Remaining Useful Life Estimation Using Neural Ordinary Differential Equations

Marco Star[1], Kristoffer McKee[2]

[1,2] *Curtin University of Technology, Perth, Western Australia, 6845, Australia*
marco.star@postgrad.curtin.edu.au
K.Mckee@curtin.edu.au

ABSTRACT

Data-driven machinery prognostics has seen increasing popularity recently, especially with the effectiveness of deep learning methods growing. However, deep learning methods lack useful properties such as the lack of uncertainty quantification of their outputs and have a black-box nature. Neural ordinary differential equations (NODEs) use neural networks to define differential equations that propagate data from the inputs to the outputs. They can be seen as a continuous generalization of a popular network architecture used for image recognition known as the Residual Network (ResNet). This paper compares the performance of each network for machinery prognostics tasks to show the validity of Neural ODEs in machinery prognostics. The comparison is done using NASA's Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) dataset, which simulates the sensor information of degrading turbofan engines. To compare both architectures, they are set up as convolutional neural networks and the sensors are transformed to the time-frequency domain through the short-time Fourier transform (STFT). The spectrograms from the STFT are the input images to the networks and the output is the estimated RUL; hence, the task is turned into an image recognition task. The results found NODEs can compete with state-of-the-art machinery prognostics methods. While it does not beat the state-of-the-art method, it is close enough that it could warrant further research into using NODEs. The potential benefits of using NODEs instead of other network architectures are also discussed in this work.

1. INTRODUCTION

Machinery prognostics is defined as the process used to estimate the remaining useful life (RUL) of machinery or its components. To estimate the RUL, data is needed that indicates the machinery's health and a prognostics method is needed which will extrapolate and find the RUL based on the current health. There are three main categories of machinery prognostics methods, physics-based, data-driven and hybrid. For this work data-driven methods are employed as they have become increasingly popular as more data is gathered on machinery using sensors (Zhao et al., 2019). Data-driven models optimize their parameters based on the historical data inputs to estimate the RUL, which is referred to as training the model.

Machine learning techniques have become increasingly used in data-driven machinery prognostics. Implementation of these techniques generally involves pre-processing the historical data, extracting features from the data that correlate well with the RUL and training the machine learning model on the data and features. However, deep learning methods do not require feature extraction as the deep neural networks can automatically extract features from raw data (W. Zhang, Yang, & Wang, 2019).

The lack of a feature extraction step is not the only benefit of using deep learning methods in machinery prognostics; they perform well compared to other data-driven methods and require no expert knowledge of the system. For example, recurrent neural networks (RNNs) are well suited for extracting features from time-series signals, such as sensor signals, and have been shown to achieve state-of-the-art performance in this task. In machinery prognostics, RNNs have been used to directly model and forecast the degradation of equipment (Y. Zhang, Xiong, He, & Pecht, 2018; Zhou, Huang, Pang, & Wang, 2019), or extract features directly from time-series inputs and output a RUL estimate (Wu et al., 2019; Huang, Huang, & Li, 2019; Listou Ellefsen, Bjørlykhaug, Æsøy, Ushakov, & Zhang, 2019). While RNNs are well suited for time-series problems other deep neural networks have also been utilized for estimating the RUL.

The convolutional neural network (CNN) is a network which is normally used in image recognition and does not need to do computations sequentially like the RNN. Due to the CNNs high performance in image analysis and recognition tasks it is common to frame the machinery prognostics task as an im-

https://doi.org/10.36001/IJPHM.2021.v12i2.2938

age recognition task. This is usually done by applying a time-frequency transform on the sensor signals and creating a spectrogram from the output coefficients and using that as the image with a corresponding RUL as a target value. For example, (Zhu, Chen, & Peng, 2019) used the wavelet transform on the input sensor data to transform it into this time-frequency format and then used a CNN to estimate the RUL. Other transforms have also been used such as, short-time Fourier transforms (STFT) (Li, Zhang, & Ding, 2019), Empirical Mode Decomposition (EMD) (Yang, Yao, Ye, & Xu, 2020), regular Fourier transform (Wang, Zhao, Ma, Chang, & Mao, 2019) and simply time-windowed raw data (Li, Ding, & Sun, 2018). All these methods applied the CNN to the transformed data to output a RUL.

Deep learning methods can estimate the RUL to a high accuracy, however, the methods mentioned so far did not focus on interpretability of the results. These methods can be a black box where the inputs are fed into the network and an output is given that can be accurate assuming the correct data was used as an input. Most deep learning methods in machinery prognostics only provide point estimates without incorporating uncertainty into the RUL estimate (Peng, Ye, & Chen, 2019). The uncertainty quantification is important in machinery prognostics for decision making due to the influence different sources of uncertainty have on the RUL prediction (Sankararaman, 2015). Some techniques have however been used to incorporate uncertainty into RUL estimates using machine learning or deep learning methods.

When using a measurable variable which somehow indicates the component's health, a Bayesian filter, such as the Kalman or Particle filter, can be directly applied to update the variable based on newly observed measurements. Hence, a machine learning model may be used to model the health indicator and extrapolate to some threshold value indicating the end of its useful life while the Bayesian filter updates the probability distribution given new measurements. For example, machine learning models such as the Relevance Vector Machine (RVM) can be used to mathematically model the battery capacity degradation while using the particle filter to update the RUL estimates based on new measurements (Saha, Goebel, Poll, & Christophersen, 2009; Hu & Luo, 2013). (Guo, Li, Jia, Lei, & Lin, 2017) did not have a measurable health indictor but constructed one with a known threshold using bearing vibration data as an input to a RNN whose output was a one-dimensional health indicator variable. This health indicator had a known threshold and a simple exponential model was fit to the health indicator and used to extrapolate to the threshold value. The exponential model parameters could be described using normal distributions with more ease than a deep neural network, which has considerably more parameters. RNNs have also been altered to model physics-based differential equations which describe health indictors such as crack length and modelling grease degradation (Yucesan & Viana, 2019; Dourado & Viana, 2019; Nascimento & Viana, 2019). These can similarly include uncertainty in their parameters due to the relative simplicity of the final degradation model. Deep neural networks can incorporate uncertainty into their structure and these are known as Bayesian neural networks. (Peng et al., 2019) went through how to alter various state-of-the-art deep neural networks into Bayesian neural networks. However, most popular Bayesian networks rely on either increasing the amount of parameters in the network (Blundell, Cornebise, Kavukcuoglu, & Wierstra, 2015) or by utilizing Monte Carlo simulations when evaluating the network (Gal & Ghahramani, 2016) making them slower than a standard deep neural network. Most of these methods that account for uncertainty either,

- Use Monte Carlo simulation thereby increasing the computational cost
- Require a physics-based mathematical description of the degradation mechanics
- Require the output to be a measurable or known physics-based health indicator

Hence, it would be useful to investigate avenues that allow for deep learning methods to account for uncertainty without these drawbacks.

Recently a type of neural network architecture known as Neural Ordinary Differential Equations (NODEs) has been introduced (Chen, Rubanova, Bettencourt, & Duvenaud, 2018). NODEs essentially describes the input to output variable transformation by a trajectory through a vector field i.e. the networks hidden variable dynamics. The vector field is defined by a neural network and the trajectory is solved using numerical ODE solver schemes such as Euler's method or Runge-Kutta methods. The investigation of these vector fields which define the hidden variable dynamics, could be a research avenue that has the potential to improve interpretability of the network or quantify uncertainty (e.g. stochastic dynamics are used instead). Existing knowledge on differential equations could open interesting research avenues to alter the NODEs architecture to give meaningful RUL estimates and reduce the black-box nature of the network. The aim of this paper is to show the applicability of NODEs in machinery prognostics by applying them to the popular C-MAPSS turbofan engine dataset and comparing their performance to other methods. This is largely to show how NODEs hold up to state of the art machinery prognostics methods. Hence, in this study NODEs will function like most black-box neural networks without a focus on its dynamics to keep the experiment simple. The idea is to first investigate if NODEs can perform well on a RUL estimation problem given degradation data. If it performs well this may increase incentive and discussion of how to leverage their properties to benefit machinery prognostics tasks. Section 5 provides a discussion on the useful properties NODEs have and how they might be leveraged for machinery

prognostics given further research. Another network known as the Residual Network (ResNet) (He, Zhang, Ren, & Sun, 2016), a type of CNN popular in image recognition tasks, will also be included in the experiments. This is mainly due to the fact NODEs were introduced as a continuous version of the ResNet so they are expected to perform similarly. The NODE-based network will be a CNN so it is easily comparable with a ResNet and the input sensor data will be transformed to spectrograms using a short-time Fourier transform (STFT). Discussion of the results, including possible research directions for utilizing or extending NODEs for machinery prognostics applications, are given after the results of the experiments.

