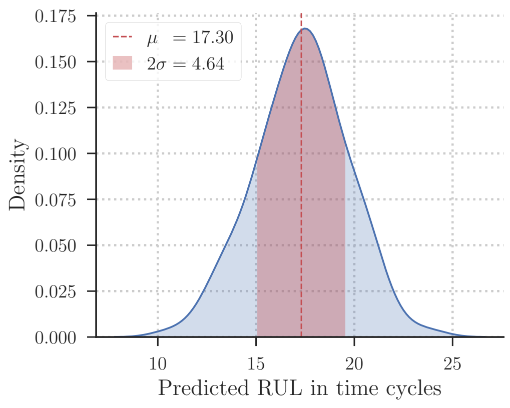


Fig. 5. Predictive distribution of the FD001 test sample shown in Fig. 1, generated by sampling 150 times the output of a B-C5D1 model. Mean (μ) and standard deviation (σ) values are reported. The true RUL value is 15.

Table 5
Performance evaluation results of the most successful Bayesian models for different quantity percentages, averaged over 10 trials. Mean and standard deviation (std) values are reported.

Model	Subset	Metric	Quantity							
			100%		50%		25%		12.5%	
			mean	std	mean	std	mean	std	mean	std
B-C5D1	FD001	RMSE	14.48	0.21	14.63	0.11	15.44	0.43	17.58	1.62
		MAE	10.83	0.10	10.72	0.14	11.53	0.38	13.36	1.28
		Score $\cdot 10^{-2}$	3.46	0.28	3.73	0.20	3.80	0.17	5.50	2.02
		Epistemic	11.24	1.78	40.55	9.64	76.74	21.88	129.16	34.58
B-C5D1	FD002	RMSE	29.87	0.12	30.58	0.16	31.87	0.17	35.69	1.70
		MAE	21.42	0.14	21.68	0.08	23.32	0.18	27.02	1.70
		Score $\cdot 10^{-2}$	172.26	10.58	188.96	8.88	202.23	5.56	276.86	46.81
		Epistemic	0.72	0.07	9.46	1.75	42.35	4.86	115.59	15.72
B-D3	FD003	RMSE	14.10	0.33	14.30	0.15	17.07	0.45	27.93	2.52
		MAE	10.05	0.32	10.63	0.18	12.99	0.38	20.85	1.71
		Score $\cdot 10^{-2}$	4.16	0.43	3.60	0.12	5.64	0.26	16.99	5.79
		Epistemic	4.39	0.78	19.59	3.20	45.16	7.61	47.33	14.33
B-C5D1	FD004	RMSE	31.35	0.15	32.48	0.88	35.85	0.80	41.57	2.54
		MAE	24.35	0.23	25.51	0.73	28.71	0.82	34.74	2.63
		Score $\cdot 10^{-2}$	85.78	2.94	95.83	15.15	162.57	18.40	332.89	110.89
		Epistemic	0.49	0.05	7.33	1.00	44.95	5.81	117.73	18.21

For what concerns the comparison between the four subsets (see Table 4), a substantial performance drop is observed in FD002 and FD004, which is in line with expectations, since, as mentioned in Section 5.2.1, it is harder to detect degradation patterns due to the presence of 6 operating conditions. Moreover, the values of epistemic uncertainty decrease inversely with the number of training samples (see Table 1): the more the training data, the lower the epistemic uncertainty. In particular, the epistemic uncertainty in FD001 and FD003 is considerably higher than FD002 and FD004. Interestingly, if we sort the models by increasing values of epistemic uncertainty, the order is the same in all the subsets (B-C2P2 < B-D3 < B-C5D1). Investigating the dependency between epistemic uncertainty and network architecture, we notice that the former is more tightly correlated with the number of layers than the number of weights (see Table 2). The explanation is that weight uncertainty propagates through the layers, thus deep networks require more training data than shallow ones to be equally confident about their predictions. This confirms once again the fact that deep networks are harder to train than shallow ones.

