Remaining Useful Life Estimation Using Temporal Convolutional Network Fusion

A Thesis Presented

by

ZVONIMIR DUJMIĆ

Submitted to the Faculty of Science of the
Vrije Universiteit Amsterdam in partial fulfilment
of the requirements for the

MASTER'S DEGREE

August 2020

Master Artificial Intelligence

1 TABLE OF CONTENT

2 INTRODUCTION

Engineering and prognostics maintenance of sophisticated machinery, such as turbofan engines, are crucial and expensive tasks. Breakdowns in machines main components can be a significant contributor in increased running costs, unplanned stops and even catastrophic failures in machines. Because of increasing system complexity trends, standard scheduled preventive maintenance is becoming less capable of meeting the increasing industrial efficiency demands. To increase machines reliability, availability and efficiency, condition-based monitoring set itself as a solution, as intelligent prognostic and health management (PHM) technologies become more capable of dealing with the task. To improve the efficiency of maintenance schedules, avoid engineering failures and implement cost savings, the estimation of remaining useful life (RUL) in machines is a crucial part of PHM systems.

The existing methods of predicting RUL can be categorized into four groups according to their techniques and methodologies: model-based, data-driven, statistical-based and hybrid approaches [1][2]. Currently, the most successful ones are statistical-based approaches, priory with the advancement of machine learning. Now, the most common method is to use one of the variations of neural networks like LSTM [3], [4]or DCNN [5], [6], due to its ability to reuse on different machines and simplicity of making.

In this thesis, we introduce one novel method in remaining useful life prediction, as well as comparison and reviews of state-of-the-art techniques in this type of task. Our approach is mostly based on successful Wave Net [7] architecture which uses temporal convolutional networks with dilations to cope with long temporal dependencies. We extend that method by combining multiple models with different window sizes to better cope with different input (window) sizes. Furthermore, in this work, we experiment with different types of autoencoders. They are used for dimensionality reduction as well as for smoothing the data. The data that we use is NASAs C-MAPSS dataset [8], which contains sensory data about turbofan engines. Each sensory data is recorded in one timestamp and timestamps are recorded at equal time intervals, meaning that data has a correlation between neighbouring values. The data that is collected in such a manner is also known as temporal data.

2.1 Background

Many data-driven approaches tried to cope with a problem of predicting remaining useful life or similar tasks known as a sequence to target prediction. Predictive maintenance is necessary because we want to maximize the usage of a specific machine, or we want to increase safety. Usually, maintenance can be held periodically or ad-hoc and that is when it is already too late. The goal is to extend lifespan with as few maintenances as possible. Making maintenance only when is necessary, but before failure, decreases the number of interventions needed and that decreases the cost of maintaining such machine [9]. With the advance of machine learning, one of the main approaches is to train a model that will predict does machine need maintenance or not. Before we can use models to make a prediction, we need to collect the data. The collected that can be from various sensors, like temperature, speed, accelerometer, etc., and of course we need to record when failure happens. The data should be collected in the same interval sizes, and each sensory value should be collected in the same timestamp.

There are three common ways to estimate how long machine will work: similarity model, survival model and degradation model [10]. When only data from the time of failure is available, then the survival model is the only choice. It looks into a current number of cycles of the machine and how much cycles usually that or similar machine can work. Based on that distribution make a prediction. If failure data is not available, but we have a knowledge of a safety threshold, the degradation model is then the choice. It can make a degradation model based on previous data from our machine to predict how the condition indicator will change in the future. This way, we can statistically estimate how many cycles it has left until condition indicator crosses the threshold. Finally, if we have previous condition data (sensory data) and recorded failure, we can use the similarity model like we use in this project. The division of these three models, similarity, survival and degradation model is based on the type of available data.

Furthermore, when making predictions using a similarity model, the main approach is to use one of machine learning models. Those models are specialized in finding patterns among data and making predictions based on it. In theory, for any set of data, we can make a function that will represent that dataset. By knowing a function, it is easy to fill gaps between numbers or predict which number comes next. Similar to that, we have input to a function, sensory data, and result that function returns, prediction of failure.

Furthermore, we can make division based on what exactly they predict. That would be binary classifiers and regression models. Binary classifiers predict only true or false values. In this case, it would be, will machine fail or not. In this type of task that is usually showed in the way that we have some window size in which we say what will happen with the machine. For example, if we take a window size of 10, we could say that the machine in the next ten cycles will fail. The problem with that is that we do not know how long the machine will work. If we say that the machine in the next ten cycle will be ok, we do not know if there is a high chance of failure in the eleventh cycle. To solve that problem, the regression model is more appropriate. It estimates how long or how many cycles are left to a machine. This task is harder to solve accurately but is more valuable due to its preciseness, and that is the task we are dealing with in this project. There are various machine learning approaches that cope with this problem. The most popular and successful are using Long Short-Term Memory (LSTM) or temporal convolutional neural networks (TCN)[11][6][5]. More about them is described in the Methods section.

2.2 Description of the problem

The remaining useful life of a machine is defined as the time difference from the current time until the end of useful life [12]. The meaning of the word „useful life“ can vary according to the study of the field, and it can generally be defined by the person who builds, maintains or uses the machine. The concept of remaining useful life is shown in Figure 1, as well as the evolution of the deterioration of the given system. In the graph on the left, we can see the system where we know the threshold when the maximum deterioration is reached. In that case, RUL is the time between point A, that represents the current time, and point B, which represents the maximum acceptable condition of deterioration. Also, point be can be set to the point of failure, which is when 100% of deterioration is reached. Consequently, „useful life“ can be defined from the beginning of the lifetime or some time point until the failure or some timepoint between starting point and failure.

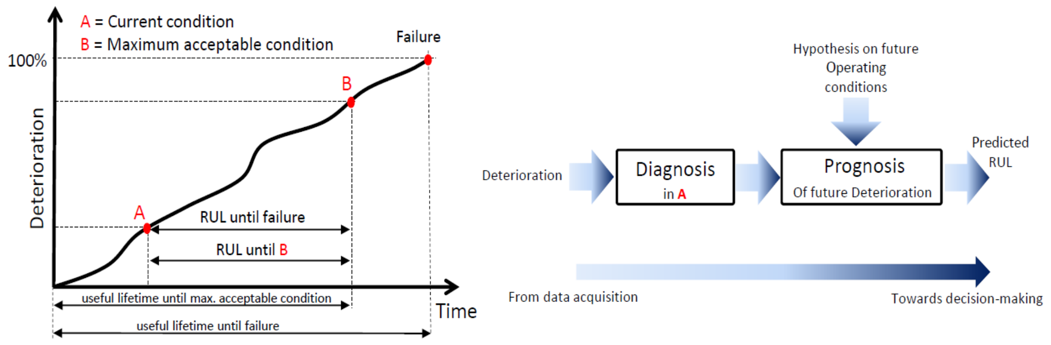
A = Current condition
B = Maximum acceptable condition
Failure
100%
Deterioration
B
A
RUL until failure
RUL until B
useful lifetime until max. acceptable condition
useful lifetime until failure
Time
Hypothesis on future Operating conditions
Deterioration
Diagnosis in A
Prognosis Of future Deterioration
Predicted RUL
From data acquisition
Towards decision-making

Figure 1 Concept of remaining useful life [13]

This task deals with predicting the future behaviour of the engineering system, which always come with several sources of uncertainty, each of those influence predictions. The uncertainties that affect prediction can be endogenous; those are one whose initial condition of the machine and its dynamical behaviour; exogenous which are measurements noise and process disturbances and future of the system which can be changed in the operating condition of the machine. The best RUL prediction would be one with high accuracy and high precision. As illustrated in Figure 1, the RUL prediction must include additional assumptions about the future of the operating condition, unlike a physical magnitude, which can be quantified with an uncertainty level in a given time. By using a model, RUL prognostics projects the diagnosis of the system in the current time, or in the case represented in Figure 1 in point A.

To conclude, the model is trying to predict when will a machine become unusable or unsafe to use. Many parameters affect when will that happened like the quality of build of a specific machine, a condition where it operates or the way it was handled in operations. All those variables create a lot of variables which make it hard for a model to predict the exact time of failure.

2.3 Contributions

The research described here culminates in several original contributions to the field of remaining useful life prognostic. These contributions lie mainly in the area of in predicting number cycles till failure. Additionally, various methods were tested in the purpose of enhancing results. The summary of these contributions is listed here.

- In this thesis, an approach to estimate the remaining useful life of a turbofan engine is proposed. The approach is based on temporal convolutional networks (TCN). TCN is a form of convolutional neural networks adopted for temporal data. It combines causal filters with dilated convolutions to allow their receptive fields to grow exponentially with depth, in the purpose of modelling the long-range temporal dependencies.

- Temporal convolutional network fusion is proposed to enhance the prediction of remaining useful life. It combines several TCNs; each of those TCNs has a different input window sizes that eventually fuse into one network to make an accurate prediction.

- One of the contributions is data analysis where we address the problem of linear decrement of RUL and nonlinear change in sensory data that are found in sensory data of C-MAPSS dataset and possibly in other data related to the prediction of remaining useful life.

- In this thesis, we explore different parameters of temporal convolutional architecture and present which parameters increase the efficiency of predictions and which parameters decrease it.

- Different versions of autoencoders are used in this project to test if they can be useful to increase the accuracy of predictive models. One version is used to smooth the data; another version is used for dimensionality reduction.

2.4 Research questions

The project aims to answer the following research questions:
Note here that remaining useful life (RUL) prediction is based on C-MAPSS dataset described in the Data analysis section and what is RUL is described in section Description of the problem.

1. Is it possible to use temporal convolutional networks to outperform other known methods such as LSTM in the task of predicting remaining useful life?
2. How autoencoders affect results in the task of predicting remaining useful life when combining with the temporal convolutional network or with long-short term memory models?
3. How different parameters of temporal convolutional networks affect results in the task of predicting remaining useful life?
4. How the implementation of fusion, e.g. combining different models with different window sizes into one affect prediction of remaining useful life?

2.5 Thesis outline

The general purpose of this thesis was to make a study about predictions on temporal data, various model types and techniques that could improve results.
In section Literature research, we investigate the current state of the art methods that deal with this task. Criteria that we apply for selecting papers is how much other resources have used them. Also, there are some notable mentions as well. In the Methods section will be described algorithms that we use in this thesis. Those are TCN, LSTM, autoencoders and fusion. In that section will not be explained how we use it in this task only what they are, how they work, the mathematics behind it, etc.
In Data analysis section, the dataset that is used for this thesis as well as for other papers in the literature research section will be described. That is C-MAPSS dataset created by NASA that contains sensory data from turbofan engines. In that dataset, there are a couple of exciting things to be explained, some of them are; the importance of each sensor, noise, temporal changes of values and much more.
Next section in this thesis is the experiments section. In there, we explain all of our experiments done in this project. The experiments were mainly focused on how to improve predictions.
Results of those experiments, our interpretation of findings and comparison to other state-of-the-art results on the same tasks are presented in the results section. In the

end, there is a discussion section where we explain our approach, findings and contribution for this task.

During the research, we have found some interesting things related to this task. Many ideas have been tested through experiments but not all of them. For the ideas that might work and possible solutions on how to fix current problems, there is a Conclusion and future work section where we explain those and why they might work. Also, that section contains a conclusion where we conclude everything that is done in this thesis.

3 LITERATURE RESEARCH

This task can be considered as part of a larger field called Prognostic and health management (PHM). Current approaches for it can be divided into four main categories; model-based, data-driven, statistical-based and hybrid approaches [1][2].

3.1 Model-based approaches

Model-based use empirical physical models and mathematical models to interpret machine degradation over time. One of the first that kind of approaches is Paris-Erdogan (PE) law that describes crack propagation [14]. Many other approaches use PE technique to deal with the problem, like Jouin et al. which proposed a model-based method for Proton Exchange Membrane Fuel Cell health assessment and prognostics [15]. Kacprzynski et al. proposed approach for gear health based on predictions that are made through the fusion of stochastic physics-of-failure models. Weng et al. proposed a three-loop Monte Carlo simulation scheme with Multi-State Physics Modelling to deal with the prognostic problem [16]. Model-based approaches are developed with prior knowledge of the failure mechanism and empirical estimation of model parameters. In other words, they can be accurate, but to develop them prior knowledge is important, and without them, it is impossible to use them.

3.2 Statistical based approaches

Statistical based approaches use only available past observed data and statistical models [12]. By that means, they do not need prior knowledge to develop a predictive model. They are effective in describing the uncertainty of the degradation process and remaining useful life estimation. That is the reason why they have been one of the most popular approaches among four main previously mentioned. Furthermore, condition monitoring data, or shorten CM, have proven helpful in boosting the performance of statistical models, especially with the estimation of the distance between the CM data and predefined threshold levels of the model.

3.2.1 Regression methods

Regression methods are widely used due to its simplicity in the task of remaining useful life. In autoregression, the future value is assumed as a linear function of past observations and random errors [17]. Quian et al. have presented an enhanced particle filter approach in combination with autoregression for predicting remaining useful life [18]. The experimental results show that this approach can improve the model fitting and the accuracy of the RUL estimation. Ibrahim et al. have combined autoregressive integrated moving average and polynomial regression to predict degradation in fuel cells [19]. Autoregression based methods are largely dependable on previous degradation trends, because of that, it can lead to unreliable predictions over time. Lu et al. 1993 have experimented with adding of random coefficients to the regression method to increase the stochastic variability of degradation [20]. It uses Monte-Carlo simulation to estimate the remaining useful life probability as well as nonlinear mixed effect model to identify degradation of the machine. At the time it gave promising results on the experimental dataset.

3.2.2 Wiener processes

Another notable method is Wiener processes. It can cope with different operating conditions and health conditions that lead to different degradation processes of the system, unlike autoregression models that relay on gaussian distribution. One of that kind of system is [21][22]. However, the drawback of those previous research works is that the Wiener processes are types of regression models, which are based on the assumption that the future state only depends on the current state and it is independent of the former state as it is the case with the Markov property. Si et al. proposed the solution for that, they implemented a particle filter to real-time estimate the non-Gaussian degradation state and random-effect parameter from measurements [23].

3.2.3 Gamma process

Gamma process is positive and strictly increasing, it is widely used for positive and strictly monotone deterioration processes, which can be useful in remaining useful life prediction where the only degradation is present. Also, the increments of degradation are modelled with an independent random variable. One of the approaches that use this method is mentioned in [24], where they use the Gibbs algorithm with stochastic filtering to find degradation state of the gamma process. Then remaining useful life can be estimated using gamma distribution and measuring distance to the predefined threshold. Same as Winer process, gamma process is based on the assumption that the future state only depends on the current state, and it is independent of the former state.