2. BACKGROUND

2.1. Short-Time Fourier Transform

The short-time Fourier transform (STFT) is a way of analyzing a non-stationary signal by approximating discrete parts of the signal as stationary. This is done through a sliding window that acts over the signal and applies the Fourier transform on the windowed signal to extract frequency information at that time. Mathematically this can be stated as Eq. (1),

$$X(\omega, t) = \int_{-\infty}^{\infty} f(t) w(t-\tau) e^{-j\omega t} dt. \tag{1}$$

Where $f(t)$ is the signal, $w(t-\tau)$ is the window function, $X(\omega, t)$ is the resulting time-frequency signal.

The STFT was used for simplicity and does not require the selection of a basis function as is the case with the Wavelet transform. Different basis function selections can influence the results significantly; this influence was avoided by using the STFT instead of the Wavelet transform (Li et al., 2019).

2.2. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) (Yann Lecun, 1989) are a feedforward neural network which use filters to extract features and relate data points which are spatially close on the grid. The filter is made up of learnable parameters called weights which are optimized during the training of the network. The main component of the CNN is the convolutional layer; this layer performs the convolution operation between the filter weights and the layer input data.

In this case, the inputs are a grid of intensity values found from the time-frequency transform. Hence, the example of the convolution layer will involve a 2D grid of input values. The filter in the 2D case is a matrix of weight parameters that act on the input values through the convolution operator. In this 2D case the convolution operation will generate output values that are elements of an output matrix whose size is dependant on hyperparameters such as filter size and stride (amount of spaces the filter "slides" over). The convolution operator in this case will involve the filter "sliding" over the input values in the grid and multiplying the filter values with the inputs element-wise and summing these to a single value (Dumoulin & Visin, 2016). The convolution layer operations are showcased in Figure 1.

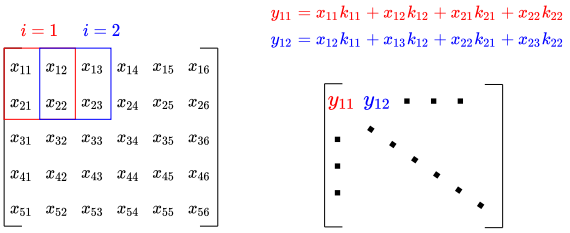


Figure 1. A 2×2 CNN filter sliding along input data (x), performing convolution to produce an output matrix, "i" denotes the position and order the filter is sliding in i.e. the filter starts at i=1 and goes to i=2 etc. The values k represent the values in the filter matrix. y are the outputs of the convolution operation and make up the elements of the CNN output matrix

The parameters in the filter are the trainable parameters which are being optimized to produce the desired output given the inputs. The CNN uses multiple filters and therefore give multiple outputs which are often flattened to a vector and becomes the input to standard feedforward neural network which produces an output of the desired size. Notice that the CNN essentially trains different filters which all give weighted averages of the local area of the input data. In this work, the local area represents the time and frequency coordinates while the values themselves represent the intensity of the frequencies at that time which indicates the energy of the signal.

2.3. ResNet

The Residual Network (ResNet) (He et al., 2016), is a convolutional neural network which learns the residual or difference in the hidden variables. The equation for the ResNet is given by Eq. (2),

$$x_{i+1} = x_i + f_i(x_i, \theta_i). \tag{2}$$

Where x_i is the variable at the ResNet network layer indexed by $i = 1, ..., n$, f_i is a neural network used by the ResNet at layer i and θ_i is the network parameters at layer i.

Equation 2 describes a ResNet block which can be illustrated using the diagram shown in Figure 2.

The addition of the inputs x_i to the neural network output $f(x_i, \theta)$ results in the ResNet block having a "skip connection". Therefore, the inputs (x_i) as well as the network outputs ($f_i(x_i, \theta_i)$) influence the final output of the ResNet block (x_{i+1}).

Notice that the network f_i describes the difference or residual

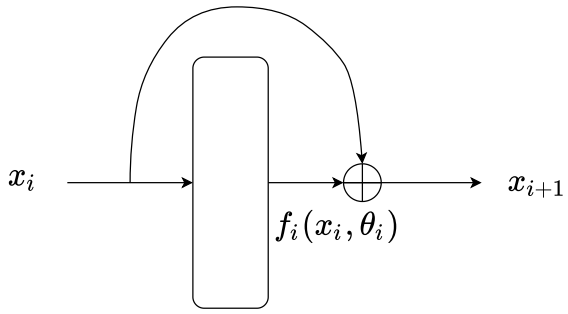


Figure 2. Schematic showing how inputs transform to outputs using the neural network $f(x_i, \theta)$ and a skip connection. Where x_i is the input data and θ are the network parameters. Note that the rectangular block represents the neural network $\bigoplus$ represents element-wise addition and x_{i+1} are the outputs.

between the two hidden variables i.e. $x_{i+1} - x_i$, hence the name ResNet. This formulation was first applied to CNNs (f_i was a CNN) and resulted in improved performance on image recognition tasks (He et al., 2016).

2.4. NODE

Neural Ordinary Differential Equations (NODEs) (Chen et al., 2018) were first introduced as a continuous version of the ResNet. The equation describing NODEs is given by Eq. (3),

$$x_{t+1} = x_t + f(x_t, t, \theta). \tag{3}$$

Where, x_t is the variable at the layer indexed by $t = 1, ..., n$, f is the neural network describing the ODE i.e. $\frac{dx_t}{dt}$ and θ represents the network parameters.

Figure 3 shows a simple illustrated example of how NODEs take input variables and transform them to an output variable.

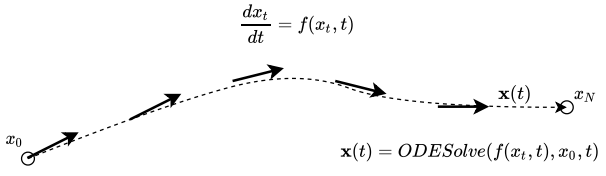


Figure 3. A visual example of how NODE propagates the input variable x_0 through the network, which is described by an ODE. Each solid arrow is an evaluation of the network $\frac{dx_t}{dt} = f(x_t, t)$ which is used to find the next step or next network hidden layer variable, where t is the depth of the network. Since, there are 5 arrows shown this is equivalent to a network with 5 hidden layers. The dotted path labelled $\mathbf{x}(t)$ is the trajectory the variables take through the NODE and x_N is the final output of the NODE.

Here, the network f is constant throughout each layer or evaluation while the ResNet had a different network block for each layer f_i. This is the major difference between NODEs and other neural networks, NODE uses a neural network to describe a larger "network". The inputs are transformed to the network output variables using a numerical differential equation solver and the network f acts as the function describing the first order ODE $\frac{dx}{dt}$. In this case the input variables are propagated through the vector field described by the neural network f. By choosing the number of steps (or step-size) in the ODE solver the amount of "layers" or functional evaluations are controlled. Since the parameters of the neural network θ do not change as the variables are propagated through the vector field; the network will be denoted as $f(x_t, t)$ instead of $f(x_t, t, \theta)$ for simplicity.

3. DATA

To compare the ResNet and NODEs they will both be trained on a prognostics dataset to estimate the RUL of machinery. This section will explain the contents of the dataset, additionally it covers how the data was prepared to become the inputs of the neural networks, as well as how the RUL targets were prepared for the supervised learning task.