Aside from performance considerations, the real advantage of Bayesian models is their capability of returning a probability distribution (see Fig. 5) instead of a point estimate as an output. The additional information provided by the predictive distribution enables us to boost the model in different ways. For example, if we use the predictive mean as the RUL estimate, we can define a threshold for the predictive standard deviation above which the estimate must be rejected because the model is not confident enough about it. Another possibility is to adopt a more conservative prognostic policy by using the p-th percentile (with low values of p) as the RUL estimate, in order to reduce the risk of late prediction.

5.5.2. *Data quantity*

As pointed out in Section 5.5.1, FD001 and FD003 are characterized by a higher epistemic uncertainty, reasonably due to the scarcity of training data. In order to validate this explanation, the most successful Bayesian models (B-C5D1 on FD001, FD002 and FD004, B-D3 on FD003) were retrained by progressively decreasing the data quantity. The results are reported in Table 5. A quantity percentage of $p\%$ indicates that only $p\%$ of the training data, chosen at random, were used for training. In line with the theory, a strong correlation exists between epistemic uncertainty, data quantity and performance: the fewer the data, the higher the uncertainty[6], the worse the performance. Surprisingly, if we halve the amount of training data, the performance degradation is negligible. Similarly, if we use a quarter of the data, the performance degradation is still moderate. A relevant performance drop is observed only when we use 12.5% of the data. If we analyze these results in relation to the values of epistemic uncertainty, we gain additional insight into the data. In particular, we conclude the following:

[6]Except for B-D3, where the epistemic uncertainty saturates at 25%, remaining almost unchanged at 12.5% due to the shallowness of the model.

- The information contained in the data is redundant, since performance degrades only marginally when the number of training samples is decreased. This means that the preprocessing procedure is not as effective as expected, as it fails in extracting important information from the raw data.
- Performance on FD001 and FD003 would benefit from enlarging the training set with new run-to-failure series, since epistemic uncertainty is 6 to 23 times higher than FD002 and FD004, which means that the training data are scarce. Data augmentation techniques could help in increasing the size of the training set.
- Performance on FD002 and FD004 would not benefit from enlarging the training set with new run-to-failure series, since epistemic uncertainty is already close to zero, which means that the training data are enough.

6. Conclusion and future work

In this work Bayesian deep learning was successfully applied to RUL estimation in the context of condition-based maintenance. The notion of epistemic uncertainty was introduced and it was shown how to quantify it by means of the Bayesian framework. In the experimental study, the proposed regression models were tested on the four subsets of the publicly available C-MAPSS dataset. Results demonstrated that Bayesian models achieve competitive performance compared to their frequentist counterparts, and that epistemic uncertainty is useful to determine whether the data are too scarce. In particular, if we halve the number of training samples, the increase in the epistemic uncertainty is only moderate and the performance degradation negligible. This means that the preprocessing procedure fails in extracting important information from the raw data.

Future work that aims at improving the state-of-the-art of RUL prediction should focus on refining the preprocessing phase. In addition to this, more effort should be spent on tuning the hyperparameters, especially for Bayesian models. In Bayesian models the number of hyperparameters is indeed larger than in frequentist ones (e.g. prior distribution, surrogate distribution, etc.), hence we have more combinations to explore. Another possible research direction that builds directly upon this work consists in applying the Bayesian RNN introduced in [12] by Fortunato et al. to RUL estimation, in order to combine the effectiveness of recurrent architectures in capturing temporal dependencies with the capability of BNNs to measure weight uncertainty. Computational power limitations due to the expensive Monte Carlo sampling could be overcome through sampling-free uncertainty estimation approaches such as the one proposed in [32] by Postels et al. Finally, since BNNs proved successful especially in classification tasks [10, 13], it would be interesting to cast the RUL prediction problem as a classification one (since labels are integer and can be capped at R_{early}), so as to harness all the benefits of BNNs. One of the advantages of BNNs for classification is that, differently from regression, aleatoric uncertainty can be easily quantified [33], which is useful in large data situations.