3.3 Hybrid approaches

Markov models make the assumption that for any given time, the conditional distribution of future states of the process given present and past states depends only on the present state and not at all on the previous states. Kharoufeh et al. propose one of such predictive hybrid models developed for RUL task. They use K-means clustering to estimate the number of states in a Markov model [25]. Problem with Markov models are hidden health states it cannot cope with. For that reason [26], [27] use hidden Markov models to overcome that problem. Cartella et al. propose Hidden Semi-Markov Models (HSMMs) with no constraints on the state duration density function and is applied to continuous or discrete observation. The model was employed to improve the flexibility of the HMMs for representing complicated state transition processes in the degradation of machines. Eventually, all Markov models are prone to the Markov property limitation which may lead to an approximation for the true process

3.4 Data-driven approaches

In the last couple of year deep learning has attracted intense interest in Prognostic and Health Management, because of its capability of representation, automated feature learning and best in class performance in slowing complex problems [28]. They tend to learn the degradation of a machine using optimization techniques form raw sensory data instead of building physic and mathematical models.

3.4.1 Multi-layer perceptions

One of the first methods used for predicting remaining useful life in this category was by Huang et al. [29], it uses multi-layer perceptron as well as condition monitoring to predict the results. Also, a notable example is an Artificial Neural Network by Tian

[30] that uses condition monitoring values at the present and past as the input and tries to predict RUL. One of the successful proposals that cope with RUL on C-MAPSS dataset is a multiobjective deep belief network (DBN) [31]. It uses a hybrid ensemble model (multiple DBNs) and an evolutionary algorithm to predict RUL.

3.4.2 Neural networks

Previously mentioned multi-layer perception gained much interest when they are improved by stacking multiple layers to fully capture the representative information from raw input data. With its high effectiveness in pattern recognition, they also become a popular method in intelligent prognostic [32]. Its deep complex structure is available to model high-level data abstractions which leads to more efficient feature extraction in comparison to shallow networks. They have gained most interest in fields of speech recognition[33], computer vision [34] and all similar fields where the input is raw high dimensional data. Similar to those is sensory data obtained for prognostic health management and remaining useful life estimation where the data is collected from multiple different sensors which can often produce noisy outputs. Bektas et al. proposed a neural network-based approach that uses data normalization and filtering methods for operational trajectories of complex systems as preprocessing of data in RUL estimation. It was tested on C-MAPSS dataset, and results were comparable to the state of the art methods at the time [35].

3.4.3 Convolutional neural networks

Babu et al. used a variation of deep neural networks, convolution neural networks, to estimate RUL of machinery [36]. His approach mostly based on techniques for image recognition, used two convolutional layers, two average pooling layers and one fully connected layer. The input is normalized variate time-series signals from sensors. The results have shown that convolutional neural networks can outperform the best method previously in regression task. Li et al. have also proposed a convolutional neural network approach but with raw input data combined in time windows for sample preparation to obtain better feature extraction [5]. The benefit of this is that little prior expertise on prognostics and signal processing is required. His model had difficulties extracting heterogeneous features representing the variation from running until the failure of the machine. So he proposed, in his next work, usage of Fourier transformation for time-frequency transformation and multiscale feature extraction to enhance previous CNN model [37]. To address the problem with the domain shifting problem, Li et al. proposed a cross-domain fault diagnosis method based on deep generative neural networks. He used the deep generative model to generate an artificial fault signal in the target domain, which is used in domain adaptation as training data [38]. Han Li et al. proposed a deep convolution neural network, which has three multiscale blocks, in it three different sizes of convolution operations are put on each block in parallel. That structure enables him to learn features of different scales.

3.4.4 Recurrent neural networks

Another notable approach for RUL estimation is to use Recurrent neural networks. They are widely used for sequence learning, like speech recognition, machine translation, natural language processing. [39]. Unlike deep neural networks which only feed-forward processed data, RNN has recurrent structure, meaning that processed data inside of a model flow in both directions. Furthermore, they implement the ability to remember better previous states, which enables them to learn long-distance relationships. Malhi et al. proposed approach for long term prognostics of machine health status based on RNN [40]. Gugulethu et al. [41] proposed Method that uses RNN to encode and drew and compare health index curve. Then it is compared to a normal health index curve to estimate RUL. RNNs, although good with this type of task, can suffer from gradient exploding and vanishing gradient problem during the backpropagation which is used for training of supervised tasks. Because of that, training is difficult and may not capture long term dependencies from the input time series [42] [43]. To overcome that problem, a particular type of RNN is developed. Long short-term memories (LSTM) implement input gates, forget gates, and output gates to overcome vanishing gradient problems and to avoid long-term dependency problems [44]. Because of its property to can capture long period temporal dependencies and nonlinear dynamics in the sequential signal, have used for speech recognition and machine translation where they achieve a state of the art results [45]. Use of LSTM on RUL estimation has used proposed by Hsu et a [4]. Wang et al. [46] Proposed bidirectional LSTM for RUL estimation task.

3.4.5 Hybrid models

Combination of LSTM and CNN is proposed by Zhao et al. [47], in that work, CNN is used for local features extraction and bidirectional LSTM for the temporal information encoding and representation learning. Combination of CNN and LSTM is proposed by [48] and [49] where instead of stacking one on another, they are used in parallel formation. The novel method of combining CNN and LSTM is proposed in by Li et al. where they use a directed acyclic graph network architecture [50]. Two outputs from CNN and LSTM are summed up and fed up into a second LSTM, whose output goes to last dense layer which predicts the result. The main problem with this architecture is a large number of tuning parameters which can lead to difficulties in practical applications. Song et al. [51] proposed a hybrid model that implements autoencoder and bidirectional LSTM to improve the prediction accuracy of RUL. Zhang et al use LSTM fusion to capture different dependency sizes. In their approach they use multiple LSTM in parallel with different window sizes and one dense layer on top to make a prediction of remaining useful life [11].

4 METHODS

Examining the literature related to the problem of predicting the remaining useful life, we have concluded that the best results were done using neural networks or some combinations of them like LSTM-Fusion [11] or MS-DCNN [6]. For that reason, we have decided to do most of our experiments with it. In this section, we will describe the methods that are used for this thesis. These methods only cover machine learning models that we use as well as related proposed methods. First, the overall about Neural networks is explained, followed by Long short-term memory. After that, we explain our primary method, Temporal convolutional network. Finally, two techniques that are used in experiments to enhance predictions, autoencoders and fusion are explained.

4.1 Neural Networks

A neural network is a computational model that is inspired by the way biological neural networks in the human brain process information. Like human brain neural networks (NN) have basic building block or neuron. In NN that is often called a node or unit. The unit receives input from other units or input from an external source. Each input has associated weight, which is assigned based on its relative importance to other inputs. The unit uses an activation function to determine the output based on the weighted sum of inputs. Common activation functions are Sigmoid, Tanh, ReLU. Learning is done using gradient descent. The goal is to find a local or global minimum of the cost function by summing up all of the difference between the actual and the expected output and then multiplying it by the learning rate.

$$\Theta_j \leftarrow \Theta_j - \alpha \frac{\partial}{\partial \Theta_j} \mathrm{J}(\Theta)$$

The $\mathrm{J}(\Theta)$ is one of the cost functions like Root mean square error or Mean squared error. The learning rate α determines how big of a step to take to convergence. In this learning process, only weights that are connected to nodes are updated.

There are many types of neural networks; the simplest one is Single-layer perception. It consists of a single layer of output nodes. Image representation is shown in Figure 2. Inputs are represented by $x_1, x_2, x_3, \dots, x_n$ with associated weights $w_1, w_2, \dots, w_n$. The output is calculated by summing inputs with weights and applying activation on it, also random bias b is added to the sum.

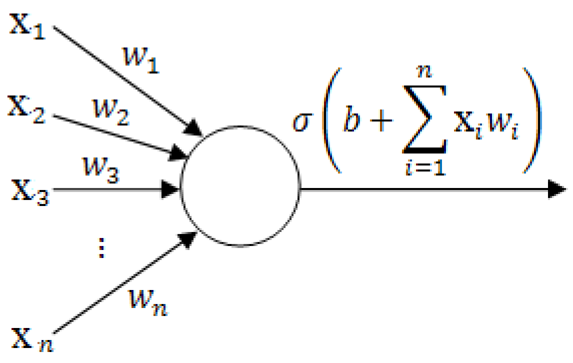

Figure 2 Single perception

Multi-layer perception is a more sophisticated type of network. Unlike single-layer perception, this type of network has multiple connected layers of perceptions, that enables more complex representations. Each unit in one layer has directed connections to the unit of the subsequent layer. Simple image representation is shown in Figure 3. This network has multiple inputs marked with $x_1, x_2, \dots, x_n$ and outputs $Y_1, Y_2, \dots, x_n$; it has one set of input layers($I_1, I_2, \dots, I_n$), one set of output layers($O_1, O_2, \dots, O_n$) and one hidden layer($H_1, H_2, \dots, H_n$). Each of this In, On or Hn is one perception (neuron) that is previously explained.

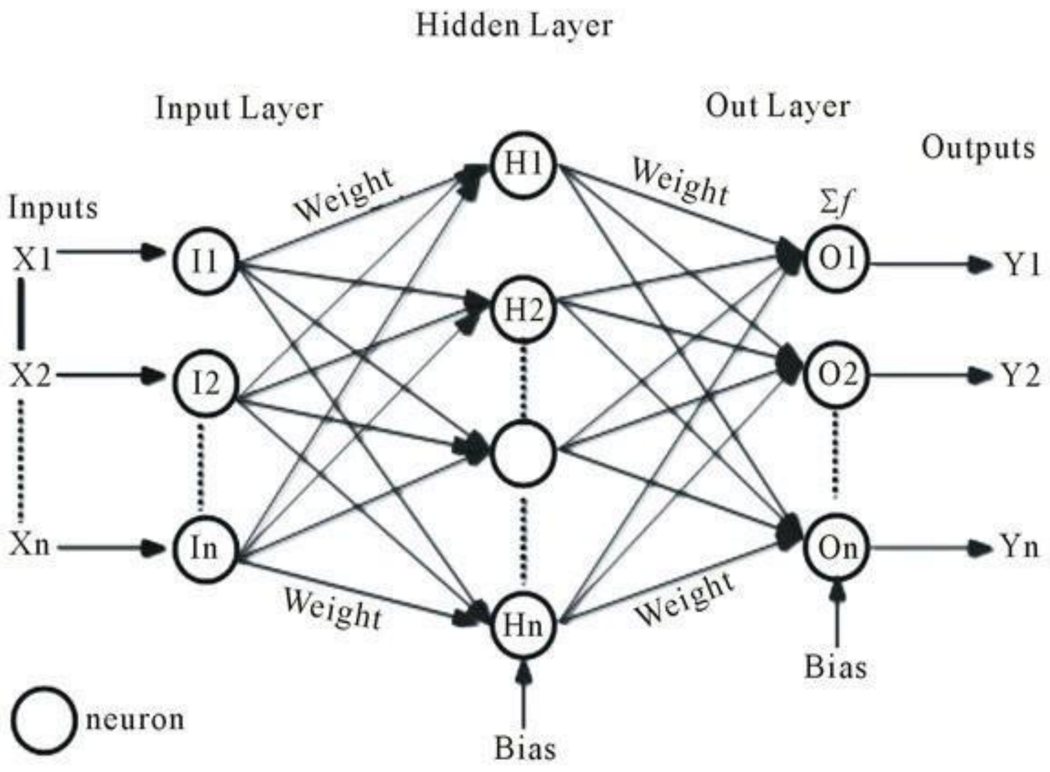

Figure 3 Multi-layer perception

Another type of neural networks is Recurrent neural networks (RNN). In them, connections between units form a directed cycle. Unlike previously mentioned architectures which propagate only data forward, RNN sends data in both directions from later processing stages to earlier stages. This allows it to exhibit dynamic temporal behaviour. Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition, speech recognition and other general sequence processors.

4.2 Long short-term memory (LSTM)

Long short-term memory networks or shorten LSTM are special kind of Recurrent neural networks, that are capable of learning long-term dependencies. They are introduced by Hochreiter et al. as a solution to problems that RNN has [44]. Namely, RNNs, even though they can cope with temporal data, they have a problem called vanishing gradient and exploding gradient, which makes them unable to link long term dependencies and for that they are unusable.
Like other RNNs, LSTM is also recurrent, meaning that they form a chain of repeating modules that connect each other in not just forward direction. However, unlike RNNs that use simple tanh function, LSTM has a more complicated structure of modules. Figure 4 shows how one such building block looks like. It has three inputs and three outputs. X_t is the input of the current timestamp, h_{t-1} is the output from the previous timestamp and C_{t-1} is the “memory” of the previous timestamp. It has two h_t outputs that pass current network output and C_t is the memory of the current unit. The single block decides by taking current input, previous output and previous memory. Moreover, it generates a new output and alters its memory.

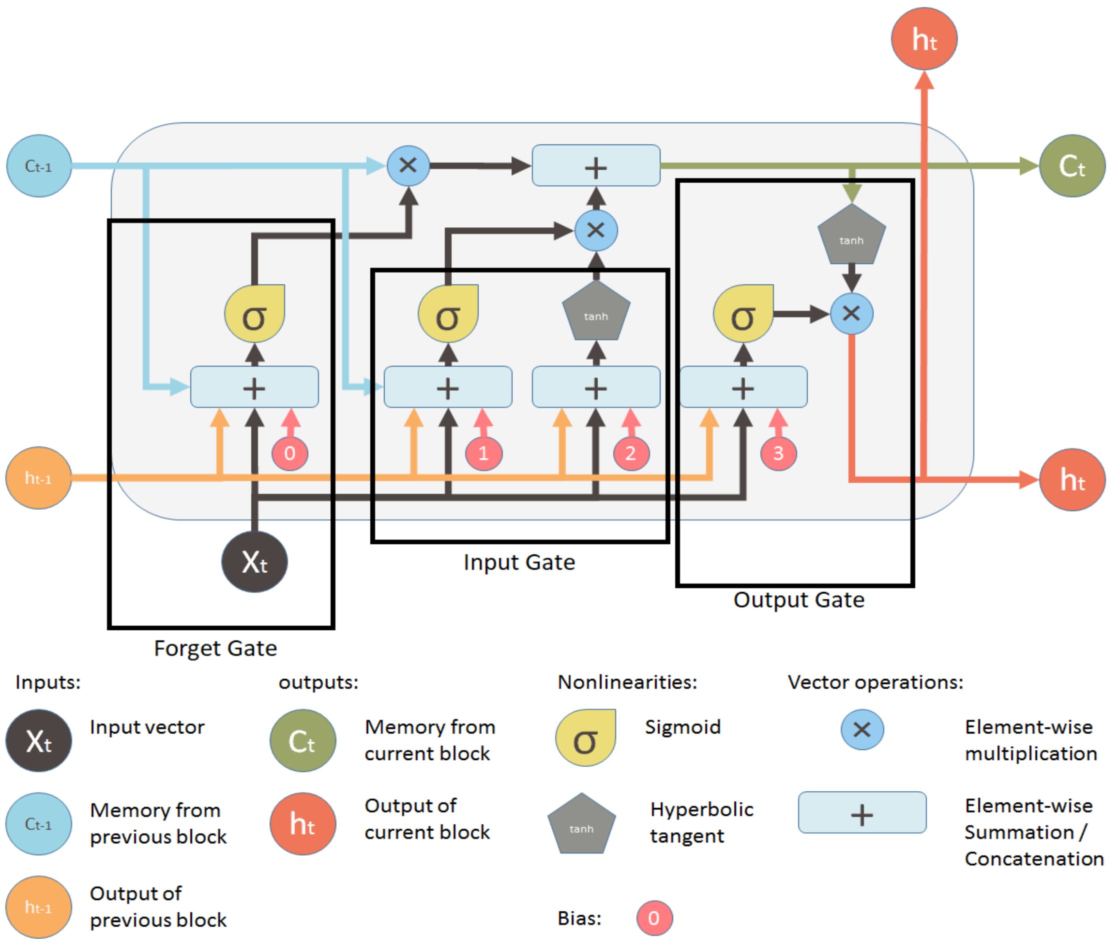

Figure 4 LSTM building block

LSTM consist of three main parts, input gate, forget gate and output gate.