3.1. C-MAPSS dataset

The data used here is NASA's Commercial modular aero-propulsion system simulation (C-MAPSS) dataset (Saxena & Goebel, n.d.). This dataset was produced through simulations of sensor readings for aircraft gas turbine engines. The RUL values and therefore the time of failure is known for each simulated engine and was found when the simulated variables reached a certain threshold determined by a failure criteria. There are four separate datasets which vary in difficulty with respect to RUL estimation, a summary of each dataset is given in Table 1. The possible failure modes that could occur in this dataset set are either the high-pressure compressor (HPC) or fan degrading below a certain threshold. Each dataset includes training and testing trajectories (time-series data) which contain the sensor values and operating conditions of a number of different units/engines. The training set is simulated until failure, hence, the final time point is the time of failure (i.e. RUL = 0 at the final time). The testing dataset contains trajectories until a random time point and the RUL at that time point is supplied for each trajectory. The trained network can be evaluated on this testing set and the estimated RUL can be compared with the true RUL value given in the testing dataset.

Each dataset consists of multiple time series for both the training and testing datasets. There are 26 columns of data for each unit. The contents of each column are specified in Table 2.

Using the CMAPSS dataset the two CNNs, both the ResNet and NODEs (now referred to as NODE-CNN) architectures will be trained using the training dataset and have their per-

Table 1. Different datasets found in CMAPSS: these state how many different engines are simulated, the number of operating conditions and the number of possible failure conditions they can experience (Fault Modes)

Dataset	Train Trajectories	Test Trajectories	Operating Conditions	Fault Modes
FD001	100	100	1	1
FD002	260	259	6	1
FD003	100	100	1	2
FD004	248	249	6	2

Table 2. Information contained in each column for the training and testing datasets

Columns	Data Contained
1	unit number
2	time (cycles)
3	operational setting 1
4	operational setting 2
5	operational setting 3
6-26	sensor measurements (1-21)

formance compared on the testing dataset. Note that, NODE-CNN will have a CNN architecture simply for ease of comparison to the ResNet. Specifics on evaluating the performance of the network on the test data will be mentioned in section 5 (Results and Discussion).

3.2. Preparing Data

For the prognostics algorithm used in this study, the data must be prepared to be suitable as an input of the CNNs used (both the ResNet and NODE-CNN). Here suitability means that the CNN can easily extract the relevant features from the data and correlate the features to the RUL. For example, the time-series signals can be converted to a 2D time-frequency spectral plot using time-frequency methods such as wavelet transforms or short-time Fourier transforms (Zhu et al., 2019; Li et al., 2019). This would convert the time-series data into an image format from which the CNN can extract features. For this experiment a similar process to (Zhu et al., 2019) was used which involved the following general steps,

1. Relevant sensor signals are taken from the machinery sensor data and normalized with Eq. (5).
2. A short-time Fourier transform is applied to each of the different time-series sensor signals (i.e. output tensor size = [amount of sensors, time, frequency]).
3. Bilinear interpolation is used to change the size of each time-frequency signal to a 30×30 format (i.e. output tensor size = [amount of sensors, 30, 30]).

A general diagram of the data transformation is shown in Figure 4.

Firstly, it is important to drop sensors whose values do not change (or change very little) during the life of the machine,

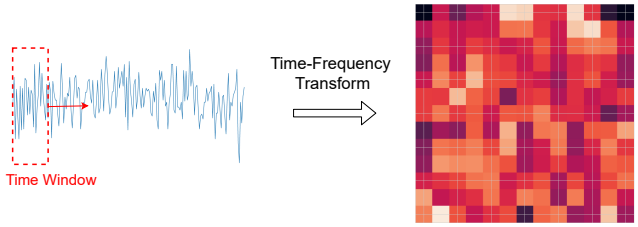


Figure 4. A single time-series signal which is transformed to the time-frequency domain using the STFT. The frequency intensities from the Fourier coefficients is shown in the image that results from the time-frequency transform

as they do not capture useful degradation information of the machine for estimating the RUL. To drop the relevant sensor signals from the dataset a small threshold standard deviation was chosen (e.g. 1×10^{-8}). Any sensor signals whose standard deviation was below this threshold were not used. Some sensor signals were constant throughout but had large outliers; these standard deviations passed the threshold test. It was found these signals did not help performance and therefore another criteria was added to ensure these sensors were also dropped. This criteria simply checked how many unique values were in the sensor signal, if there were less unique values than a threshold value (less than 3 unique values in this case) it was dropped. This procedure identified a group of sensor signals which were better suited for the machinery prognostics task. The sensors which were dropped are listed in Table 3.

Table 3. Sensors dropped from the total of 21 sensors

Sensor Number
1
5
16
18
19

It is common in prognostics to normalize the time series data so all sensor signals are similar in magnitude. This is often done by taking the means and standard deviations of the sensor signals for each unit and applying Eq. (4),

$$\hat{\mathbf{x}}_d^{(n)} = \frac{\mathbf{x}_d^{(n)} - \mu_d^{(n)}}{\sigma_d^{(n)}}. \tag{4}$$

Where, $\hat{x}_d^{(n)}$ are the normalized sensor values for unit number n and sensor d. Similarly, $\mu_d^{(n)}$ denotes the means of each sensor, $\sigma_d^{(n)}$ denotes the standard deviations and $x_d^{(n)}$ are the raw sensor values.

Normalization is often carried out this way, by calculating the mean and standard deviation based on the sensor data for a particular unit. However, with multiple operating conditions

such as in the datasets FD002 and FD004 (see Table 1), (Pasa, Medeiros, & Yoneyama, 2019) showed it is often more useful to normalize the sensor data based on a particular operating condition. Hence, for these datasets the normalization of the data can done by applying Eq. (5) (Pasa et al., 2019),

$$\hat{\mathbf{x}}_d^{(n)} = \sum_{c=0}^{N_c} \delta_c \odot \frac{\mathbf{x}_d^{(n)} - \mu_d^{(c)}}{\sigma_d^{(c)}}. \tag{5}$$

In this case, $\mu_d^{(c)}$ and $\sigma_d^{(c)}$ are the mean and standard deviation of sensor signal d given it is in operating condition c, N_c denotes the total number of operating conditions (e.g. 6 for FD002 and FD004), $\odot$ is the element-wise multiplication operator and $\delta_c = 1$ when the data at that time point is in the operating condition c otherwise $\delta_c = 0$. The different operating conditions can be found through different combinations of the three operational setting variables. For FD001 and FD003 the settings have little change during operation and hence, only have one operating condition. FD002 and FD004 have 6 distinct clusters/combinations the operational settings could fall under, hence, they have 6 operating conditions. A clustering algorithm can be used to classify the sensor values at any time into one of these operating conditions. In this case the K-means clustering algorithm was used for simplicity. The comparison between using the operating condition normalization (Eq. (5)) versus the standard signal-based normalization (Eq. (4)), applied on the same unit is shown in Figures 5 and 6.

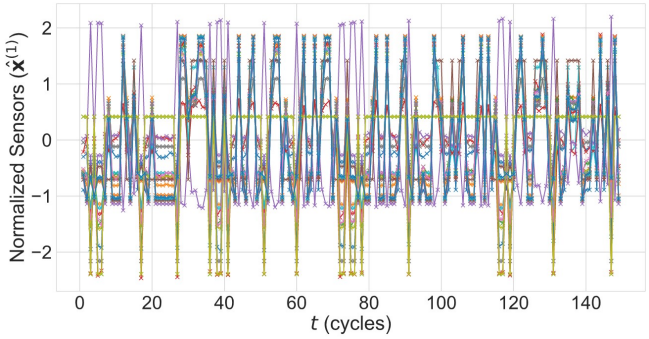


Figure 5. Normalization based on signal mean and standard deviation applied on unit 1's sensor values in the training dataset

Secondly, a short-time Fourier transform is performed on these signals to achieve a time-frequency representation i.e. a plot of intensities (based on Fourier coefficients) along the time-frequency axes. The transform is applied to the signal starting from the initial time to the current time. These time-frequency output intensities are stored in a matrix whose size based on the time window size chosen. Each of the signals also have varying length and hence to store batches of input data and have consistent output sizes for the CNN, the inputs should have the same size. To achieve this bilinear interpolation will be used to convert the time-frequency signals to a 30×30 size "image-like" format.