References

[1] Y. Lei, N. Li, L. Guo, N. Li, T. Yan, J. Lin, Machinery health prognostics: a systematic review from data acquisition to RUL prediction, Mechanical Systems and Signal Processing 104, 2018, pp. 799–836.
URL https://doi.org/10.1016/j.ymssp.2017.11.016

[2] A. L. Ellefsen, E. Bjørlykhaug, V. Æsøy, S. Ushakov, H. Zhang, Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture, Reliability Engineering & System Safety 183, 2019, pp. 240–251.
URL https://doi.org/10.1016/j.ress.2018.11.027

[3] A. Jardine, D. Lin, D. Banjevic, A review on machinery diagnostics and prognostics implementing condition-based maintenance, Mechanical Systems and Signal Processing 20, 2006, pp. 1483–1510.
URL https://doi.org/10.1016/j.ymssp.2005.09.012

[4] G. Sateesh Babu, P. Zhao, X.-L. Li, Deep convolutional neural network based regression approach for estimation of remaining useful life, in: S. B. Navathe, W. Wu, S. Shekhar, X. Du, X. S. Wang, H. Xiong (Eds.), Database Systems for Advanced Applications, Springer International Publishing, Cham, 2016, pp. 214–228.
URL https://doi.org/10.1007/978-3-319-32025-0_14

[5] S. Zheng, K. Ristovski, A. Farahat, C. Gupta, Long short-term memory network for remaining useful life estimation, in: 2017 IEEE International Conference on Prognostics and Health Management (ICPHM), 2017, pp. 88–95.
URL https://doi.org/10.1109/ICPHM.2017.7998311

[6] N. Gugulothu, T. Vishnu, P. Malhotra, L. Vig, P. Agarwal, G. M. Shroff, Predicting remaining useful life using time series embeddings based on recurrent neural networks, CoRR abs/1709.01073.
URL https://arxiv.org/abs/1709.01073

[7] X. Li, Q. Ding, J.-Q. Sun, Remaining useful life estimation in prognostics using deep convolution neural networks, Reliability Engineering & System Safety 172, 2018, pp. 1–11.
URL https://doi.org/10.1016/j.ress.2017.11.021

[8] Y. Roh, G. Heo, S. E. Whang, A survey on data collection for machine learning: a big data - AI integration perspective, CoRR abs/1811.03402.
URL https://arxiv.org/abs/1811.03402

[9] E.-J. Wagenmakers, M. Lee, T. Lodewyckx, G. J. Iverson, Bayesian versus frequentist inference, Springer New York, New York, NY, 2008, pp. 181–207.
URL https://doi.org/10.1007/978-0-387-09612-4_9

[10] C. Blundell, J. Cornebise, K. Kavukcuoglu, D. Wierstra, Weight uncertainty in neural networks, in: Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37, ICML'15, JMLR.org, 2015, pp. 1613–1622.
URL https://arxiv.org/abs/1505.05424

[11] Y. Gal, Z. Ghahramani, Bayesian convolutional neural networks with Bernoulli approximate variational inference, in: 4th International Conference on Learning Representations (ICLR) workshop track, 2016.
URL https://arxiv.org/abs/1506.02158

[12] M. Fortunato, C. Blundell, O. Vinyals, Bayesian recurrent neural networks, CoRR abs/1704.02798.
URL https://arxiv.org/abs/1704.02798

[13] K. Shridhar, F. Laumann, M. Liwicki, A comprehensive guide to Bayesian convolutional neural network with variational inference, CoRR abs/1901.02731.
URL https://arxiv.org/abs/1901.02731

[14] A. Kendall, Y. Gal, What uncertainties do we need in Bayesian deep learning for computer vision?, CoRR abs/1703.04977.
URL https://arxiv.org/abs/1703.04977

[15] Z. Tian, An artificial neural network approach for remaining useful life prediction of equipments subject to condition monitoring, in: 2009 8th International Conference on Reliability, Maintainability and Safety, 2009, pp. 143–148.
URL https://doi.org/10.1109/ICRMS.2009.5270220

[16] C.-D. Lai, D. Murthy, M. Xie, Weibull distributions and their applications, Springer London, London, 2006, pp. 63–78.
URL https://doi.org/10.1007/978-1-84628-288-1_3