1. Forget gate uses a sigmoid function to decide what details should be discarded from the block. It takes the output from the previous block h_{t-1} , input vector X_t , a memory from previous block C_{t-1} and bias to produce the forget value. That value is combined with old memory using element-wise multiplication. The formula for it is:

$$\mathrm{f}_t = \sigma(w_\mathrm{f}[h_{t-1}, x_t]) + b_\mathrm{f}$$

Where f_t Represents forget gate, σ is the sigmoid function, w_f are weights, h_{t-1} is output from the previous LSTM block, x_t represents input at current timestamp and b_f are biases.

2. Input gate decides which value from the input should be used to modify the memory. It has the same inputs as forget gate, but it has two functions, sigmoid and tanh. The output of this network will element-wise multiple the new memory valve and add to the old memory to form the new memory. Mathematically speaking sigmoid function is:

$$i_t = \sigma(w_i[h_{t-1}, x_t]) + b_i$$

Where i_t Represents the input gate, σ is the sigmoid function, w_i are weights, h_{t-1} is output from the previous LSTM block, x_t represents input at current timestamp and b_i are biases of the input function.

tanh function is:

$$C_{t+1} = tanh(w_c[h_{t-1}, x_t]) + b_c$$

Where C_{t+1} Represents vector of new candidate values, w_c are weights, h_{t-1} is output from the previous LSTM block, x_t represents input at current timestamp and b_c are biases.

Finally, the output looks like:

$$C_t = f_t * C_{t-1} + i_t * C_{t+1}$$

3. Output gate has the same structure as forget gate, but instead of combining with memory, it uses tanh function to give weightage to newly created memory which is then multiplied with output gate result. The formula of its sigmoid functions is:

$$o_t = \sigma(w_o[h_{t-1}, x_t]) + b_o$$

Where o_t represents the output gate, σ is the sigmoid function, w_o are weights, h_{t-1} is output from the previous LSTM block, x_t represents input at current timestamp and b_o are biases of the output function.

Finally, output gate is:

$$h_t = o_t * tanh(C_t)$$

Where C_t is output from the output function.

As mentioned before, these blocks are stacked and connected, as shown in Figure 5.

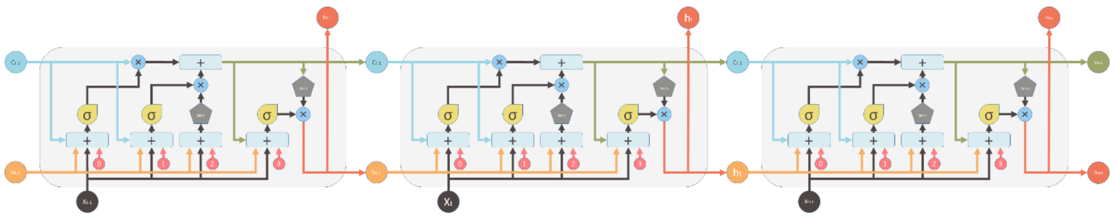

Figure 5 repeating modules of LSTM

Learning of LSTM is done using backpropagation through time. In feed-forward networks backpropagation moves backwards from the final error through the outputs, weights and inputs of each hidden layer, assigning those weights responsibility for a portion of the error by calculating their partial derivatives, or the relationship between their rates of change. Those derivatives are then used by our learning rule, gradient descent, to adjust the weights up or down, whichever direction decreases error. LSTM rely on an extension of that backpropagation, backpropagation through time (BPTT). There, time is simply expressed by a well-defined, ordered series of calculations linking one time step to the next, which is all backpropagation needs to work. Neural networks, whether they are recurrent or not, are simply nested composite functions like f(g(h(x))). Adding a time element only extends the series of functions for which we calculate derivatives with the chain rule, which means that now for each gate, we need to calculate derivatives and update weights.

4.3 Temporal Convolutional network (TCN)

In this section, we will describe our primary method, Temporal convolutional network, how it works, and why it is suitable for this type of task. The first idea comes from Googles DeepMind paper Pixel Recurrent Neural Network, where the authors use convolutional neural networks for sequence modelling with fixed dependency range [52]. They use convolution only in the elements from current timestamp or earlier in the previous layer. Another big breakthrough also comes from DeepMind. In this paper with temporal convolution, they introduce dilation and residual block. Those are the main parts of the temporal convolutional network [7]. The TCN is designed from two basic principles:

1. The convolutions are causal, meaning that there is no information leakage from future to past.
2. Its input can be a sequence of any length, and its output can be a map to a sequence of the same length [7].

The first one is achieved by using causal convolutions, or in other words, the output of convolution in time t is calculated only by taking time t element and an element that is before it. The second point is accomplished in a way that all hidden layers are the same length as the input layer.

Problem with simple causal convolution is that it only looks back at history with linear size, meaning that the receptive field grows linearly with every layer which results in slow training for large sequences. For that reason, using dilated convolutions enables using longer sequences. Dilated convolution, for input $x \in \mathbb{R}^T$ and a filter $f: \{0, \dots, k-1\} \to \mathbb{R}$ is defined as operation F on element s of the sequence, or more formally:

$$\mathrm{F}(\mathrm{s}) = (\mathrm{x} *_\mathrm{d} f)(\mathrm{s}) = \sum_{i=0}^{\mathrm{k}-1} f(i) \cdot x_{s-\mathrm{d}\cdot\mathrm{i}}$$

Where $d = 2^n$ is dilation factor, n is level of network and k is the filter size. The term $\mathrm{s} - \mathrm{d}\cdot\mathrm{i}$ stands for direction of the past. The image of dilated convolution is shown in Figure 6.

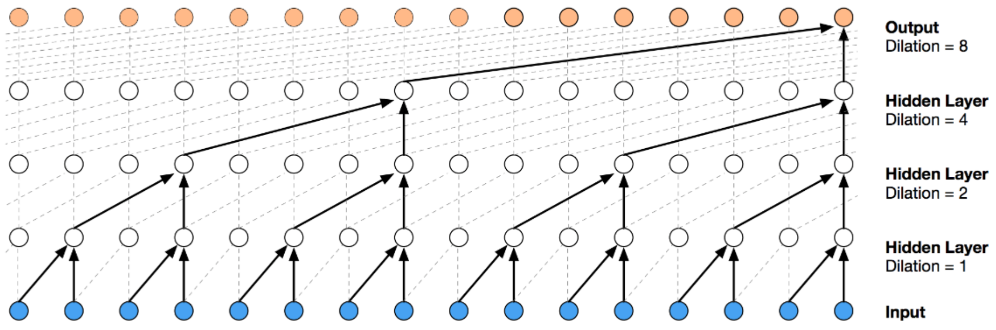

Figure 6 Dilated convolution example with input 2 and dilations 1,2,4,8 [7]

Larger dilation number enables output to represent a wider range of inputs, thus effectively expanding the receptive field of a network. The receptive field of a TCN can be increased by choosing larger filter size k or increasing the dilation factor d since the compelling history of one layer is $(k-1)d$.
Another essential architectural part of a TCN is residual blocks. TCN employ a generic residual module in place of a convolutional layer, each of these blocks contains a branch leading out to a series of transformations F whose outputs are added to the input x of the block.

$$o = Activation(x + F(x))$$

A residual block has two layers of dilated convolutions and rectified linear units (ReLU) as non-linearities. Also, it applies weight normalization and a spatial dropout which is added after each dilated convolution for regularisation. Example of a block is shown in Figure 7.

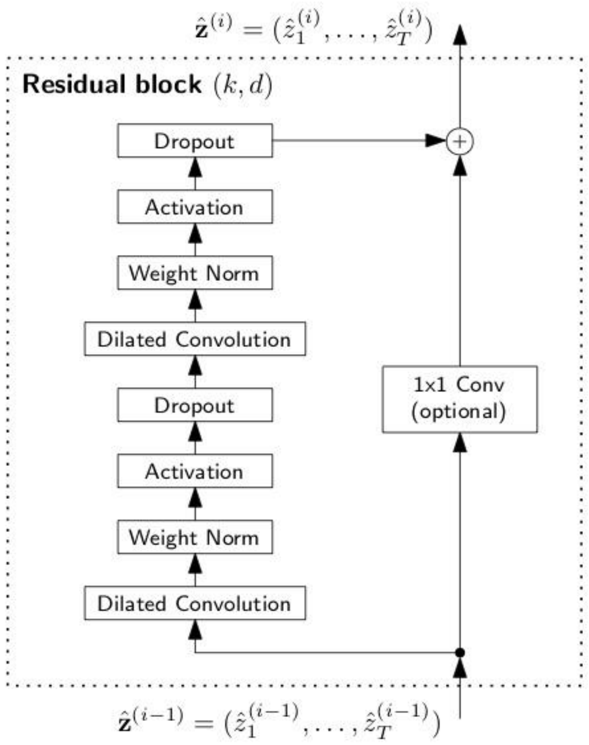
$\hat{\mathbf{z}}^{(i)} = (\hat{z}_1^{(i)}, \ldots, \hat{z}_T^{(i)})$
Residual block (k, d)
Dropout
Activation
Weight Norm
Dilated Convolution
Dropout
Activation
Weight Norm
Dilated Convolution
1x1 Conv (optional)
$\hat{\mathbf{z}}^{(i-1)} = (\hat{z}_1^{(i-1)}, \ldots, \hat{z}_T^{(i-1)})$

Figure 7 example of TCN residual block [1]

Main benefits of TCNs are parallelism, flexible receptive field size, low memory requirements and capturing local information. In RNNs predictions for later time steps must wait for the predictor to finish, which in TCN that is not the case. Its convolutions can be calculated in parallel because the same filter is used in each layer. Because of that, long input sequences can be processed as a whole instead of sequential as in RNNs. Because of its flexible receptive field size, TCNs have the ability to change its receptive field size in multiple ways. One of the methods is to stack more dilated convolutions; another is to increase filter size or using larger dilation factors. Therefore, it enables better control of the model's memory sizes, and it is easier to adapt to different domains. Unlike other methods, the TCNs require less time, as well as lower memory consumption for training in comparison to RNNs. This is mainly represented with long input sequences. That is because TCNs filters are shared across a layer, with the backpropagation path depending only on the network depth. Using convolution operation helps in capturing local information along with temporal information.

[1] Image downloaded from https://dida.do/blog/temporal-convolutional-networks-for-sequence-modeling April 2020

4.4 Autoencoders

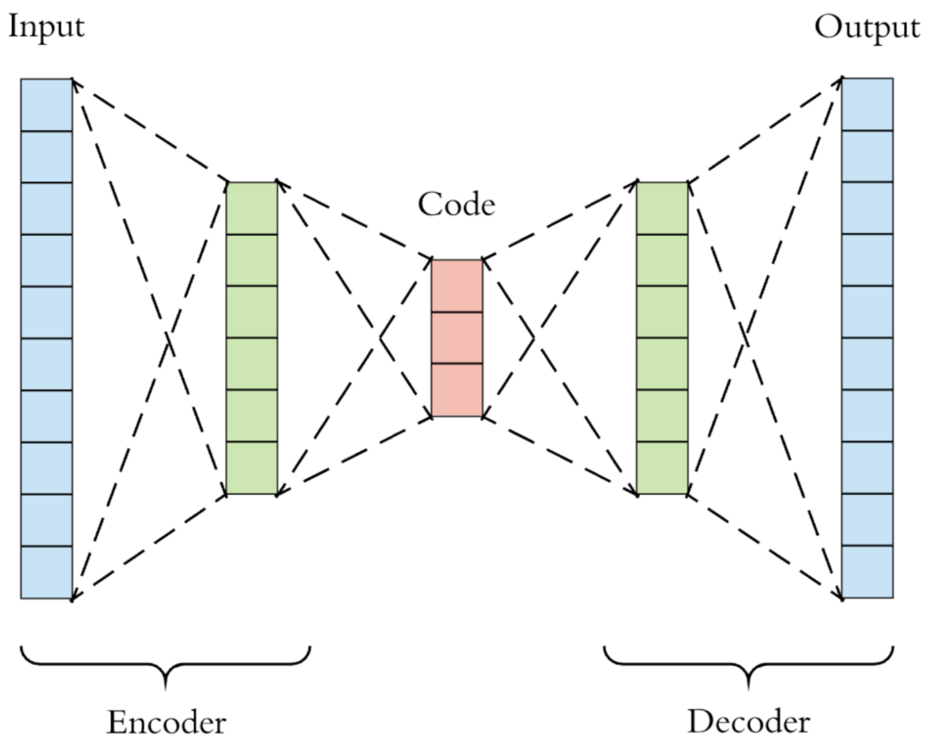

Figure 8 illustration of an autoencoder [2]

The temporal convolutional network can use large input sets and make predictions based on that, predictions that can be represented as an output vector. Furthermore, to test the power of TCNs, we can predict input from the output of the network. Surprisingly that has its own useful purpose and ist called autoencoder. Autoencoders are a form of compression. They are composed of two parts encoder and decoder. Usually, we build them from fully connected layers, but that does not have to be a case, we can also use TCN or LSTM or any other network type. In this part, we will focus on fully connected autoencoder. Encoder part encodes a data, and decoder, which input is decoders output tries to replicate input data. An illustrative example of the autoencoder network can be seen in Figure 8. They are related to principal component analysis (PCA). If we use linear activation function within our network, the bottleneck part, the one that comes out encoder will correspond to the principal component from PCA. The math behind it is quite simple:

$$\phi : X \rightarrow F$$

$$\psi : F \rightarrow X$$

[2] Image downloaded from https://blog.paperspace.com/autoencoder-image-compression-keras/ April 2020

The encoder function denoted with ϕ maps input vector X to latent space F. The decoder function ψ, maps latent space F to the output. In this case, the output is the same as the input. Thus, we are trying to reproduce the same vector as we have. The encoder can be represented by the standard neural network function: $z = \sigma\ (Wx + b)$, where z is the latent dimension. Similar to the encoder, the decoder can be represented in the same fashion but with different weight, bias and potential activation function $x' = \sigma(W'z + b')$. In terms of these function, we can write loss function, which is used to train the neural network through the standard backpropagation procedure.

$$\mathrm{L(x, x')} = \|x - x'\|^2 = \left\|x - \sigma'\left(W'\left(\sigma(Wx + b)\right) + b'\right)\right\|^2$$

One crucial thing to determine in autoencoder is how significant the bottleneck will be. That will determine the amount of compression and its quality. If it is too small, it will be hard to extract features from it. Furthermore, we want the bottleneck to be as small as possible. How exactly small it should be determined by testing different sizes and comparing results.