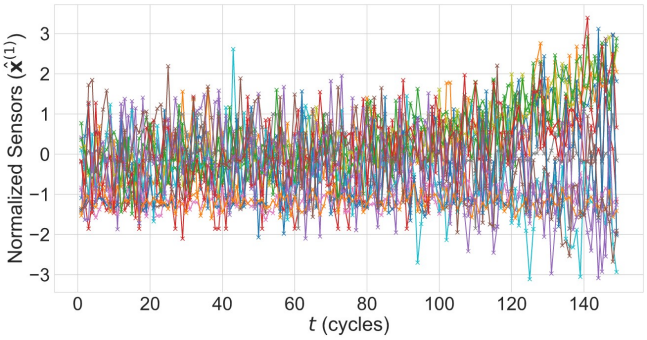


Figure 6. Normalization based on operating condition mean and standard deviation applied on unit 1's sensor values in the training dataset

The testing and training data preparation differed in one major aspect; the testing signals were converted to spectrograms using the entire signal while the training signals were broken up into smaller signals then converted. The reason for this is that the testing signals have a corresponding RUL value and are not at their end of life, while all the training signals would have a target of $RUL = 0$ if the entire signal was used. Hence, the training signals were split into multiple signals based on a hyperparameter called $split$ which determined the proportion of the signal to converted into a spectrogram and used as an input. This split hyperparameter calculated the proportion of the signal to be used through Eq. (6),

$$p_i = \frac{i}{split + 1}, \text{ for } i = 1, ..., split. \tag{6}$$

Here, p_i is a multiplier that determines the proportion of the signal used. For example if L is the total length of the signal then $L_i = L \times p_i$ would be the length of the cut signal used to create a spectrogram. Note that $split$ cuts the training signal into a number of smaller signals through the different values of $i = 1, ..., split$. The denominator of the equation is also $split + 1$ so that the entire signal is never used as the entire signal has a corresponding RUL value of zero which is not useful.

Finally the target RULs were altered to incorporate a maximum RUL value. The maximum RUL signifies that the net-

work cannot estimate the RUL beyond this point as the input data is not in the degradation stage. This was noticed when training Multilayer Perceptron Networks (MLP) on the C-MAPSS dataset as the networks early RUL estimates would stay relatively constant up to a point then start to decline (Heimes, 2008). Hence, here the target RULs all had a maximum value of $r_{max} = 130$ cycles which also allows for more direct comparison to other prognostics methods which apply the same r_{max}. By setting a maximum RUL value not only is the predictive performance increased, it also signifies the RUL value where the network cannot make a reasonable RUL prediction for higher values. The point at which the RUL starts to decline from r_{max} also signifies the start of degradation as the network now has data which it can use to predict the RUL with more confidence.

4. METHODOLOGY

This section states the details of the experiments performed on the data using the different network architectures. The general process used to estimate the RUL will be stated as well as the specifics of the hyperparameters used to construct the network. Both the ResNet and NODE-CNNs have similar processes for estimating the RUL, therefore, a general overview is given below which is relevant to both architectures. The general steps involved with transforming the input data into an output RUL estimate are as follows,

1. A separate CNN is used to downsample (reduce the dimensions of) the prepared input data described in the previous section.
2. This step applies either the ResNet or NODE to extract features from the downsampled input data.
3. The final output from the ResNet/NODE is flattened to a vector and the input to a standard feedforward MLP network. The output of this MLP is the RUL estimate of the machinery.

Bayesian optimization tools were used to optimize the hyperparameters. The software package Ax which uses the package BoTorch as a backend (Balandat et al., 2019) was used to tune the hyperparameters using Bayesian optimization. Bayesian optimization uses a curve-fitting method known as Gaussian process regression to optimize the hyperparameters based on some criterion (Frazier, 2018). It takes a vector of hyperparameters as inputs and outputs some criterion or loss value. In the deep learning case the hyperparameters are used to train the neural network; after training the criterion can be calculated and this represents the point on a curve. The goal of Bayesian optimization is to find the vector of input hyperparameters that minimizes the criterion which is the equivalent of finding the minimum of the curve. Each new criterion calculated for a set of input hyperparameters is treated as a data point describing the curve and Bayesian methods are used to update the curve based on this new data.

The criterion used for hyperparameter optimization in this work is the normalized score function (Eq. (7)). The score function was designed to result in higher loss values for RUL estimates that are larger than the actual RUL (Saxena, Goebel, Simon, & Eklund, 2008). This is done to encourage more conservative estimates so that failures do not occur before the prediction from the estimated RUL.

$$s = \begin{cases} \frac{1}{N}\sum_{i=1}^{N} \exp\left[\frac{-(\hat{y}_i - y_i)}{13}\right] - 1, & \text{if } \hat{y}_i \leq y_i \\ \frac{1}{N}\sum_{i=1}^{N} \exp\left[\frac{(\hat{y}_i - y_i)}{10}\right] - 1, & \text{if } \hat{y}_i > y_i \end{cases} \tag{7}$$

Where $\hat{y}_i$ is the estimated RUL of unit i from the network, y_i is the actual RUL of unit i and N is the total amount of units in the dataset. Notice here an average score is used (referred to as normalized score), hence it is the score found in (Saxena et al., 2008) divided by the total amount of units N.

The criterion is evaluated on a validation set, separate from the dataset the network is trained on when tuning the hyperparameters. Here the validation set was acquired from the original training data which is further split into a training dataset and a validation dataset. An 80%/20% (training/validation) split was used on the original training data in this particular case. The new training set is used to optimize the network during the hyperparameter optimization while the validation set acts as a testing dataset that the network did not see during training. The validation loss therefore represents a testing loss that can be used to optimize the hyperparameters. Note that the testing dataset is left alone and untouched by any optimization process to avoid overfitting.

This hyperparameter optimization method was chosen as it is more computationally efficient than other optimization methods such as random search which randomly tests different hyperparameter combinations. Bayesian optimization reduces the amount of time required to find well performing hyperparameters compared to other methods which may find more optimal hyperparameters but require a longer time to search for them. The hyperparameters which are being tuned are,

- Learning rate
- Batch size
- Time window for STFT
- split (see Eq. (6))
- Weight decay (L2 regularization on the network parameters)
- Training epochs

Once the hyperparameters are chosen the networks could be trained on the full training dataset. The network is trained using the Mean Squared Error loss function (Eq. (8)).

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2. \quad (8)$$

4.1. ResNet

For the ResNet experiment the time-frequency data was the input to an initial set of downsampling convolutional layers which extracted features and reduced the dimensionality of the data. The output of the downsampling layers became the inputs to the first ResNet block, while the outputs of one ResNet block were the inputs of the next. The output of the final ResNet block was flattened and passed through a feedforward network which would output the RUL estimate. Table 4 shows the basic layout of the ResNet architecture used as well as the size of the output tensors of each layer. Note that the network sizes were simply chosen based on other examples of CNNs used in machinery prognostics tasks and further optimization could be done if necessary.

Table 4. Architecture of the ResNet. Note that bs refers to batch size of the training data and channels are the number of filters each CNN uses. The last part of the table specifies the ResNet Block used in the architecture and uses generic dimension (dim) keywords instead of numbers to define a generic ResNet Block. Also note the ResNet block in the downsampling layers lowers the dimensions by setting the stride of the first Conv2D layer in the ResNet block to the appropriate value.

Layers	**Layer Output Size**
Downsampling Layers	
Conv2D	(bs,channels,28,28)
ResNet block	(bs,channels,14,14)
ResNet block	(bs,channels,7,7)
ResNet Layers	
ResNet block $\times n$	(bs,channels,7,7)
Feedforward Layers	
BatchNorm	(bs, channels, 7, 7)
ReLU	(bs, channels, 7, 7)
AvgPooling	(bs, channels, 1, 1)
Flatten	(bs, channels)
Linear	(bs, 816)
ReLU	(bs, 816)
Linear	(bs, 200)
ReLU	(bs, 200)
Linear	(bs, 1)
ResNet Block	
BatchNorm	(bs, channels, dim, dim)
ReLU	(bs, channels, dim, dim)
Conv2D	(bs, channels, dim, dim)
BatchNorm	(bs, channels, dim, dim)
ReLU	(bs, channels, dim, dim)
Conv2D	(bs, channels, dim, dim)

Figure 7 shows the layout of the ResNet used to estimate the RUL using the turbofan engine data.