[17] S. Wu, N. Gebraeel, M. A. Lawley, Y. Yih, A neural network integrated decision support system for condition-based optimal predictive maintenance policy, IEEE Transactions on Systems, Man, and Cybernetics -

Part A: Systems and Humans 37, 2007, pp. 226–236.
URL https://doi.org/10.1109/TSMCA.2006.886368

[18] [dataset] A. Saxena, K. Goebel, PHM08 challenge data set, in: NASA Ames Prognostics Data Repository, NASA Ames Research Center, Moffett Field, CA, 2008.
URL https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/

[19] [dataset] A. Saxena, K. Goebel, Turbofan engine degradation simulation data set, in: NASA Ames Prognostics Data Repository, NASA Ames Research Center, Moffett Field, CA, 2008.
URL https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/

[20] A. Saxena, K. Goebel, D. Simon, N. Eklund, Damage propagation modeling for aircraft engine run-to-failure simulation, in: 2008 International Conference on Prognostics and Health Management, 2008, pp. 1–9.
URL https://doi.org/10.1109/PHM.2008.4711414

[21] [dataset] A. Agogino, K. Goebel, Milling data set, in: NASA Ames Prognostics Data Repository, NASA Ames Research Center, Moffett Field, CA, 2007.
URL https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/

[22] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, Journal of Machine Learning Research 15, 2014, pp. 1929–1958.
URL http://jmlr.org/papers/v15/srivastava14a.html

[23] D. P. Kingma, J. Ba, Adam: a method for stochastic optimization, in: 3rd International Conference on Learning Representations, 2015.
URL https://arxiv.org/abs/1412.6980

[24] L. Jayasinghe, T. Samarasinghe, C. Yuen, S. S. Ge, Temporal convolutional memory networks for remaining useful life estimation of industrial machinery, ArXiv abs/1810.05644.
URL https://arxiv.org/abs/1810.05644

[25] A. S. Yoon, T. Lee, Y. Lim, D. Jung, P. Kang, D. Kim, K. Park, Y. Choi, Semi-supervised learning with deep generative models for asset failure prediction, CoRR abs/1709.00845.
URL https://arxiv.org/abs/1709.00845

[26] D. P. Kingma, T. Salimans, M. Welling, Variational dropout and the local reparameterization trick, in: Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2, NIPS'15, MIT Press, Cambridge, MA, USA, 2015, pp. 2575–2583.
URL https://arxiv.org/abs/1506.02557

[27] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, Journal of Machine Learning Research - Proceedings Track 9, 2010, pp. 249–256.
URL https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.207.2059

[28] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic differentiation in PyTorch, in: NIPS Autodiff Workshop, 2017.
URL https://openreview.net/forum?id=BJJsrmfCZ

[29] C. R. Qi, L. Yi, H. Su, L. J. Guibas, PointNet++: deep hierarchical feature learning on point sets in a metric space, CoRR abs/1706.02413.
URL https://arxiv.org/abs/1706.02413

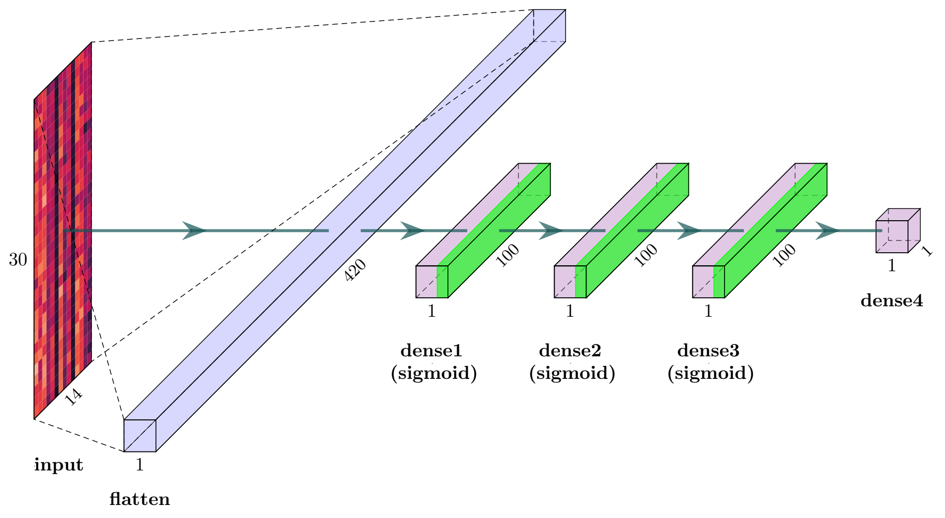
[30] S. G. K. Patro, K. K. Sahu, Normalization: a preprocessing stage, CoRR abs/1503.06462.
URL https://arxiv.org/abs/1503.06462