4.5 Fusion

In short, the purpose of fusion is to collect different sizes of temporal data models and combine them into one model, to get a model that has flexible input size and it is resistant to temporal variations. Fusion allows us to train a model that can cope with the temporal context of varying time steps. To train a model that predicts RUL, we use time-series sequences and the labelled remaining useful life. $(x_1, x_2, x_3, \dots x_t), y$ here y denotes labelled remaining useful life, and x_t denotes tensor of multiple sensory data $x = s_1, s_2 \dots . s_n$, where s represents each sensory value in a given timestep and n number of sensors. t stands for a number of time steps or window sizes.

Each model is trained separately; for each time-series sequence, the target RUL is the same. Meaning, in the given training example $\left\{(x_1, x_2, x_3 \dots, x_t)j, y_j\right\}_{j=1}^{N}$ stands that $y_j = y, \forall \mathrm{t}$. Here N denotes the number of models we will fuse, and j denotes jth window size in N.

The last thing left to do is to fuse those models. For that, we use high-level latent representations from the TCN subnetworks that were previously trained and feed them into another network. For that last layer, it is possible to use any other machine learning model. In our case, we have experimented with Neural Networks and Temporal convolutional network.

5 DATA ANALYSIS

Now that all methods used in the experiments are explained. We need to present in detail the dataset that was used. In this section, we will describe the dataset that is used for this thesis. In the first subsection, the overall about C-MAPSS dataset is written. The second subsection is reserved for data exploration, and in the final subsection, we explain preprocessing that is done in this thesis.

5.1 C-MAPSS dataset

The data that we use in theses thesis is simulated by C-MAPSS, created by NASA for the Prognostic and Health Management Challenge PHM08 [4], [53], [54]. It is used to predict the failure of turbofan engines, and it is publicly available. Even though C-MAPASS is an advanced simulation environment with many setups, it is still a simulation. And like that, It is not always able to completely recreate real-life situations just like it is not able to include every possible situation. However, for this task, it is more than suitable.
The data contains time-series measurements of various sensors, and the same measurements are typically measured in a commercial modern turbofan engine. All engines in the dataset are of the same type, but each starts with different degree of initial variations which are unknown to the user. The main components are high-pressure turbine (HPT), low-pressure turbine (LPT), fan, nozzle, high-pressure compressor (HPC) and low-pressure compressor (LPC). A simplified diagram of the simulation engine can be seen in Figure 9.

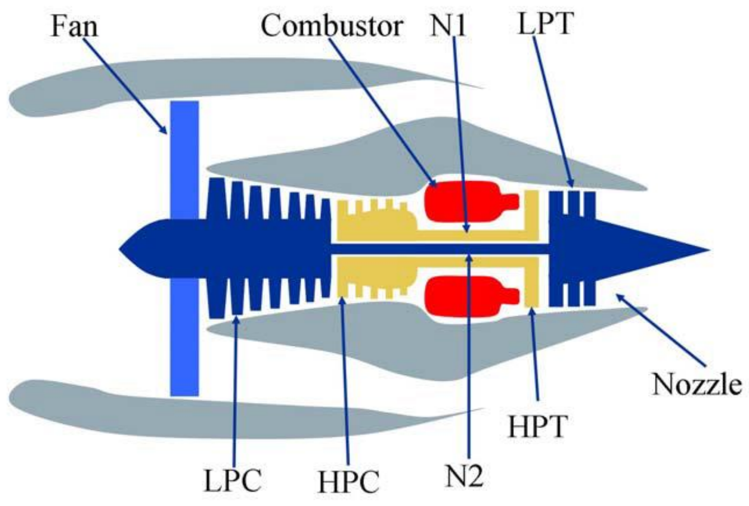

Figure 9 A simplified diagram of the simulation engine in C-MAPSS [54]

The C-MAPASS is a turbofan engine simulation environment it includes many editable input parameters which enables the user to enter specific values like fan flow, fuel flow, fan-pressure ratio etc. Its input contains 14 different factors that affect the degradation of the turbofan engine. Furthermore, the simulation produces output in the form of sensory values that, as a whole, can be interpreted as the health condition of the turbofan engine. It has 21 different outputs which are shown in Table 1. The table contains a column description which describes the sensory input. Also, the table contains column trend which describes a pattern over time how sensory values change, arrow up indicates that parameter is increasing over time, arrow down indicates that parameter is decreasing, while line indicates that parameter is irregular with time.

	Description	Trend
1	Total temperature at fan inlet	-
2	The total temperature at the low-pressure compressor outlet	↑
3	The total temperature at the high-pressure compressor outlet	↑
4	The total temperature at the low-pressure turbine outlet	↑
5	The pressure at fan inlet	-
6	Total pressure in bypass-duct	-
7	Total pressure at the high-pressure compressor outlet	↓
8	Physical fan speed	↑
9	Physical core speed	↑
10	Engine pressure ratio	-
11	Static pressure at high-pressure compressor outlet (Ps30)	↑

12	The ratio of fuel flow to Ps30	↓
13	Corrected fan speed	↑
14	Corrected core speed	↓
15	Bypass ratio	↑
16	Burner fuel-air ratio	-
17	Bleed enthalpy	↑
18	Demanded fan speed	-
19	Demanded corrected fan speed	-
20	High-pressure turbine coolant bleed	↓
21	Low-pressure turbine coolant bleed	↓

Table 1 Sensor outputs of the simulation engine

The C-MAPSS dataset contains four subsets or smaller dataset as they come in different files. Furthermore, those four have different operating conditions and fault modes. The description of them is described in Table 2. Each subset contains train and test data. Test data contains sensory values for some number of cycles between min and max number of cycles which is written in Table 2 until the engine stops working. Test data contains only one portion of cycles, that is not before the fault of the engine, and corresponding RUL.

Sub-dataset:	FD001	FD002	FD003	FD004
Engines in the training set	100	260	100	249
Engines in test set	100	259	100	248
Max/min cycles for training	362/128	378/128	525/145	543/128
Max/min cycles for test	303/31	367/21	475/38	486/19
Operating conditions	1	6	1	6
Fault modes	1	1	2	2
TW length	30	21	36	18
Training samples	17731	48558	21120	56815
Test samples	100	259	100	248

Table 2 Description of C-MAPSS dataset

5.2 Data exploration

In this section, we will do data exploration to show how data looks like. Worth to notice that this data, like all other sensory data, is contaminated with sensor noise. The sub-dataset that is used for this section is an FD001 train set. That dataset is selected because a large number of state-of-the-art paper test on it as well it was suggested by [50] as most representative dataset [5], [6], [11].

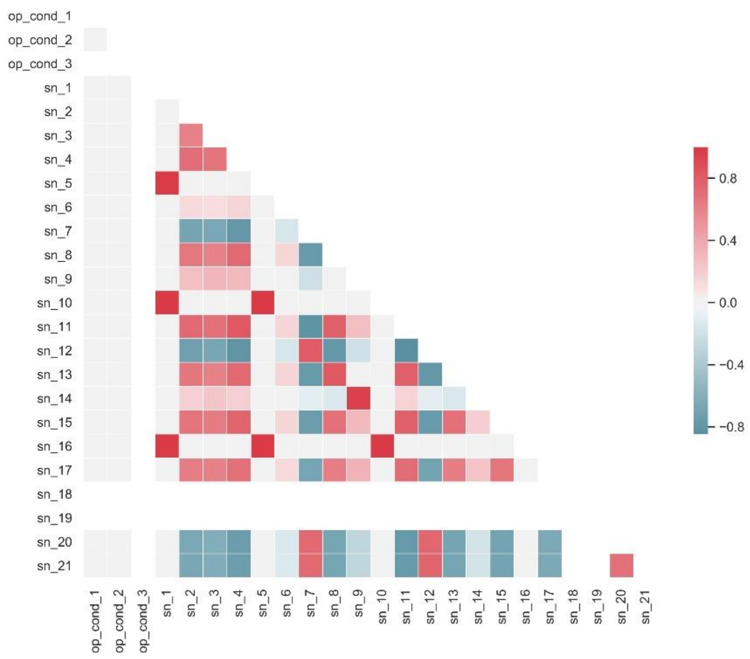

Figure 10 data correlation

In Figure 10, we can see the corelation matrix of data correlation for all engines, colour show how strongly sensory values are collated. Red means positive correlation; blue means negative correlation, and white indicate that there is no correlation at all. The stronger the colour, the larger the correlation magnitude. We can see that sensory values 1,5,6,10,16,18,19, that were previously indicated as not changing over time, are perfectly correlated. For that reason, those can be removed, but in our experiments, we kept them. The reason why they are kept in this research is to simulate real-life condition where exist possibility of faulty and unnecessary sensors. Also, the idea of research was to do as little as possible on data before making a predictive model.

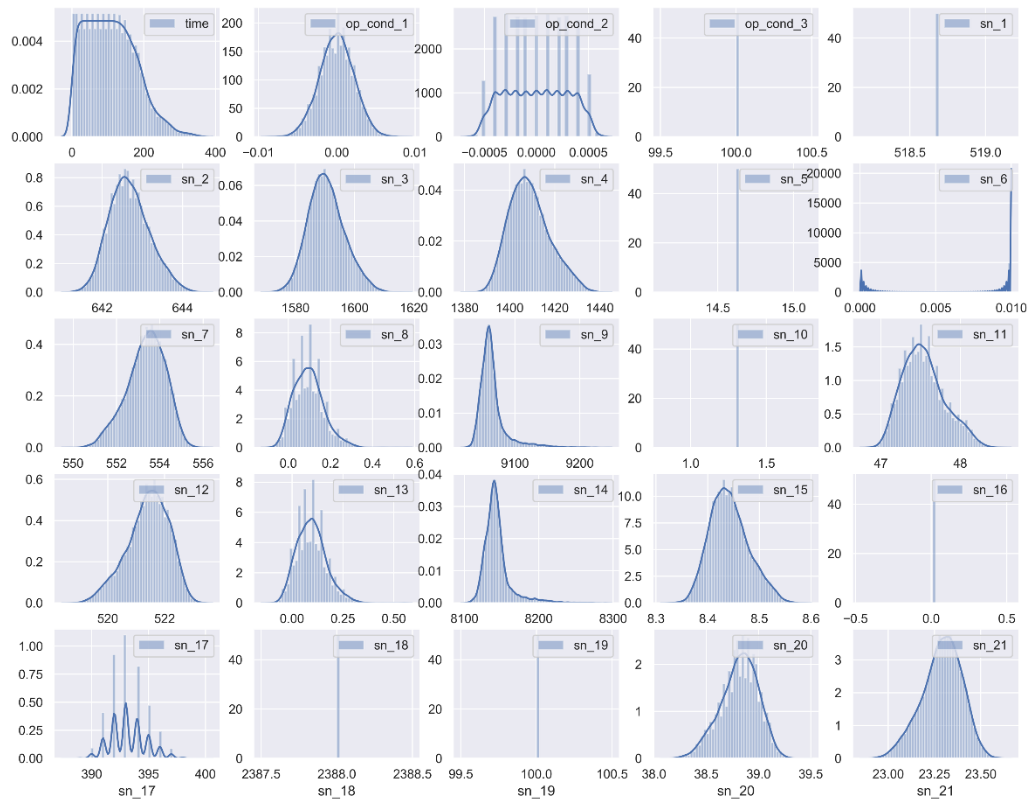

Figure 11 distribution of various data columns

Figure 11 shows the distribution of various data columns, and here we can see that all variables look like skewed Gaussian distribution, except for sensors mentioned before that do not change a lot. Furthermore, these observations hold when plotting all engines and when the plot is made for a specific engine.

In Figure 12, we can see a plot of how sensory data change over cycles. Each line in a plot represents a different engine, and each subplot represents a different sensor. They all finish at RUL 0, meaning that far-right on the x-axis is the failure of an engine. Furthermore, we can see how each engine does not have the same amount of cycles in the dataset. By the looks of data, we see that changes in of sensory values significantly change when approaching to the failure. Furthermore, we can see that sensory data does not follow the linear function of decreasing RUL over time. However, it stays continual for a long time and exponentially decreases as a failure of the machine is close.

That part is crucial for predicting RUL, as we will see how machine learning models have difficulties predicting when data is mostly flat and at the end have a sharp downturn.