Hyperparameters such as learning rate, weight decay and batch size affect the optimization process when training the network

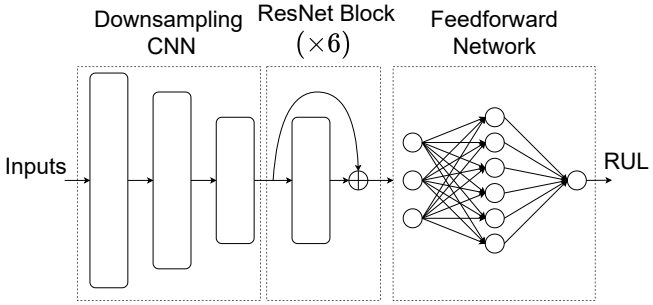


Figure 7. An illustration of the ResNet architecture layout used in this experiment. Note each rectangular block is a CNN and the $\bigoplus$ is an addition operator

and hence affects the overall performance of the trained network. Other hyperparameters such as the kernel/filter size of the convolutional layers or the number of filters the convolution layers use (convolution channel size), directly affects the network architecture and therefore also affect performance. The ResNet hyperparameters used in the experiments are stated in section A of the appendix.

4.2. NODE-CNN

Much like the ResNet described previously, the NODE-CNN layout (Figure 8) involves a downsampling layer that reduces the dimensions of the input data and extracts features. The main difference between the ResNet and the NODE-CNN network layout is that the ResNet Blocks described in the previous section are replaced with a NODE-CNN block. This block is effectively a CNN whose output (the rate of change of the hidden network variables) is used in a numerical ODE solver and propagates the hidden state forward one layer. Finally, like the ResNet a feedforward network is used to transform the output of the previous CNN network to a single value which is the RUL estimate. The NODE-CNN layout and data output sizes of each layer are shown in Table 5. Similarly to the ResNet sizes the hidden layer sizes here are not optimized but simply chosen based on similar CNNs used in prognostics tasks.

The NODE-CNN is similar to the ResNet, however, the ResNet blocks can be seen as using Euler's method to solve for each blocks output. For this NODE-CNN the Runge-Kutta method is used instead; this improves how the trajectory of the differential equation is propagated through the network layer space. The reason for using Runge-Kutta instead of an adaptive solver is due to the more reliable performance of using a fixed number of steps (the solver will not stop half-way through due to stiffness of the ODE). It is also used because of the size of the problem and the networks involved which caused large computation times when using adaptive solvers. Hence, the Runge-Kutta (RK) solver was used to train the network reliably and in a reasonable amount of time. Like the

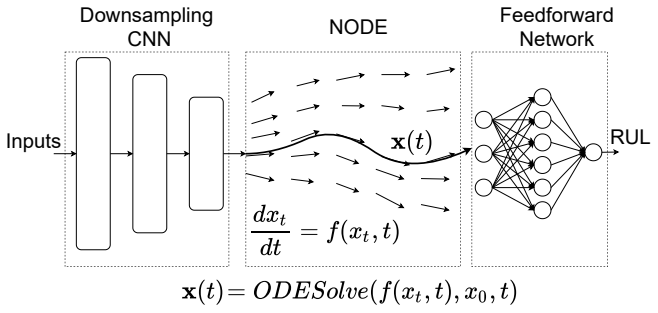


Figure 8. An illustration of the NODE-CNN architecture used for this experiment. Note that $f(x, t)$ is the CNN that describes the gradient at any point (x, t) and is used in the numerical ODE solver to propagate the trajectory forward. In this case t is not time but the depth of the NODE network.

Table 5. Architecture of the NODE-CNN

Layers	Layer Output Size
Downsampling Layers	
Conv2D	(bs, channels, 28, 28)
BatchNorm	(bs, channels, 28, 28)
ReLU	(bs, channels, 28, 28)
Conv2D	(bs, channels, 14, 14)
BatchNorm	(bs, channels, 14, 14)
ReLU	(bs, channels, 14, 14)
Conv2D	(bs, channels, 7, 7)
CNN-NODE	
BatchNorm	(bs, channels, 7, 7)
ReLU	(bs, channels, 7, 7)
Conv2D	(bs, channels, 7, 7)
BatchNorm	(bs, channels, 7, 7)
ReLU	(bs, channels, 7, 7)
Conv2D	(bs, channels, 7, 7)
BatchNorm	(bs, channels, 7, 7)
Feedforward Layers	
BatchNorm	(bs, channels, 7, 7)
ReLU	(bs, channels, 7, 7)
AvgPooling	(bs, channels, 1, 1)
Flatten	(bs, channels)
Linear	(bs, 816)
ReLU	(bs, 816)
Linear	(bs, 200)
ReLU	(bs, 200)
Linear	(bs, 1)

ResNet, hyperparameters were chosen through the Bayesian optimization procedure. The NODE-CNN hyperparameters used in the experiments are stated in section B of the appendix.

5. RESULTS AND DISCUSSION

To compare the performance of the models the RMSE between the RUL estimates for the testing dataset and the actual RUL values are used as a performance indicator. The RMSE can be calculated as shown in Eq. (9),

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}. \tag{9}$$

The RMSE is commonly used in many similar works when dealing with the C-MAPSS dataset allowing for direct comparison between different methods. RMSE values are acquired for each of the C-MAPSS testing datasets were calculated using Eq. (9) (Table 6). The other testing performance criterion used was the score originally introduced in (Saxena et al., 2008). A normalized version of this score is shown in Eq. (7) which was used for hyperparameter tuning, however, other prognostics methods use the non-normalized version (Eq. (10)) and it will therefore be used for comparison here.

$$s = \begin{cases} \sum_{i=1}^{N} \exp\left[\frac{-(\hat{y}_i - y_i)}{13}\right] - 1, & \text{if } \hat{y}_i \leq y_i \\ \sum_{i=1}^{N} \exp\left[\frac{(\hat{y}_i - y_i)}{10}\right] - 1, & \text{if } \hat{y}_i > y_i \end{cases} \tag{10}$$

Table 7 shows the (non-normalized) scores compared with the other prognostics methods. Table 8 shows the times taken to train the networks as well as the times taken to evaluate each testing dataset. Data preparation occurred before training and hence, these times are not fully indicative of real time performance. However, the times taken for the STFT to generate the images are given in the table as well. Different hyperparameter choices (e.g. time window) affected the time values, hence, the times for each network and dataset was given. Also note due to the different amount of spectrograms/images being generated based on the $split$ hyperparameter, the total times can differ significantly; for ease of comparison the time taken to generate one image was stated in the table under the column "STFT times".

Each unit in the testing dataset went through the same data preparation process as the training set which involves normalizing based on the operating condition and applying STFT on the signals. Figure 9 shows the box and whisker plots that were generated using the difference between estimated and target RUL values for each unit in the testing datasets. The figure gives an idea of the spread of errors between estimated RUL and the true RUL values over all the units in each testing dataset. Note that the negative values correspond to an estimate that was less than the true RUL, while positive values indicate a larger RUL estimate. Finally, Figure 10 shows how the RMSE on each testing dataset alters by adjusting number of steps in the numerical ODE solver. In this case the variable associated with "time", describes the depth of the NODE network. The network depth is controlled by choosing an interval ($t \in [0, 1]$) and splitting the interval into a even number of steps. Hence, increasing the number of steps increases the resolution, which in standard ODEs would increase the ac-

curacy at the expensive of computational time. For this test, the NODE-CNN was retrained with 10 steps using 50 epochs, so there were more step options to test before reaching zero steps when reducing number of steps. Note, the newly trained networks do not have optimal RMSE values; but this experiment is only concerned with the relative differences in RMSE as the number of steps changes. In regards to hardware, the training and testing of the neural networks were performed on a NVIDA GeForce RTX2080 GPU.