[31] F. O. Heimes, Recurrent neural networks for remaining useful life estimation, in: 2008 International Conference on Prognostics and Health Management, 2008, pp. 1–6.
URL https://doi.org/10.1109/PHM.2008.4711422

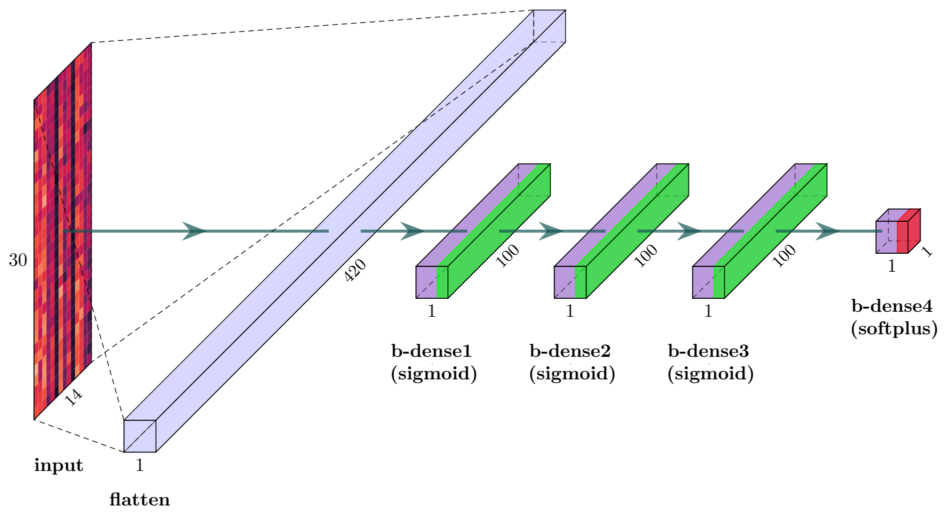
[32] J. Postels, F. Ferroni, H. Coskun, N. Navab, F. Tombari, Sampling-free epistemic uncertainty estimation using approximated variance propagation, International Conference on Computer Vision (ICCV).
URL https://arxiv.org/abs/1908.00598

[33] Y. Kwon, J.-H. Won, B. Joon Kim, M. Paik, Uncertainty quantification using Bayesian neural networks in classification: application to ischemic stroke lesion segmentation.
URL https://openreview.net/forum?id=Sk_P2Q9sG

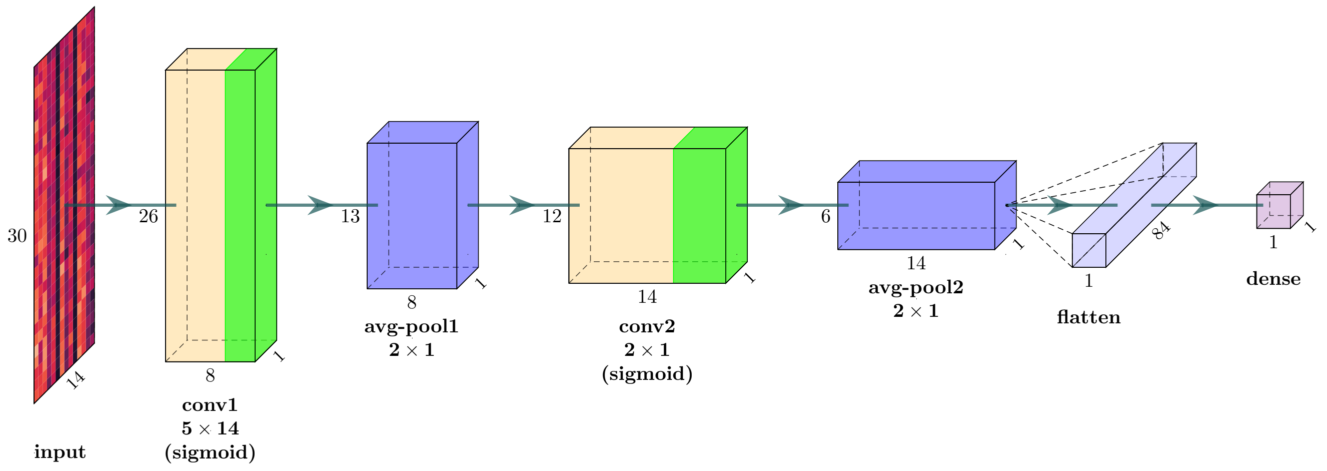
Appendix A. Neural network architectures of the proposed regression models