Figure 13 shows a histogram of all 100 engines in train and test and number of timesteps that are given. The left images show a histogram for the train set, and the right image is for the test set. This is interesting to show because of window size approach that was applied in this thesis. All engines that do not have enough timesteps for selected window sizes are removed. In train set, that means that model will not be trained on them, while in the test set means that model will not be tested on them. That means that for larger window sizes we will have fewer data to test on. Furthermore, we can see that test dataset has, on average more timesteps than test dataset. That is partly because all train data finish with failure (RUL = 0) while test data does not. From this, we can see that window size cannot be larger than 200 because of the lack of test data. sizes just below 150 seams to be biggest to apply

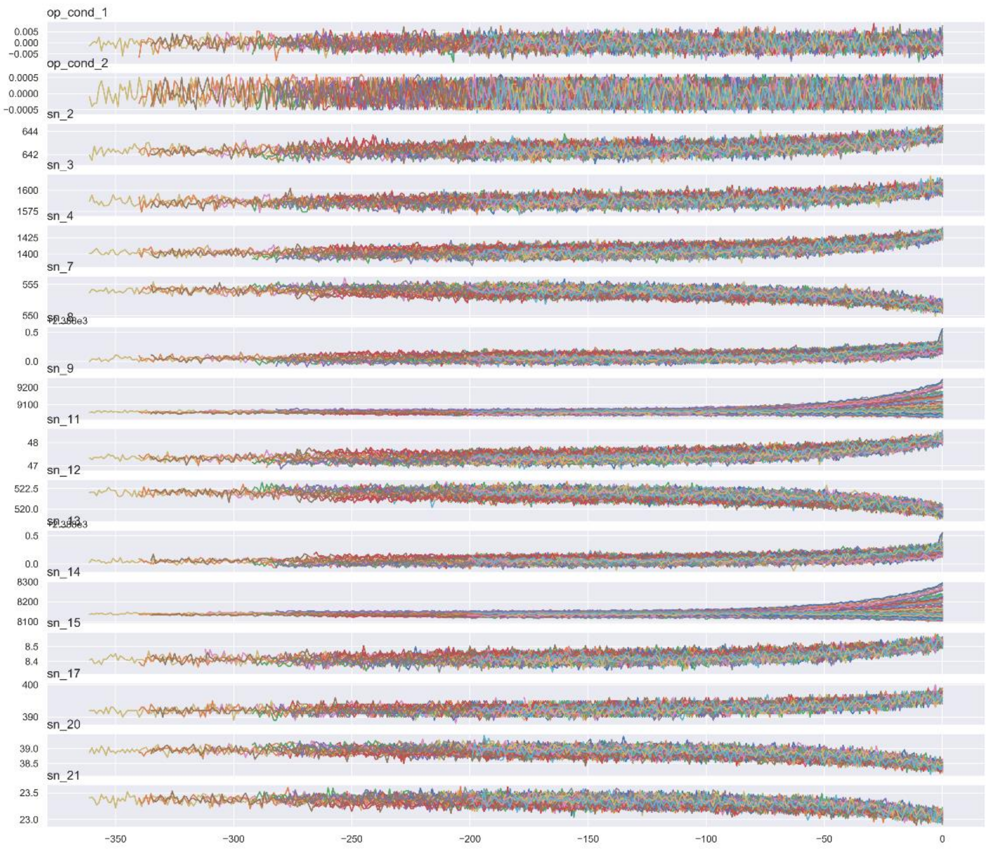

Figure 12 All engines time series, each line is a different engine, response x-axis is RUL with the rightmost point being the last cycle for all engines

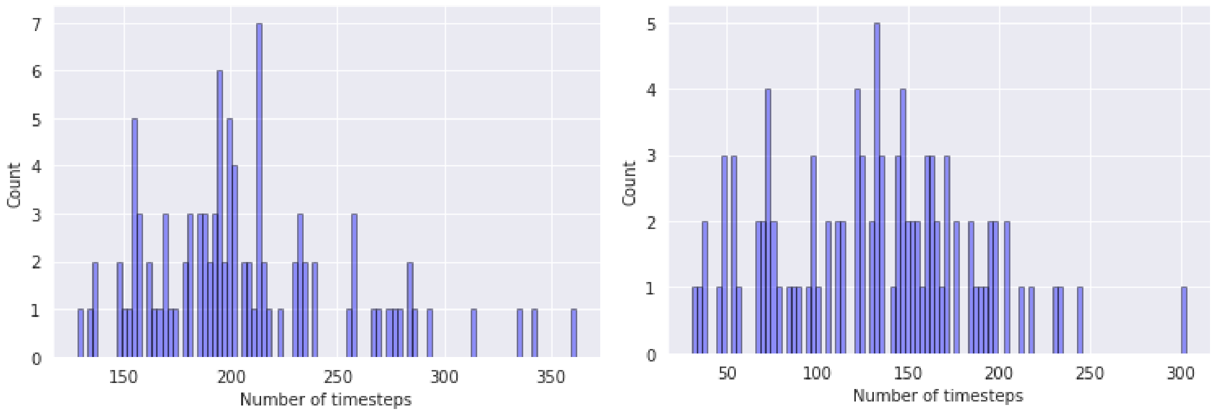

Figure 13 histograms for number of timesteps for each engine in train set (left) and test set (right)

5.3 Preprocessing

Regarding the preprocessing, the main idea is to keep sensory data untouched as much as possible. One thing that is applied to the data is Minmax Scaling to better fit values for Temporal convolutional networks [55]. Reason for that is different sensors and values which from one sensor can vary around 100 and for the other around 0.1. Those kinds of differences will not produce good results when using any type of neural networks. Formula from Minmax Scaling is

$$Xsc = \frac{X - min(X)}{max(X) - min(X)}$$

One thing that is also applied as preprocessing is changes in RUL target function. As RUL defines a number of leftover cycles, it should be linearly decreasing with time. However, in practical application, equipment does not degrade linear. As shown in Figure 14, the sensory values do not have significant changes in the green section. For that reason, in this thesis, we implement a piecewise linear function to represent RUL. That means that we select a threshold, usually, 125 for FD001 dataset as shown by [50] and [51], and all RULs that are larger than threshold equalizes with it to get a function as shown in Figure 14.

Furthermore, in processing, we have applied time window processing. In this time series-based problem, more information can be gathered by considering temporal dimension in comparison to taking data point from a single timestamp. In data preparation, we use a time window for the multivariate temporal information. We select the size of a window, and at each timestep past sensory data within the time window are taken from the feature vector, and used as the input for the network. The example is shown in Figure 15, there we have two images a) and b). In a) we can see sensory values over time and in b) is remaining useful life over time. Market with green is selected window of size 40, from 60 to 100 cycles. The RUL for that window is value in cycle 100, shown in image b) which is 92.

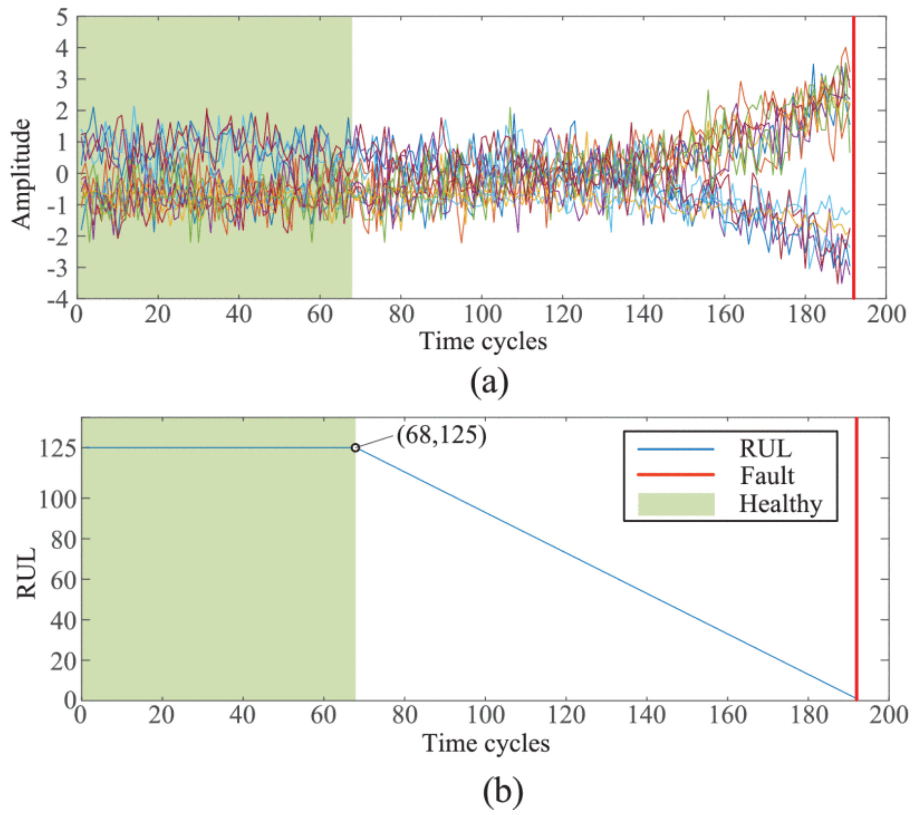

Figure 14 a) 14 sensory values over time b) piecewise linear RUL function

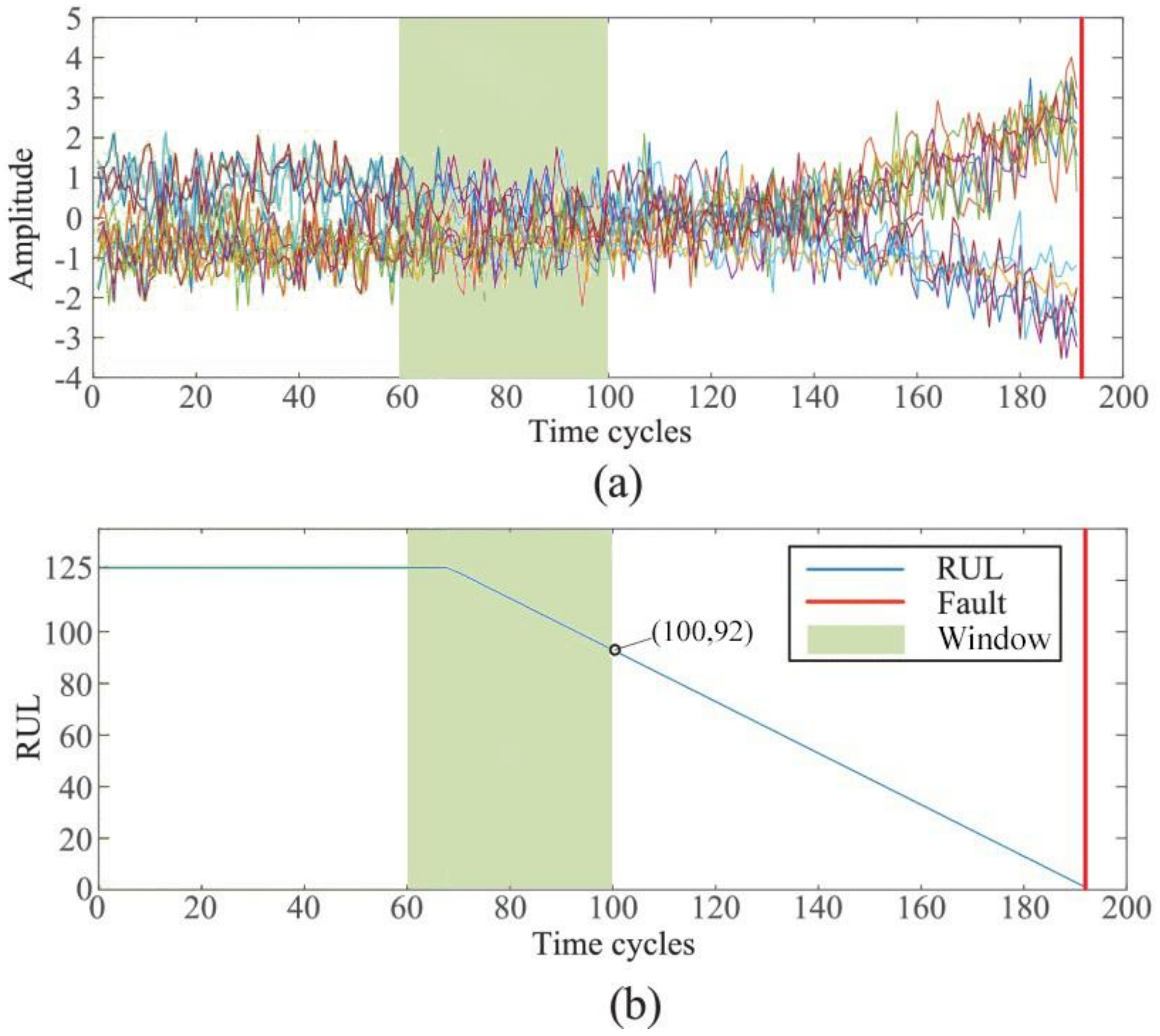

Figure 15 time window representation a) time window in sensory data b) expected RUL for the time window

6 EXPERIMENTS

Now, after everything abut our methods, and the data that we use is explained, we can star chapter about our experiments. In this section, we will explain the experiments that were conducted in this thesis. The first subsection is about the performance metrics that are used in this thesis. Next, we introduce the environment and framework that is used for experiments. The first experiment that is explained is our TCN fusion, which is followed by experiments related to autoencoders. In this section, we also explain how piecewise function and hyperparameters affect the results.

6.1 Performance metrics

First, we are going to start with the performance metrics that are used in this thesis. As the goal was to predict remaining useful life, we wanted to be as close to that number as possible, regardless if we were overdue the failure time by a couple of cycles or we predicted failure too early. The difficulty of predicting failure before it happens is that the machine could work more if we predict failure too early, and stopping it too soon could lead to unnecessary costs. In another case, if we do not anticipate on time, the machine will fail before we notice.
Furthermore, the performance metrics are crucial in the training of a model, with the incorrect metrics, the final product will not be as accurate as we thought it is on development [56]. In our case, we want to penalize higher predictions that are far and penalize less close predictions. Root mean square error (RMSE) is selected for metrics in our experiments for reasons mentioned before as well because it is the most popular benchmark method among other papers like [5],[11], [47] etc. The formula for RMSE is:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} d_i^2}$$

Where N denotes the number of samples and d denotes result in the ith iteration.

In a nutshell, RMSE penalizes more one big missed predicting than a few smaller ones. For example, if we have 10 predictions and each of our predictions is missed by 1, that will result in RMSE score of 1. In contrast, if we accurately predict 9 of 10 and miss one by the difference of 10, that will result in score 3,16. Even though when summing error differences in both cases, it is the same number. That shows how metrics favours consistency in predictions over high deviations. Also, making prediction smaller or larger than the truth does not have a different influence on this metric, which means that we would not know if our model is only predicting RUL smaller than it is. In comparison to other more straightforward metrics like percentage of accurately predicted RULs, which can be represented like 90% accuracy for predicting accurately 9 out of 10 predictions, RMSE score does not give us straight forward meaningful insight how accurately has model predicted. Also, worth to mention is that for this metric, the best score is 0. Nevertheless, as metrics for comparison of predicting models in a regression task, that for each prediction try to be as close as possible to the correct result, this method is more than adequate.

Another metric that we use is Mean Absolute Error (MAE). It measures the average magnitude of the errors in a set of predictions without considering direction. Meaning that it does not care if the prediction is early or late. Also, it does not penalize more predictions that are less accurate like RMSE. The formula for MAE:

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - x_j|$$

Here n is a number of predictions, y_j are predicted values and y_j are true values. This metric is used to simplify how accurate the model is. Unlike RMSE, MAE will give us a number which people intuitively can use to indicate how accurate the model is. For example, if MAE score is 10, that means that model on average misses time of failure by 10 timesteps, cycles or whatever the time metrics is.
Similar to MAE is Mean absolute percentage error (MAPE). It measures accuracy as a percentage and can be calculated as the average absolute error for each time period minus actual values divided by actual values. The formula for MAPE is:

$$MAPE = \frac{1}{n}\sum_{j=1}^{n}|\frac{y_j - x_j}{x_j}|$$

Where y_j are predicted values, x_j are actual values and n is number of predictions. This method is used to show error difference in percentage over time as other methods without absolute part would give us wrong impressions.

6.2 Default learning parameters

For experiments in this task, we use the same learning parameters in every training process of a learning model. Reason for that is to more focus on models itself and changes that we made in them, then on fine-tuning of learning parameters. We use a setup of maximum 100 epochs and with a batch size of 200 We have done a couple of experiments with bigger epoch size, and we have concluded that after 100 epochs, the model does not learn significantly. To make a more accurate comparison of models, we have decided that all should have the same learning parameter and stick to 100 epoch size. Furthermore, we implemented random sampling, so our model will not get the same sensory input in long sequences. For validation, we use 5% of the data. That number, as well as the batch size of 200, was selected due to the size of training data and small experiments where we have decided that this parameter fits best for our experiments. The learning process is done using RMSprop backpropagation algorithm to update the weights in the network as a similar setup was suggested by [5].