5.1. Discussion

The ResNet and NODE-CNN showed good performance when compared to the other methods in Tables 6 and 7. In particular, ResNet and NODE-CNN performed especially well on the FD002 and FD004 datasets. This is most likely due to the fact that these datasets have multiple operating conditions and it was shown in (Pasa et al., 2019) that performing operating condition based normalization (Eq. (5)) improves performance in this case. Hence, ResNet and NODE-CNN outperformed the other methods on the FD002 and FD004 datasets which did not use operating condition based normalization (only the methods from (Pasa et al., 2019) used this normalization technique). When comparing the performance on datasets FD002 and FD004 to (Pasa et al., 2019), it can be seen that NODE-CNN outperformed their regular (non-ResNet) CNN. The improved performance could be due to the fact that ResNet and NODE outperform regular CNNs in image recognition tasks (He et al., 2016; Chen et al., 2018), hence, the architecture of NODE-CNN may simply be an improvement to the simpler CNN architecture used in (Pasa et al., 2019). For the datasets with only one operating condition (FD001 and FD003) also had relatively good performance compared to state-of-the-art techniques. However, the best results for these datasets was achieved by (Listou Ellefsen et al., 2019) who used a Restricted Boltzmann Machine to extract degradation features and input them into an LSTM, which estimated the RUL. It should also be noted that (Listou Ellefsen et al., 2019) did not employ the same operating condition based normalization as was done in this study. Hence, it may still retain state-of-the-art performance for the FD002 and FD004 datasets if the sensor signals were normalized this way. The multi-scale CNN (Li et al., 2018) also outperformed the NODE-CNN and ResNet in terms of the RMSE measure for FD001 and FD003, but NODE-CNN and ResNet had better scores. This could be due to the fact both ResNet and NODE-CNN had their hyperparameters tuned to minimize the normalized score criterion and hence would produce more conservative RUL estimates; but not necessarily better RMSEs. However, a more detailed analysis would be required to state this as fact. While not outperforming all these techniques completely; the aim of showing NODEs and ResNets could compete with state-of-the-art techniques has been achieved.

Using the intuition of an ODE modeling a dynamic system, it may be expected that changing the time step-size used in the numerical ODE solver would affect the accuracy of the NODE-CNN. The altering of this step-size would normally provide a trade-off between accuracy and computation time. (Queiruga, Erichson, Taylor, & Mahoney, 2020) mentioned that the solution accuracy of NODEs does not change with step-size in the same manner as a standard ODE would. For their tests they used NODEs to model the dynamics of a mechanical system; while here NODE-CNN was used to model some unknown and more abstract dynamics of the hidden variables in a neural network. Hence, their accuracy refers to the error between the "true" trajectory and the NODEs trajectory at each time-point; while the error discussed here is the error between the final estimates the network produced and some known value. Their results showed that NODEs using 4th order Runge-Kutta did achieve some increased accuracy with increased resolution (decreasing time step) but not significant compared to what is expected by a numerical integrator ODE solver. Figure 10 shows the number of steps did not affect the testing loss on the final outputs except in extreme cases (number of steps altered by a factor of 10 from the original training value) before the error increased. This could be due to the fact the number of steps corresponds to the depth of the NODE-CNN and not necessarily an increase in resolution (and therefore accuracy). This suggests one must be careful when designing the network and specifying the problem if a relationship between step-size and solution accuracy is to be achieved. Here the network was used as a standard black-box, feedforward network and so it did not benefit from the accuracy-time trade-off. (Queiruga et al., 2020) also mention how NODEs can be altered to achieve this property; section 5.2 discusses this further.

Table 8 states the training and testing times of the NODE-CNN and shows they are significantly greater than the ResNets. For training times, this is partly due to the fact NODE-CNN had more training epochs as chosen by the Bayesian hyperparameter optimization (epochs are stated in Appendices A and B). Another factor is that NODE-CNN uses Runge-Kutta method for the ODE solver, meaning there are more functional evaluations at each time point. Euler's method would bring the times down closer to the ResNet times due to only evaluating the network once at each time point. Table 8 also shows the testing evaluation times for Euler method. Testing times are used as training times have more factors that influence them such as training hyperparameters, hence, testing times are easier to compare. The idea is to now discuss possible research avenues that could be explored which may provide benefits to machinery prognostics and can be achieved due to the unique architecture and properties of NODEs.

Table 6. RMSE performance on each of the testing CMAPSS datasets with maximum RUL value of 130

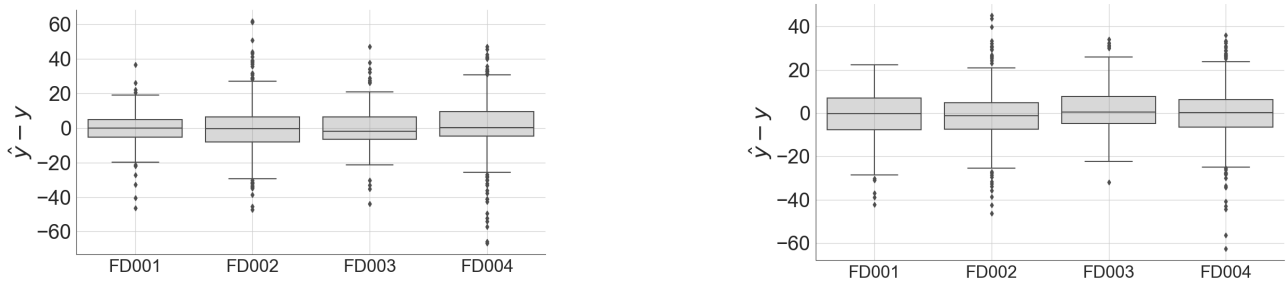
Method	FD001	FD002	FD003	FD004
ResNet (This work)	12.81	16.92	14.68	18.25
NODE-CNN (This work)	13.65	**14.30**	12.65	**15.06**
CNN (Pasa et al., 2019)	15.00	17.50	14.80	17.40
LSTM (Pasa et al., 2019)	16.50	18.10	15.90	17.20
MLP (Pasa et al., 2019)	15.10	18.00	14.30	16.60
CNN (Li et al., 2018)	12.61	22.36	12.64	23.31
LSTM (Listou Ellefsen et al., 2019)	**12.56**	22.73	**12.10**	22.66

Table 7. comparison of the scores (Eq. (10)) on each of the testing CMAPSS datasets with maximum RUL value of 130

Method	FD001	FD002	FD003	FD004
ResNet (This work)	250	2250	428	1842
NODE-CNN (This work)	235	**886**	270	**947**
CNN (Pasa et al., 2019)	369	1757	332	1678
LSTM (Pasa et al., 2019)	444	942	718	1487
MLP (Pasa et al., 2019)	411	1113	1091	2755
CNN (Li et al., 2018)	273	10412	284	12466
LSTM (Listou Ellefsen et al., 2019)	**231**	3366	**251**	2840

Table 8. Network training times (with the data already prepared) and testing times when evaluating the trained network on new data. Note RK4 stands for Runge-Kutta (4th order) method when solving NODEs. Euler's method testing times is also shown to illustrate the time difference between the ODE solver methods. Times for the STFT are also shown. Note due to the $split$ hyperparameter each dataset has a different amount of images to prepare and therefore the time per image generated is given for better comparison.

Dataset	NODE (RK4) Training time (s)	NODE (RK4) Testing time (s)	ResNet Training Time (s)	ResNet Testing time (s)	NODE (Euler) Testing time (s)	ResNet STFT time (s)	NODE STFT time (s)
FD001	405.847	1.076	294.622	0.341	0.653	0.157	0.145
FD002	645.829	2.780	567.658	0.909	1.276	0.381	0.388
FD003	188.352	1.066	73.068	0.344	0.522	0.162	0.159
FD004	697.284	2.669	349.930	0.861	1.226	0.387	0.384