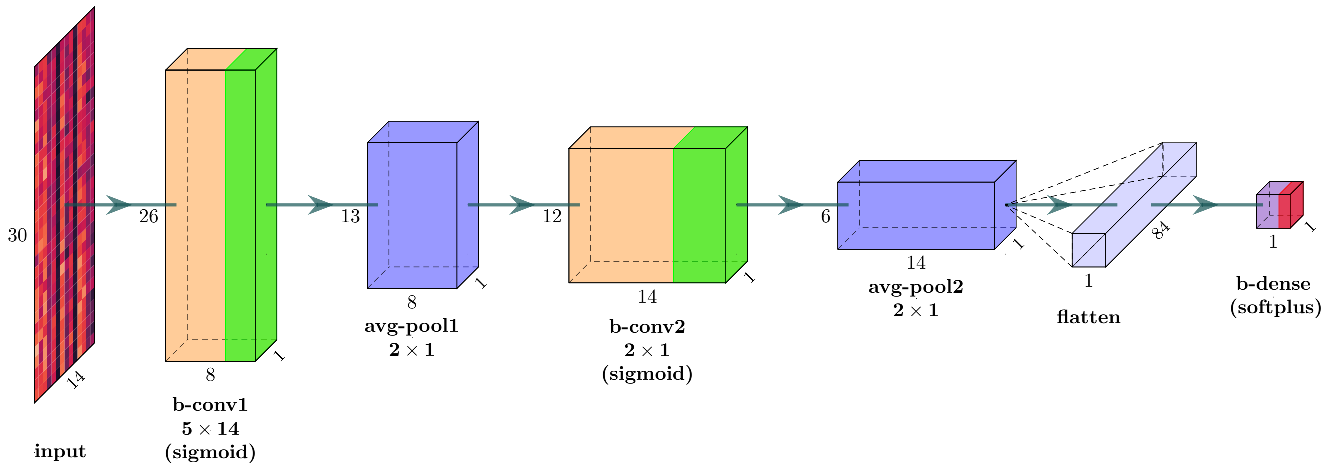
(6a) F-D3 regression model with frequentist dense layers.