6.3 Environment and frameworks

For this project, we used Python 3, Jupyter environment on google Colab platform [3]. It is a free service offered by Google. The main advantage is that code is running on Google cloud, which has implemented GPU and TPU accelerators, which speeds up the training process. Furthermore, we use TCN library freely provided on GitHub [4]. The library has an implementation of the temporal convolutional network, which is described in previous chapters, and it has all necessary arguments that are described in Table 3. Furthermore, we have used free Python libraries TensorFlow[5] , Keras [6], scikit-learn[7], NumPy[8] and Pandas[9].

[3] https://colab.research.google.com/
[4] https://github.com/philipperemy/keras-tcn
[5] https://www.tensorflow.org/
[6] https://www.tensorflow.org/guide/keras
[7] https://scikit-learn.org/stable/
[8] https://numpy.org/
[9] https://pandas.pydata.org/

Name	Description
`nb_filters`	The number of filters to use in the convolutional layers. Would be similar to units for LSTM.
`kernel_size`	The size of the kernel to use in each convolutional layer.
`dilations`	A dilation list. Example is [1,2,4,8,16,32,64]
`nb_stacks`	The number of stacks of residual blocks to use.
`padding`	The padding to use in the convolutions. 'causal' for a causal network (as in the original implementation) and 'same' for a non-causal network.
`use_skip_connections`	If we want to add skip connections from input to each residual block.
`return_sequences`	Whether to return the last output in the output sequence or the full sequence.
`dropout_rate`	A number between 0 and 1. The fraction of the input units to drop.
`activation`	The activation used in the residual blocks o = activation(x + F(x)).
`kernel_initializer`	The initializer for the kernel weights matrix (Conv1D).
`use_batch_norm`	Whether to use batch normalization in the residual layers or not.
`kwargs`	Any other arguments for configuring parent class Layer. For example "name=str", Name of the model. Use unique names when using multiple TCN.

Table 3 TCN Library arguments

6.4 TCN Fusion

In order to capture different latent representations, experiment with TCN fusion is proposed. The idea is to make N numbers of TCNs all with varying sizes of window and then fuse them into one which will give the final result. As shown in Data exploration, sensory data have a significant difference when RUL is large and when RUL is small. For that reason, we are the opinion that one machine learning model would have difficulties to make predictions where there is such a significant difference in data. In our experiments, we have trained five TCNs. Each has two convolutional layers with dilations, the first layer has 12 filters and second has 6 after each layer dropout of 20% is applied. Those numbers are selected due to experiments made in hyperparameter tuning. After those two layers of TCN, there is a final layer that is a dense layer. Furthermore, the input is normalized as described in the previous chapter, and on expected RUL piecewise function is applied with a threshold of 137 for maximum RUL. The only difference between TCNs is that TCN1 has a window size of 10, TCN2 30, TCN3 50, TCN4 70 and TCN5 has window sizes of 90. Those sizes are selected as it was proposed by [11]. Each of those 5 TCNs is trained separately on the task of predicting RUL. After training is completed and best models are selected, from each model is stripped off a final dense layer, and its latent representations are connected to one. The first test was to connect them to one dense layer. The second test was to stack them and make one more TCN layer with a dense layer. A simple illustration is shown in Figure 16 where on the left is a variation with one more TCN layer and dense layer, while on the right is a variation with the final dense layer. Furthermore, the numbers below represent the input window sizes of each TCN.

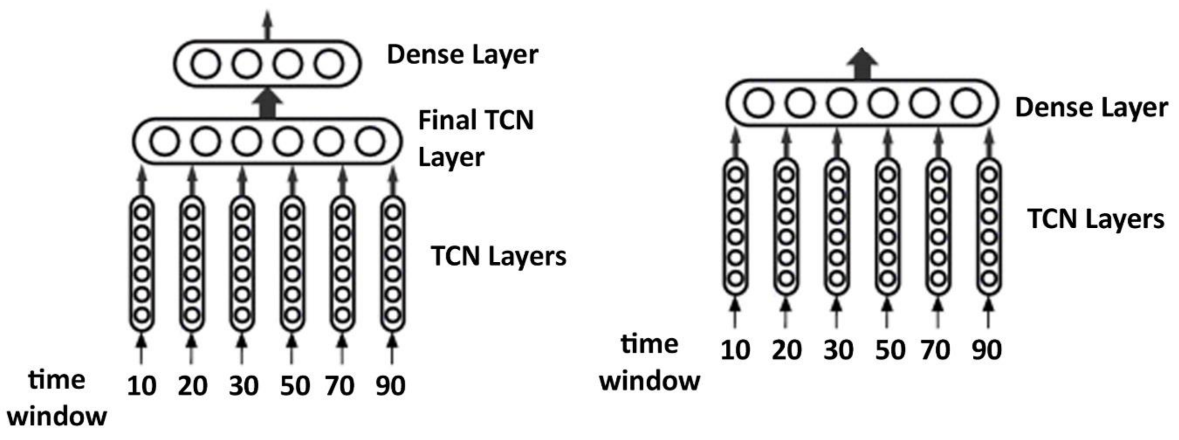

Figure 16 Temporal convolutional network fusion, on the left with final TCN layer and dense layer, on right only with a dense layer

6.5 Autoencoders

During the experimentation process, we have also tested various iterations of autoencoders. First, we have used autoencoders as smoothing function for data processing, then we have tested autoencoders with TCN, and finally, we have tested if autoencoders improve results on LSTM.

6.5.1 Noise reduction Autoencoder

First, we did a test using autoencoder as a smooth function to reduce noise in sensory data. The idea was to find the best neural network autoencoder architecture that will solve this problem. In this process, we take input, and we want to replicate that input, so our input data is the same as our prediction data. The input data is a one-dimensional array of each timestep in the dataset. The autoencoder that we use for it is made of the six-layer neural network, three for the encoder and three for the decoder. The first layer of the encoder is the size of 32, second has 16 and bottleneck has a size of 8, the decoder is mirrored representation of 16 then 32 and final layer with size same as the number of input features. All activation functions are ReLu, except for the last layer where the sigmoid function is used because of its ability to produce results that are positive and negative numbers. Those numbers were selected by making small hyperparameter tunning and using suggestions made by [57] and [51].

6.5.2 TCN and autoencoder

For this experiment, we have used TCN with two layers. The first layer of TCN with twelve filters and dilations (1,2,4,8,16,32), after which dropout of 20% is applied, the second layer has six filters, same dilations and dropout as the previous layer, and at the end one dense layer with a linear activation function. Learning is done using RMSprop backpropagation function, where the loss function is RMSE, it is done in a series of 100 epochs with a batch size of 200. That set up as used as a standard TCN model in our experiments which will be used later on, the number were selected by examining the papers [7], [58]. The dataset that we are testing on is FD001. For autoencoder, we have used previously mentioned noise reduction autoencoder form whom we have striped decoder part and only used encoder. We have also tested different output sizes of encoder part, as well as using full autoencoder for a noise reduction.

6.5.3 LSTM and autoencoder

As previously shown, smoothing autoencoders or dimension reduction autoencoders do not affect results when performed on TCN, we have experimented with autoencoders and LSTM. In this experiment we have compared the base LSTM which has two layers, the first layer has 100 units and the second layer has 50 units after each layer dropout of 20% is applied, the final layer is a dense layer. The input data is made using a window size of 50 timesteps. The autoencoder uses RMSprop backpropagation algorithm and RMSE as the loss function. We use a batch size of 200 with random sampling, a maximum of 150 epoch with an early stop if the model does not improve in the next ten epochs. For all these experiments we use same LSTM architecture. The autoencoder that we use is the same architecture as the one described in TCN autoencoder experiments.

6.6 Piecewise remaining useful life function

In this thesis, we have implemented a piecewise function on the remaining useful life to enhance results. As described in section

Preprocessing, sensory data at earlier stages are different than in the late stages. That causes a problem with predicting large remaining useful life when failure is not going to happen in the near future. We have done an experiment to compare how TCN is performing with and without piecewise RUL. For this experiment we have used two same TCNs, each has a first layer of TCN with 12 filters and dilations (1,2,4,8,16,32), after which dropout of 20% is applied, the second layer has 6 filters, same dilations and dropout as the previous layer, and at the end one dense layer with a linear activation function. Learning is done using RMSprop backpropagation function, where the loss function is RMSE, it is done in a series of 100 epochs with a batch size of 200. The dataset that we are testing on is FD001.

6.7 Hyperparameter tuning

Deep learning models, such as neural networks, temporal convolutional networks or long-short term memory models, are strongly affected by hyperparameters, for that reason, it is crucial to select hyperparameters appropriately. The main hyperparameters of the temporal convolutional network are described in Table 3. In this thesis, we have explored a couple of the most significant ones. In this thesis, we have explored a couple of the most significant ones. We have tested different window sizes (10,30,50,70,90,120,137) as suggested by various papers. Next, we have tested if the increase in the number of layers affects the results. The tests were conducted on one, two, three and four layers. Finlay, we have tested how kernel size affects the results. The kernel was used to make convolution between numbers in TCN before adding weights and biases. Sizes that we tested were 2, 3, 9 and 25. the idea was to see how the increase in kernel size affects the results.

7 RESULTS

7.1 Noise reduction autoencoder

In this test, we have used autoencoder to smooth sensory data. How smoothed data looks like is shown in Figure 17. The orange lines represent smoothed data, and blues are original. The data is from engine 3 data FD001; it includes all sensors, including ones that do not change over time. From the plots, we can see that most of a data is smoothed and noise is reduced. Sensors s1, s5, s6, s10, s16, s18, s19 do not change over time, and also those are the areas where autoencoder made most errors. Namely, autoencoder inserted some noise in it instead of keeping the line straight, or not changing the number over time. The test that we did was testing different topologies with more layers, but they did not produce any improvement to autoencoder. Only a smaller number of layers has produced worse results. For one layer R2 distance accuracy was 0,61, for two layers 0,86 and the for selected the layered architecture achieved score of 0,97. Table of scores can be found in Table 4.

Number of layers	R2 distance score
1	0,6148
2	0,8628
3	0,9762
4	0,9776
5	0,9745

Table 4 R2 distance score of autoencoders based on numbers of layers

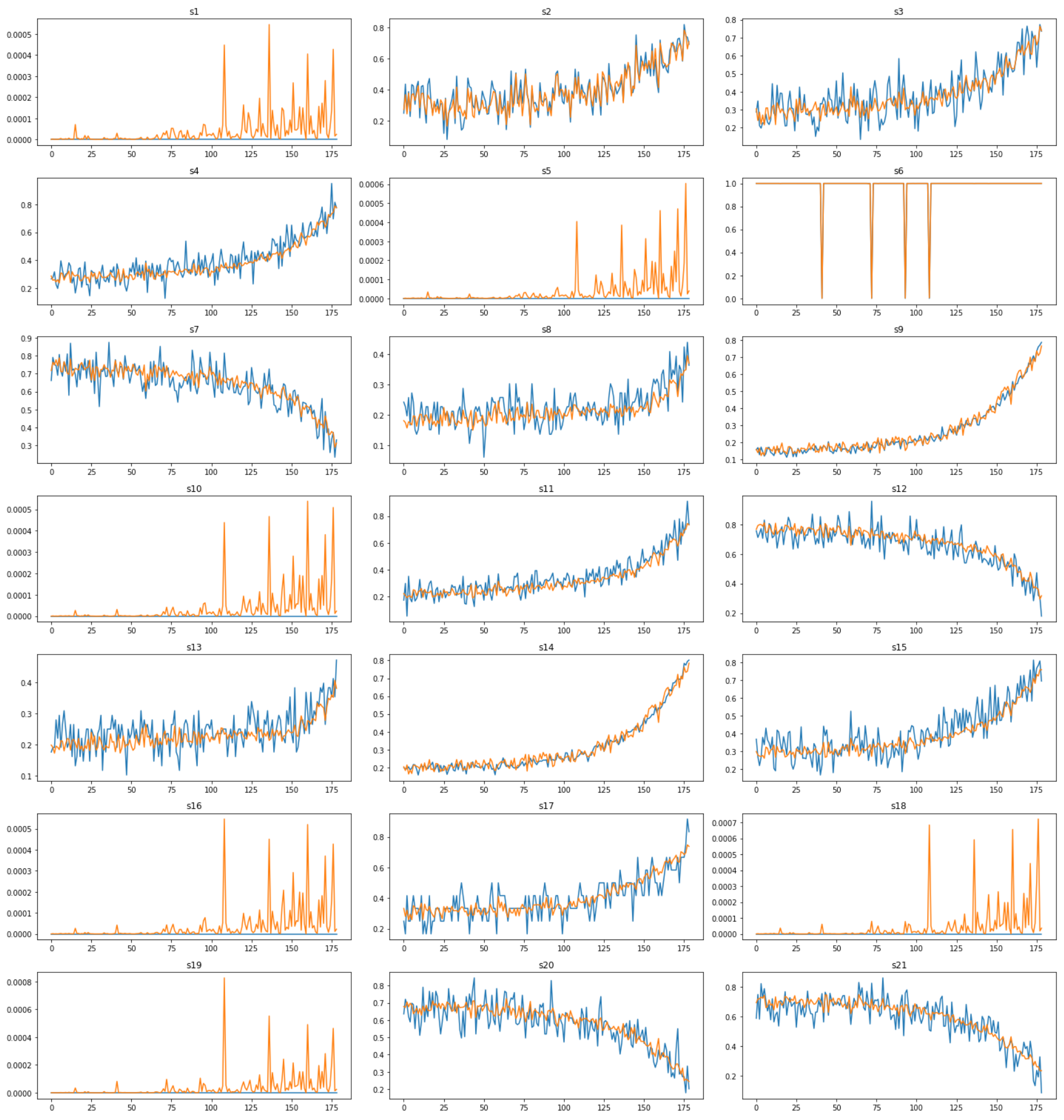

Figure 17 Results of smoothing data with autoencoder on engine 3. Orange is smoothed data, and blue is original

7.2 TCN and Autoencoder

In this experiment, we have used the previously mentioned architecture of autoencoder, use it as feature reduction and combine it with TCN. In Figure 18, we can see the results of autoencoder in combination with TCN. In that figure, Y-axis represents score in RMSE, and x-axis represents a number of features that came out of autoencoder, 0 represents autoencoder that is used for smoothing. In that chart, we can see the test results of our autoencoder when is not used only for smoothing. We have tried several different bottleneck sizes to see how they differ. As we can see, they do not much make a difference if we have 7,8 or 9 features that are reduced from the original 25 features. Best performance is done by smoothing, but still in comparison to not using it at all it does not contribute performance. We could conclude that TCNs are resistant to sensory noise that is in the dataset. Probably the reason for that is that TCN has enough robustness to cope with a problem of noise in a data. Furthermore, TCN can easily handle a larger number of input data then our 25 features as it is usually used for much more like in [7]. So decreasing the amount of input data does not help a lot.