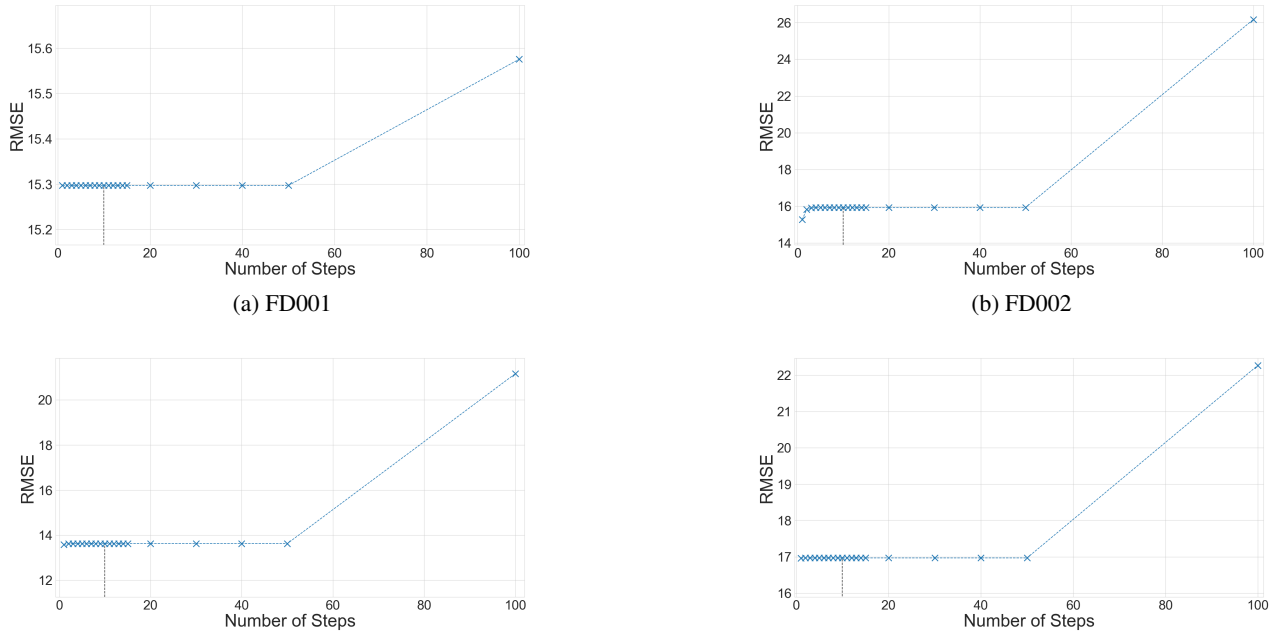
(a) ResNet

(b) NODE-CNN

Figure 9. Box and whisker plot for the ResNet (9a) and NODE-CNN (9b) RUL differences ($\hat{y}$ = estimated RUL and y = true RUL) over all the units for each testing dataset

5.2. Future Work

The continuous nature of NODE allows for adaptive step-sizes or increasing/decreasing the resolution of step sizes. This flexibility could provide certain benefits when it comes to the network performance. Although as mentioned in the discussion above, one must be careful with how the problem is framed if this property is to be achieved. (Queiruga et al., 2020) essentially achieve this by making the network parameters a function of time and refining step-sizes as training time increases. A network could be trained with higher resolution (low step-size over a specific interval) and then the step-size could be reduced or customized depending on the computing power available or required accuracy of the task. Simi-



(a) FD001

(b) FD002

(c) FD003

(d) FD004

Figure 10. Graphs showing testing performance as measured by the RMSE metric verses the number of steps used for the numerical integration method (4th order Runge-Kutta). Here the network was retrained using 10 steps in the ODE solver otherwise the same hyperparameters mentioned in Appendix B were used, but 50 epochs are used for faster training as the relative performance is of interest here. During testing the number of steps was altered and the RMSE was recalculated for the testing dataset as shown on the plot. The vertical dotted line indicates the number of steps the network was trained on.

lar topics on how to train NODEs with higher resolution and then lowering it while retaining an acceptable accuracy can be found in (Queiruga et al., 2020). Another benefit could come from the differential equation formulation of the network. This allows for interesting avenues to explore by exploiting the properties of differential equations. For example, NODEs can be extended to neural stochastic differential equations to incorporate uncertainty into the network outputs (Li, Wong, Chen, & Duvenaud, 2020). Uncertainty quantification of RUL estimates is considered a challenge in machinery prognostics (Lei et al., 2018), hence, this differential equation approach to neural networks could provide for interesting solutions to this problem.

6. CONCLUSIONS

It has been shown that the ResNet and CNN-NODE can perform competitively to other state-of-the-art machinery prognostics methods. This could lead to NODE being utilized and adapted for machinery prognostics tasks due to the potential benefits of treating neural networks as differential equations. The main benefits discussed were adaptive network sizes and quantifying uncertainty in estimates. NODEs describe a vector field which in this case propagates the hidden network variables; starting from the inputs and letting the variables at the end of the trajectory become the outputs. The amount of steps to get from the initial layer to the final layer however can be chosen by the user. Hence, there is a possibility to train a network and then depending on the computing power available picking the appropriate step-size for the ODE solver at the cost of the accuracy of the solution. Ideas like this have been explored in (Queiruga et al., 2020) but have yet to be applied in a machinery prognostics setting. The other benefit of using NODE or extending NODE to other differential equation types is that differential equations are well understood compared to deep neural networks. With knowledge of differential equations NODE could be extended to solve stochastic differential equations (Li et al., 2020) and include confidence intervals for the RUL estimates. By including confidence intervals for RUL estimates this would capture uncertainty and would allow for improved decision making capability in machinery prognostics tasks.

REFERENCES

Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G., & Bakshy, E. (2019). Botorch: Programmable bayesian optimization in pytorch. *CoRR*, *abs/1910.06403*. Retrieved from http://arxiv.org/abs/1910.06403

Blundell, C., Cornebise, J., Kavukcuoglu, K., & Wierstra, D.

(2015). Weight uncertainty in neural networks. *ArXiv*, *abs/1505.05424*.

Chen, T. Q., Rubanova, Y., Bettencourt, J., & Duvenaud, D. (2018). Neural ordinary differential equations. *CoRR*, *abs/1806.07366*. Retrieved from http://arxiv.org/abs/1806.07366

Dourado, A., & Viana, F. A. C. (2019). Physics-Informed Neural Networks for Corrosion-Fatigue Prognosis. In *Annual conference of the phm society* (Vol. 11, pp. 1–12). doi: https://doi.org/10.36001/phmconf.2019.v11i1.814

Dumoulin, V., & Visin, F. (2016). A guide to convolution arithmetic for deep learning. *ArXiv*, *abs/1603.07285*.

Frazier, P. I. (2018). *A tutorial on bayesian optimization.*

Gal, Y., & Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *ArXiv*, *abs/1506.02142*.

Guo, L., Li, N., Jia, F., Lei, Y., & Lin, J. (2017). A recurrent neural network based health indicator for remaining useful life prediction of bearings. *Neurocomputing*, *240*, 98–109. Retrieved from http://dx.doi.org/10.1016/j.neucom.2017.02.045 doi: 10.1016/j.neucom.2017.02.045

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-December*, 770–778. doi: 10.1109/CVPR.2016.90

Heimes, F. O. (2008). Recurrent neural networks for remaining useful life estimation. In *2008 international conference on prognostics and health management, phm 2008.* doi: 10.1109/PHM.2008.4711422

Hu, Y., & Luo, P. (2013, July.). Performance data prognostics based on relevance vector machine and particle filter. *Chemical Engineering Transactions*, *33*, 349-354. Retrieved from https://www.cetjournal.it/index.php/cet/article/view/CET1333059 doi: 10.3303/CET1333059

Huang, C. G., Huang, H. Z., & Li, Y. F. (2019). A Bidirectional LSTM Prognostics Method Under Multiple Operational Conditions. *IEEE Transactions on Industrial Electronics*, *66*(11), 8792–8802. doi: 10.1109/TIE.2019.2891463

Lei, Y., Li, N., Guo, L., Li, N., Yan, T., & Lin, J. (2018, May). Machinery health prognostics: A systematic review from data acquisition to RUL prediction. *Mechanical Systems and Signal Processing*, *104*, 799–834. doi: 10.1016/j.ymssp.2017.11.016

Li, X., Ding, Q., & Sun, J. Q. (2018). Remaining useful life estimation in prognostics using deep convolution neural networks. *Reliability Engineering and System Safety*, *172*(December 2017), 1–11. Retrieved from https://doi.org/10.1016/j.ress.2017.11.021 doi: 10.1016/j.ress.2017.11.021

Li, X., Wong, T.-K. L., Chen, R. T. Q., & Duvenaud, D. (2020, August). Scalable gradients for stochastic differential equations. In S. Chiappa & R. Calandra (Eds.), *Proceedings of the twenty third international conference on artificial intelligence and statistics* (Vol. 108, pp. 3870–3882). PMLR. Retrieved from http://proceedings.mlr.press/v108/li20i.html