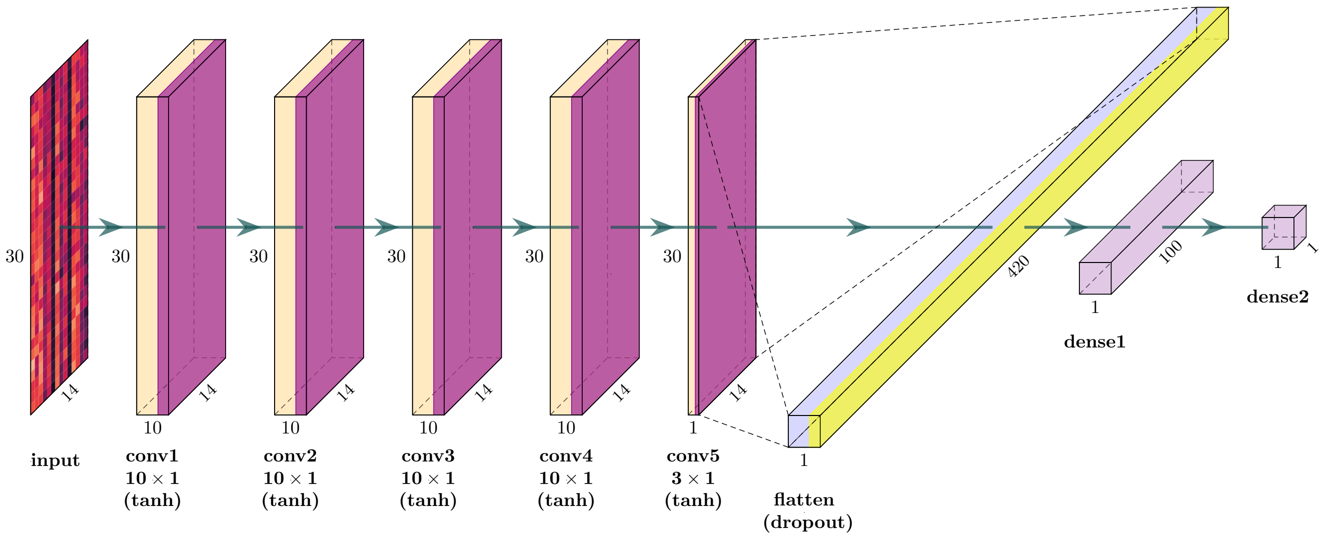
(6b) B-D3 regression model with Bayesian dense layers.



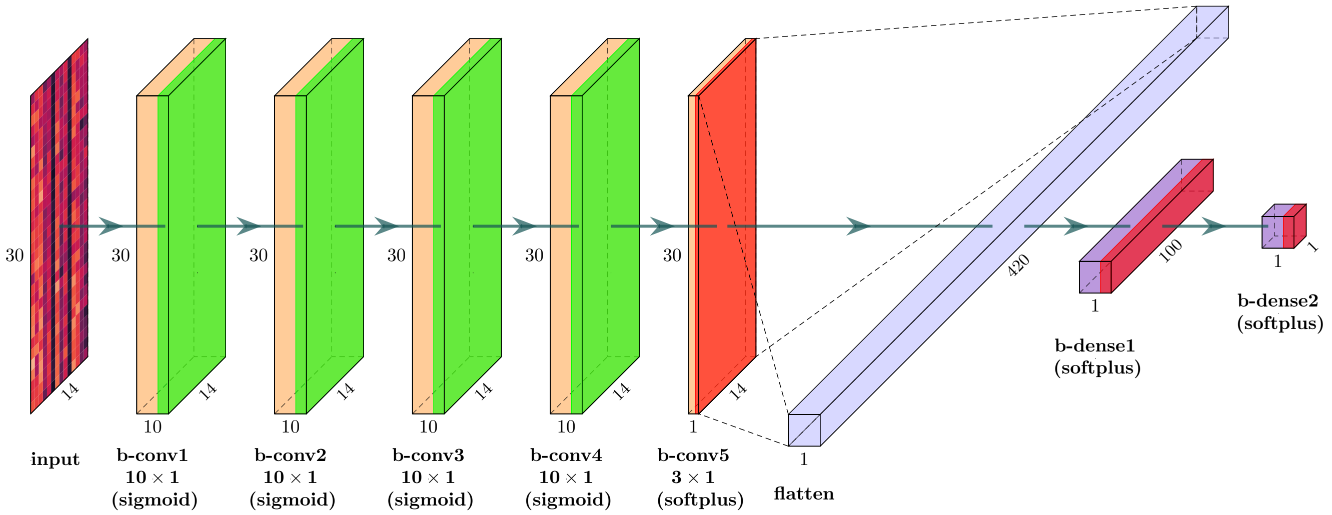
(6c) F-C2P2 regression model with frequentist convolutional and dense layers.



(6d) B-C2P2 regression model with Bayesian convolutional and dense layers.



(6e) F-C5D1 regression model with frequentist convolutional and dense layers.



(6f) B-C5D1 regression model with Bayesian convolutional and dense layers.