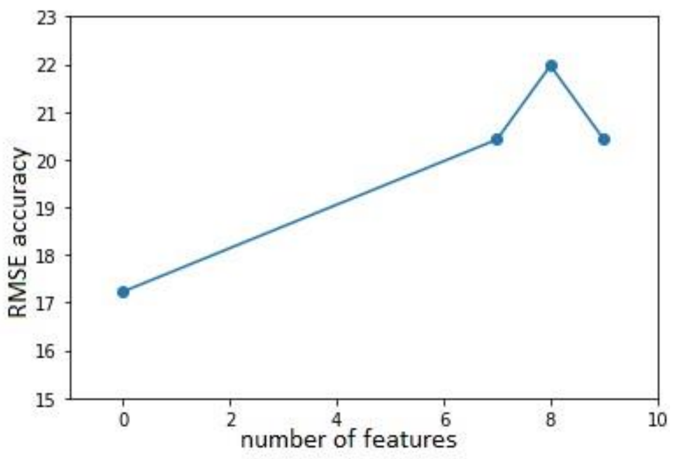

Figure 18 TCN and autoencoder results, the y-axis, is RSMS accuracy, the x-axis is number features, 0 is for smoothing autoencoder

7.2.1 LSTM and autoencoder

In Table 5, we can see the results of our tests conducted by combining LSTM and autoencoder.. AE Smooth represents an experiment where have we used autoencoder to smooth all 21 signals in propose of noise reduction. AE 21to9 stands for autoencoder that makes dimensionality reduction from 21 sensory data to 9. AE 14to9 is for LSTM whose input is encoded data of 14 selected sensors reduced to 9. The idea behind the latter is to remove those sensory data on which autoencoder performs poorly, or better to say sensory data that do not provide valuable information. AE 14 smooth is a noise-reduced version of 14 selected sensory inputs. By the results shown in Table 5, we can see that none of the autoencoder methods has provided improvement in accuracy of predicting remaining useful life. Similar to TCN, we can see that reduction of dimensionality does not improve results; on the contrary, it makes it worse. Again, the reason can be similar to TCN as they both use a large number of neurons to make predictions. Also, LSTM uses blocks of neurons so it ca extract valuable information even from sort to say irrelevant input.

Model	RMSE Score
LSTM	17.59
AE Smooth	19.16.
AE 21to9	18.75
AE 14to9	19.60
AE 14 smooth	18.41

Table 5 Results of experiments with LSTM and autoencoder

7.3 Piecewise remaining useful life function

The final accuracy of TCN with piecewise RUL measured in RMSE is 10.23, while without piecewise RMSE score is 12.24, as shown in Figure 19. Note that piecewise function is used only on training data for one model while both models are tested on the same non-piecewise data.

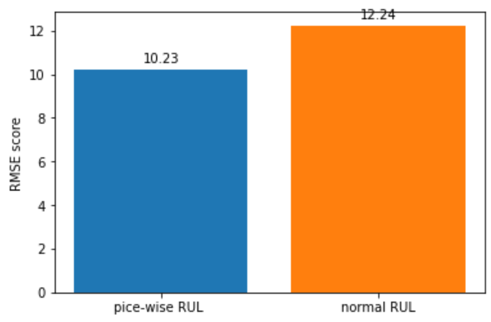

Figure 19 piecewise RUL vs normal RUL in RMSE accuracy

In Figure 20, we can see the results of a model that is trained on piecewise RUL function. The orange bar is ground truth RUL while the blue bar is estimated, the left side shows the smallest RUL while the right is largest. When predicting the RUL to the model is given only N number of timesteps of sensory data, where N is the size of the window and appropriate RUL number for that window. Also, the fact that they are arranged from smallest to largest is not something that is in the data, that is only for visual representation. In compression of Figure 20 with Figure 21, which represents a model that is trained without piecewise RUL, we can see that in the second plot, the estimation of larger RUL is more accurate. At the same time, it lacks down in the estimation of small and medium-size RUL. The model with piecewise RUL performs worst where RUL is high, obviously because in training set it has never seen such data or if it was, that kind of data was rare. That lack of higher data compensates with high accuracy on the rest of the RUL estimation.

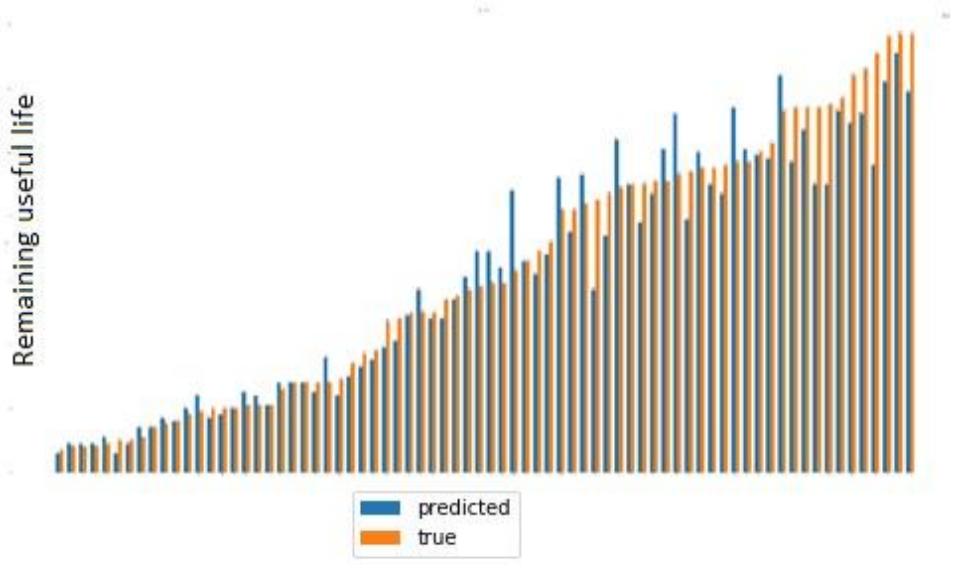

Figure 20 Temporal convolutional network, RUL estimation with piecewise function

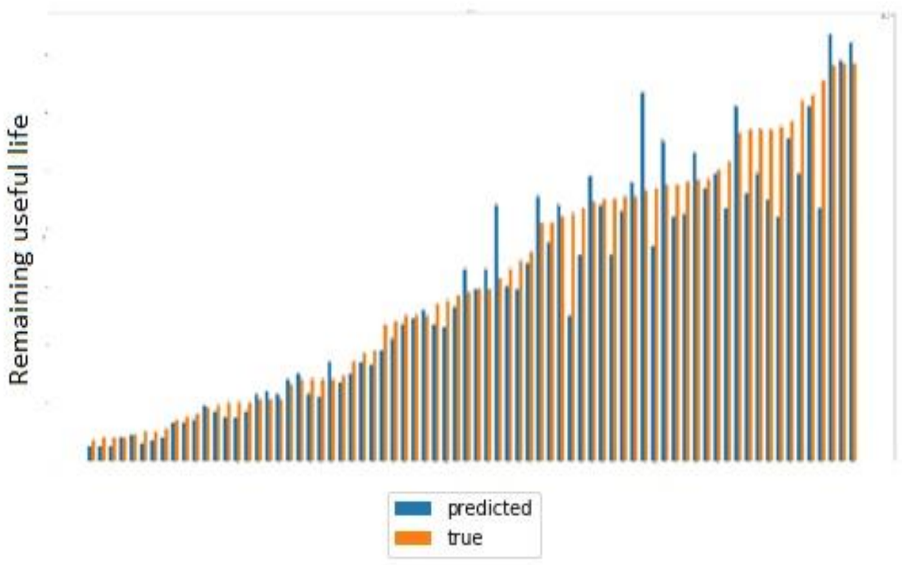

Figure 21 Temporal convolutional network, RUL estimation without piecewise

To better explain the difference between two scores in Figure 22 and Figure 23. Figure 22 is a graph that shows the average error difference for both models. The error is shown as a mean absolute percentage error between predicted and true value. In the graph, the y-axis shows the mean absolute difference in cycles of predicted and true value represented in percentage, while the x-axis represents RUL. Figure 23 shows a graph of mean absolute error of both models, similar to Figure 22, x-axis shows RUL while the y-axis shows the mean absolute error of a model. By looking at Figure 22, we can see how piecewise function has affected the predictions. The predictions are much more accurate when RUL is smaller in comparison to the model that was not trained on piecewise data.
Furthermore, we can see how by the end of the graph, when RUL becomes higher, the percentage of those to models become more similar. As shown in Figure 23, we can see how the error of the piecewise model is having almost constant error until RUL becomes as high as 95 and then exponentially starts to grow. Also, that graph shows the error of both models and how inaccurate in terms of cycles they become as predicting RUL get a wider range. However, in terms of percentage error in the prediction, that error for a model trained on piecewise function stays more constant.

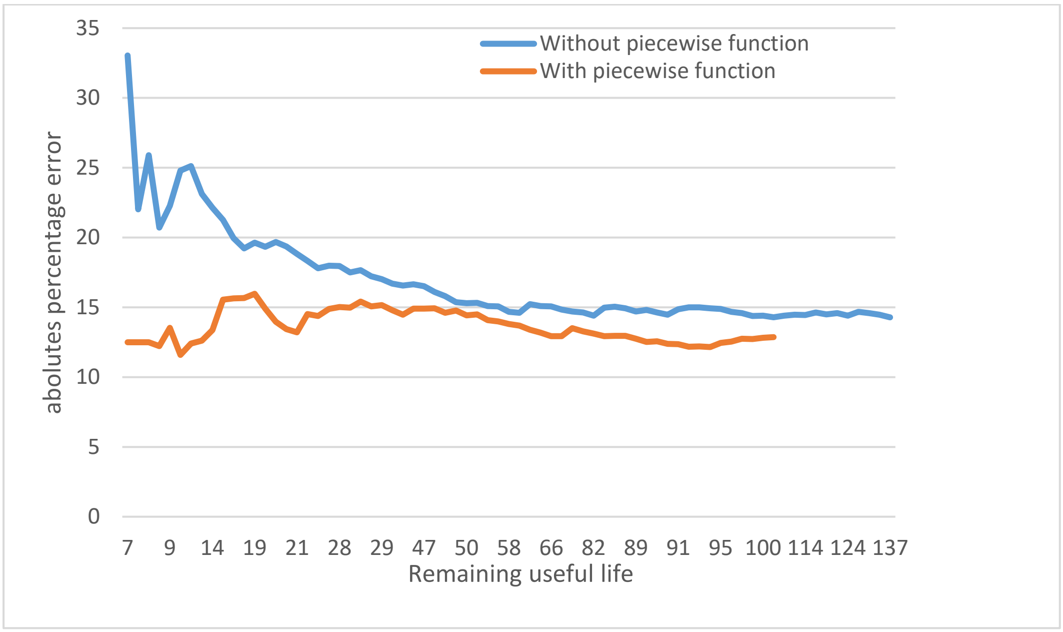

Figure 22 Mean absolute percentage error between models that are trained with and without piecewise function.

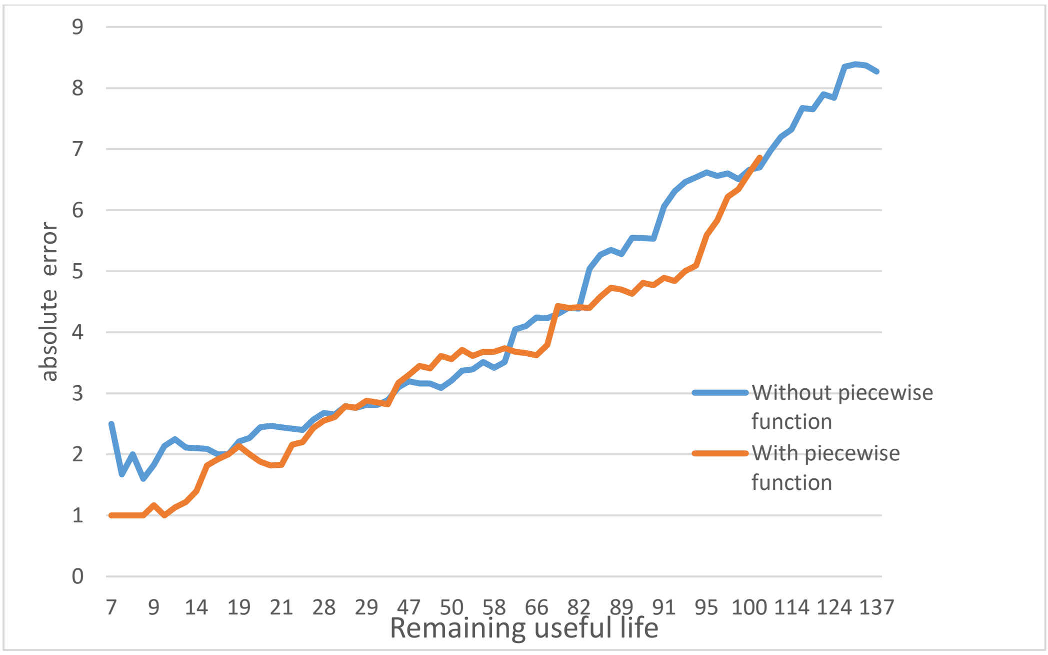

Figure 23 Mean absolute error between models that are trained with and without piecewise function.

7.3.1 Window size

The first one is how window size affects accuracy. Window size is not mentioned in Table 3 as a parameter because it is more related to preprocessing of data. In **Pogreška! Izvor reference nije pronađen.**, the result of different window sizes is s hown. On the y-axis is shown RMSE accuracy while the x-axis represents window size. We can see that TCN performs better with larger window size. Problem with taking too large window sizes is that in this dataset, test data has a variable number of samples before ground truth RUL. The smallest one has a size of 30 steps while the largest has a size of 137 timesteps.

When we take a larger window size, we are not able to test on smaller sizes, so those are not taken into consideration. For that reason, window size between 90 and 120 performs best.