Li, X., Zhang, W., & Ding, Q. (2019, Feburary). Deep learning-based remaining useful life estimation of bearings using multi-scale feature extraction. *Reliability Engineering and System Safety*, *182*, 208–218. doi: 10.1016/j.ress.2018.11.011

Listou Ellefsen, A., Bjørlykhaug, E., Æsøy, V., Ushakov, S., & Zhang, H. (2019, March). Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture. *Reliability Engineering and System Safety*, *183*, 240–251. doi: 10.1016/j.ress.2018.11.027

Nascimento, R. G., & Viana, F. A. C. (2019). Fleet prognosis with physics-informed recurrent neural networks. *CoRR*, *abs/1901.05512*. Retrieved from http://arxiv.org/abs/1901.05512

Pasa, G., Medeiros, I., & Yoneyama, T. (2019). Operating Condition-Invariant Neural Network-based Prognostics Methods applied on Turbofan Aircraft Engines. In *Annual conference of the phm society* (Vol. 11, pp. 1–10). doi: https://doi.org/10.36001/phmconf.2019.v11i1.786

Peng, W., Ye, Z.-S., & Chen, N. (2019). Bayesian Deep Learning based Health Prognostics Towards Prognostics Uncertainty. *IEEE Transactions on Industrial Electronics*, 1–1. Retrieved from https://ieeexplore.ieee.org/document/8681720/ doi: 10.1109/TIE.2019.2907440

Queiruga, A., Erichson, N., Taylor, D., & Mahoney, M. W. (2020). Continuous-in-depth neural networks. *ArXiv*, *abs/2008.02389*.

Saha, B., Goebel, K., Poll, S., & Christophersen, J. (2009). Prognostics methods for battery health monitoring using a Bayesian framework. *IEEE Transactions on Instrumentation and Measurement*, *58*(2), 291–296. doi: 10.1109/TIM.2008.2005965

Sankararaman, S. (2015). Significance, interpretation, and quantification of uncertainty in prognostics and remaining useful life prediction. *Mechanical Systems and Signal Processing*, *52-53*(1), 228–247. doi: 10.1016/j.ymssp.2014.05.029

Saxena, A., & Goebel, K. (n.d.). *Turbofan Engine Degradation Simulation Data Set.* Moffett Field, CA: NASA Ames Research Center. Retrieved from http://ti.arc.nasa.gov/project/

prognostic-data-repository

Saxena, A., Goebel, K., Simon, D., & Eklund, N. (2008, October). Damage propagation modeling for aircraft engine run-to-failure simulation. In *2008 international conference on prognostics and health management* (Vol. 41, pp. 1–9). IEEE. doi: 10.1109/PHM.2008.4711414

Wang, Q., Zhao, B., Ma, H., Chang, J., & Mao, G. (2019). A method for rapidly evaluating reliability and predicting remaining useful life using two-dimensional convolutional neural network with signal conversion. *Journal of Mechanical Science and Technology*, *33*(6), 2561–2571. doi: 10.1007/s12206-019-0504-x

Wu, J., Hu, K., Cheng, Y., Zhu, H., Shao, X., & Wang, Y. (2019). Data-driven remaining useful life prediction via multiple sensor signals and deep long short-term memory neural network. *ISA Transactions*, *97*, 241–250. Retrieved from https://doi.org/10.1016/j.isatra.2019.07.004 doi: 10.1016/j.isatra.2019.07.004

Yang, W., Yao, Q., Ye, K., & Xu, C. Z. (2020). Empirical Mode Decomposition and Temporal Convolutional Networks for Remaining Useful Life Estimation. *International Journal of Parallel Programming*, *48*(1), 61–79. Retrieved from https://doi.org/10.1007/s10766-019-00650-1 doi: 10.1007/s10766-019-00650-1

Yann Lecun. (1989). *Generalization and Network Design Strategies.*

Yucesan, Y. A., & Viana, F. A. C. (2019). Wind Turbine Main Bearing Fatigue Life Estimation with Physics-informed Neural Networks. In *Annual conference of the prognostics and health management society* (Vol. 11, pp. 1–14). doi: 10.36001/PHMCONF.2019.V11I1.807

Zhang, W., Yang, D., & Wang, H. (2019). Data-Driven Methods for Predictive Maintenance of Industrial Equipment: A Survey. *IEEE Systems Journal*, *13*(3), 2213–2227. doi: 10.1109/JSYST.2019.2905565

Zhang, Y., Xiong, R., He, H., & Pecht, M. G. (2018). Long short-term memory recurrent neural network for remaining useful life prediction of lithium-ion batteries. *IEEE Transactions on Vehicular Technology*, *67*(7), 5695–5705. doi: 10.1109/TVT.2018.2805189

Zhao, R., Yan, R., Chen, Z., Mao, K., Wang, P., & Gao, R. X. (2019, January). Deep learning and its applications to machine health monitoring. *Mechanical Systems and Signal Processing*, *115*, 213–237. doi: 10.1016/j.ymssp.2018.05.050

Zhou, Y., Huang, Y., Pang, J., & Wang, K. (2019). Remaining useful life prediction for supercapacitor based on long short-term memory neural network. *Journal of Power Sources*, *440*(March), 227149. Retrieved from https://doi.org/10.1016/j.jpowsour.2019.227149 doi: 10.1016/j.jpowsour.2019.227149

Zhu, J., Chen, N., & Peng, W. (2019, April). Estimation of Bearing Remaining Useful Life Based on Multi-scale Convolutional Neural Network. *IEEE Transactions on Industrial Electronics*, *66*(4), 3208–3216. doi: 10.1109/TIE.2018.2844856

BIOGRAPHIES

Marco Star is a mechanical engineering PhD student at Curtin University in Perth Western Australia where he also acquired his BEng. in mechanical engineering (2019). His research interests involve investigating deep learning methods for machinery prognostics. This includes quantifying uncertainty in RUL estimates resulting from deep learning methods, as well as exploring and altering network architectures to achieve useful properties in relation to machinery prognostics.

Kristoffer McKee has obtained his undergraduate degree (BEng in Mechanical Engineering) from The Cooper Union in New York, NY, USA in 1998. He has obtained his Masters of Science and PhD in Mechanical Engineering from Rensselaer in Troy, NY, USA in 2000 and 2003 respectively. He also has a Masters of Arts in Education, specializing in secondary math, which he obtained from Brooklyn College in Brooklyn, NY, USA in 2005. His research interests include machine diagnostics and prognostics, machine learning and deep learning, engineering education, and topics in humanitarian engineering.

APPENDIX

A. ResNet Hyperparameters

This appendix contains the hyperparameters used for training the ResNet on different datasets. Table 9 states the values of these hyperparameters.

B. NODE-CNN Hyperparameters

This appendix contains the hyperparameters used for training the NODE-CNN on different datasets. Table 10 states the values of these hyperparameters.

Table 9. Hyperparameters used to train the ResNet and affect its architecture. Hyperparameters for each C-MAPSS dataset are listed

Hyperparameter	FD001	FD002	FD003	FD004
Batch size (bs)	586	213	538	348
Learning rate	3.39×10^{-5}	6.97×10^{-3}	9.79×10^{-3}	2.21×10^{-5}
Weight Decay (L2)	2.10×10^{-4}	7.54×10^{-4}	4.31×10^{-4}	2.17×10^{-4}
Time Window	28	10	12	15
Data split ($split$ in Eq. (6))	184	197	64	123
Training epochs	99	57	59	62
Convolution Kernel size	3	3	3	3
Convolution channel size	64	64	64	64
Amount of ResNet blocks (n)	6	6	6	6

Table 10. Hyperparameters used to train the NODE-CNN and affect its architecture. Hyperparameters for each C-MAPSS dataset are listed

Hyperparameter	FD001	FD002	FD003	FD004
Batch size (bs)	206	580	207	482
Learning rate	9.22×10^{-3}	5.52×10^{-4}	1.11×10^{-3}	5.46×10^{-3}
Weight Decay (L2)	1.14×10^{-4}	1.08×10^{-9}	3.26×10^{-9}	2.75×10^{-8}
Time Window	30	10	38	15
Data split ($split$ in Eq. (6))	103	70	96	75
Training epochs	136	149	65	148
Convolution Kernel size	3	3	3	3
Convolution channel size	64	64	64	64
steps for RK solver	5	5	5	5