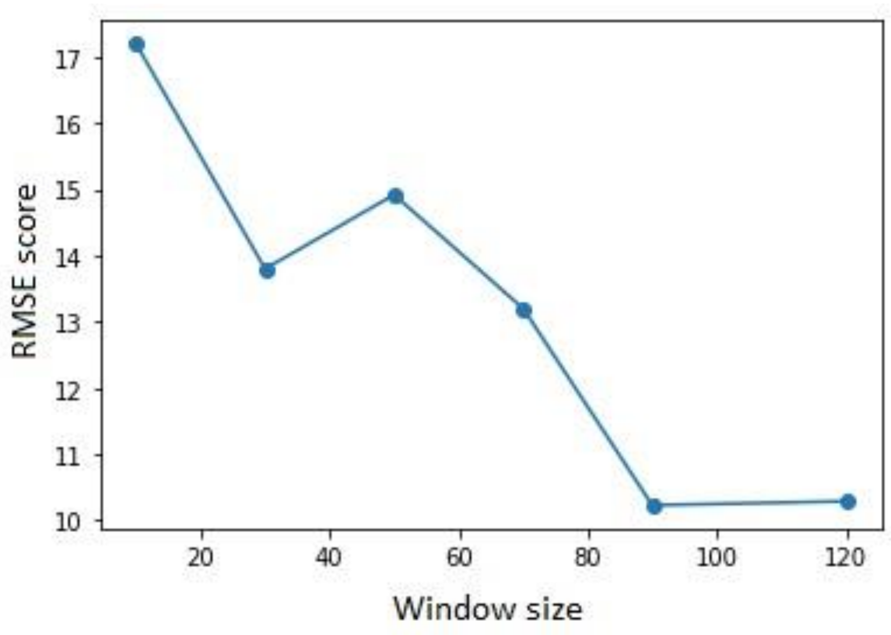

Figure 24 Results of different window sizes on TCN performance y-axis represent RMSE score, and x-axis represents window size

7.3.2 Number of layers

Next experiment that was done in this thesis is exploring how a number of layers affect results. By this number of layers, we do not mean on layers created by dilation method. In Figure 25 are shown results of TCN networks with 1,2,3 and 4 layers. Those are selected as a minimum number of layers, as we will later see, all number above three layers do not provide us with significant. Improvement. Each layer is built with filter size half of size of its predecessor, starting with 32 filters as it was suggested by [7]. Like in the previous chart, the y-axis represents the RMSE score, while the x-axis represents a number of layers. By results, we can see that 2 or 3 layers perform best. For most of the further testing, it was selected two layers because of similar to three-layer TCN best performance, and also because fewer layers mean less training time.

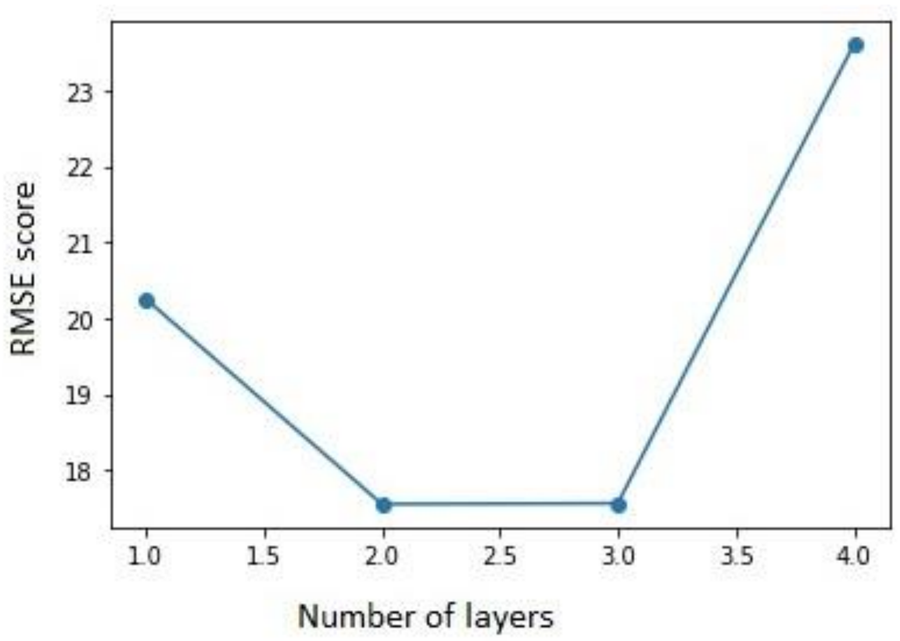

Figure 25 Effect of a number of layers on TCN performance. Y-axis represents the RMSE score, and x-axis represents the number of layers

7.3.3 Kernel size

One more experiment that is done with TCN hyperparameters is exploring how different kernel sizes affect speed and results. In Figure 26, on the right, we can see on the y-axis RMSE score and on the x-axis number of filters that are used. Of course, we cannot use one filter, so it starts at 2, followed by 3, 9 and 25. By results, we can see that they are pretty close to each other. We do not have high fluctuations as with changing window sizes. Furthermore, the left in Figure 26 represents the training speed of each epoch measured in seconds. It is a visible constant trend of increasing training time as a number of kernels increase. Worth to mention is that we usually train for 150 epochs so to calculate complete training time that speed number needs to be multiplied by 150.

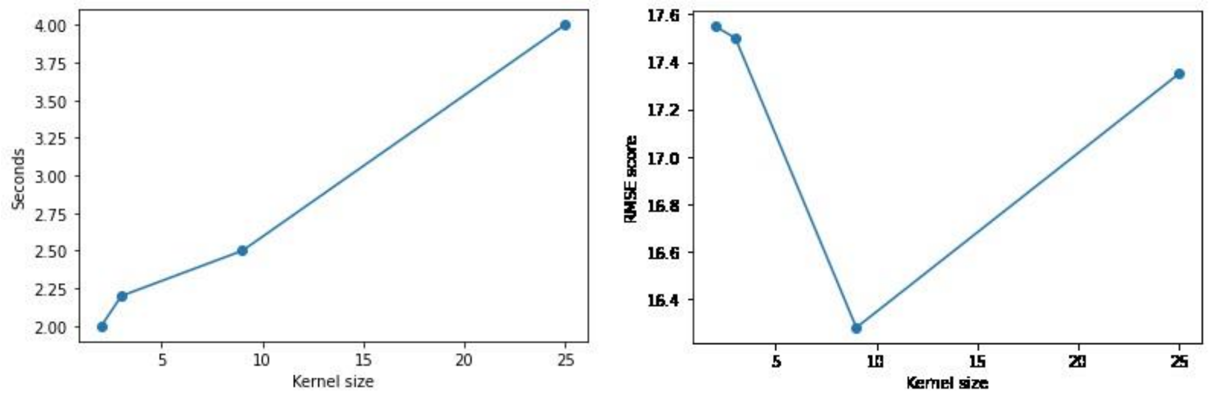

Figure 26 Effects of kernel size left and speed of training per epoch right on TCN performance.

7.4 Comparison to state-of-the-art methods

In this section, we will compare the results of our method to the other state-of-the-art methods that deal with the task of predicting the remaining useful life on C-MAPSS dataset. Unlike other proposals, in this thesis, the focus was only on FD001 subset of C-MAPSSS dataset, and that is where are we going to make a comparison. Also, the selected metrics for this is RMSE because all papers present their results in that metric. In Table 6, we can see the result in a comparison of the proposed method with the latest released papers on C-MAPSS dataset. The selected scores were taken from the papers where methods were presented, and we did not try to recreate those methods. All these methods are explained in section Literature research, and as we can see, all of them are based on neural networks. Furthermore, for this comparison, we have selected only papers with best results. As we can see, our method of TCN-fusion outperforms all other methods in this task. In comparison, we have included a version without piecewise RUL, which significantly improves results, of shown methods only GRU-ED [41] method does not use that function. The proposed method can achieve the prognostic results comparable to the state-of-the-art methods with simple architecture and little to no preprocessing and without prior knowledge.

Name	RMSE score on FD001
LSTM-Fusion [11]	11.18
MS-DCNN [6]	11.44
Li-DAG [50]	11.96
GRU-ED [41]	12.45
DCNN [5]	12.61
TCN-Fusion (Proposed method)	**10.23**
TCN-Fusion Without piecewise	12.24

Table 6 Performance comparison of the proposed method and the latest released papers on the C-MAPSS dataset

8 DISCUSSION

The objective of this research was to develop a model and test various methods to check, which is best in the task of predicting the remaining useful life of the turbofan engine. The dataset that we use for this is NASAs C-MAPSS dataset that contains various sensory data of simulated turbofan engines. Most of the experiments are based on LSTM and TCN networks. We have tried to implement an autoencoder, which was successful in combination with some other machine learning models. In our case, the autoencoder was not helpful. We have also tried to smooth our data to reduce noise with autoencoders, but it did not improve predictions. We can conclude that the robustness of LSTM and TCN is resistant to noise in sensory data. Also, its robustness is capable of ignoring irrelevant inputs which were reduced in first experiments with an autoencoder.

Furthermore, by experimenting with hyperparameters of TCN, we concluded that having larger window sizes give us the best results. As represented in WaveNet paper [7] TCN with dilations is successfully used on an even larger sequence of data, so making better result with larger window size was kind of expected result. The main challenge in this task for us was the difference in data variations when failure is not going to happen soon (large RUL) and data when failure is close (small RUL). When RUL is large, changes in data over time are minor and determining exactly how long till failure was challenging. As it was implemented in other works that deal with RUL task, we have implemented piecewise function to flatten large RULs and to help TCN deal with data variations more easily. That method provided us with much success. Furthermore, to cope with the problem of the difference between high and low RUL, we have implemented fusion as it was presented in the paper [11]. The idea behind that is to use multiple TCNs with different windows sizes so each TCN can capture different temporal information. That method provided us with state-of-the-art results in the task of predicting the remaining useful life. When we compare the result of only one TCN with large window sizes and multiple TCNs combined with fusion give us similar results. The only difference is training time, in TCN fusion we have to train five separate TCNs which takes much more time in comparison to training only one.

Retrospectively looking, this research certainly can be better. To improve the research, for example, better hyperparameter tunning can be done. In our analysis, we were restricted by time and computing limitations. Because of that, some decisions had to be made, for example, testing kernel size. In an ideal world, we would test every kernel size, but in this case, we were interested in the increase in size would improve results, for that reason we have tested numbers 2,3,9 and 25. Just a few smaller number and one much larger. Something similar was with layer size experiment. We have tested several layers until the size was too large. There is a possibility that using 10 or 9 layers could make better results than our two-layer architecture. There are several similar examples of small testing or selecting some parameter numbers that are not necessarily the best but presented as most suitable by other proposals.

Nevertheless, the results achieved using TCN network outperform other known methods that cope with this task. When we look at the results, they are not 100% accurate in prediction. By that, this method does not solve the problem of predicting the remaining useful life and has room for a lot of improvement, which is described in the next chapter.

9 CONCLUSION AND FUTURE WORK

We will start this section by answering our research questions mentioned in the Introduction section.

1. *Is it possible to use temporal convolutional networks to outperform other known methods such as LSTM in the task of predicting remaining useful life?*

By experimenting with temporal convolutional neural networks with fusion and without fusion, we have proven that they provide better results on C-MAPSS dataset then state-of-the-art methods.

2. *How autoencoders affect results in the task of predicting remaining useful life when combining with the temporal convolutional network or with long-short term memory models?*

By experimenting with autoencoder, we have concluded that in combination with TCN or LSTM, they do not provide any benefit in the task of predicting remaining useful life

3. *How different parameters of temporal convolutional networks affect results in the task of predicting remaining useful life?*

To answer this question, we have a separate results section.

4. *How the implementation of fusion, e.g. combining different models with different window sizes into one affect prediction of remaining useful life?*

Implementing fusion on the temporal convolutional network has improved results and gave us even better results than just TCN alone.

In this thesis, we have shown that applying temporal convolutional networks on the problem of predicting remaining useful life provide us with excellent results and outperforms state-of-the-art methods in this field. Furthermore, implementing fusion has improved results even more and gave us even better results than just TCN alone. We have conducted several experiments with different parameters of TCN and represent how they affect the prediction. Also, by experimenting with autoencoder, we have concluded that in combination with TCN or LSTM, they do not provide any benefit.

The area of remaining useful life prognostics holds many opportunities for continuing research beyond the scope of this thesis. Several of such proposal are mentioned below in this chapter.

Use two separate TCNs for large RUL and small RUL. The sensory data have a significant difference when the machine is near its failure and when it is not. Meaning that data starts exponentially increase or decrees as approaches failure and remaining useful life only decreases linearly with each cycle. The idea behind this one is to use two separate TCNs which eventually will fuse together. The first temporal convolutional network should be trained on data where the remaining useful life is significant, and the second will be trained where RUL is small. For example, first TCN let us call it TCN large and second TCN small, TCN large will be trained only on data where result RUL is above 30, and TCN small should be trained on data where RUL is below and equal to 30. Here we need to intrude a threshold that will divide a large RULs from small ones. Also, it could be multiple TCNs, for different sizes of RUL, not just small and large.

Use big window size and fill rest with zeros. As shown in results TCNs, perform best when the window size is large. Also, not all test inputs are of the same size. When using larger window sizes, we usually ignore test cases where input is smaller than our window sizes. One idea the is notable for testing is to take a large window size, around 120 and fill rest of input with zeros, or replicate last few timesteps. The idea comes from computer vision, where the common way to make them the same size input is to fill the rest of the image (input) with black pixels or just to stretch border pixels.

Test the proposed method on other datasets FD002, FD003 and FD004. In this thesis, we have only experimented with one subset of C-MAPSS dataset, the FD001 dataset. Dataset FD003 is quite similar to FD001, unlike FD002 and FD004, that are more complex. In them, the operation condition changes as well as failure conditions, which makes them harder to predict. That would make them exciting datasets to test the proposed method and compare them to current state-of-the-art methods.

Perform a grid search to improve hyperparameter optimization. This was not done due to lack of time for this project. To fully optimize the proposed method, some sort of hyperparameter optimization needs to be done. One popular method is to perform a grid search. That would be to select a couple of values for each parameter and make permutations to get the best results. Even though the current result gives us state-of-the-art results, the TCNs is not optimized, and for sure, it can perform even better.

10 REFERENCES

[1] J. Lee, F. Wu, W. Zhao, M. Ghaffari, L. Liao, and D. Siegel, “Prognostics and health management design for rotary machinery systems - Reviews, methodology and applications,” *Mech. Syst. Signal Process.*, vol. 42, no. 1–2, pp. 314–334, Jan. 2014, doi: 10.1016/j.ymssp.2013.06.004.

[2] Y. Lei, N. Li, L. Guo, N. Li, T. Yan, and J. Lin, “Machinery health prognostics: A systematic review from data acquisition to RUL prediction,” *Mechanical Systems and Signal Processing*, vol. 104. Academic Press, pp. 799–834, 01-May-2018, doi: 10.1016/j.ymssp.2017.11.016.

[3] S. Zheng, K. Ristovski, A. Farahat, and C. Gupta, “Long Short-Term Memory Network for Remaining Useful Life estimation,” *2017 IEEE Int. Conf. Progn. Heal. Manag. ICPHM 2017*, pp. 88–95, 2017, doi: 10.1109/ICPHM.2017.7998311.

[4] C. S. Hsu and J. R. Jiang, “Remaining useful life estimation using long short-term memory deep learning,” in *Proceedings of 4th IEEE International Conference on Applied System Innovation 2018, ICASI 2018*, 2018, pp. 58–61, doi: 10.1109/ICASI.2018.8394326.

[5] X. Li, Q. Ding, and J. Q. Sun, “Remaining useful life estimation in prognostics using deep convolution neural networks,” *Reliab. Eng. Syst. Saf.*, vol. 172, no. June 2017, pp. 1–11, 2018, doi: 10.1016/j.ress.2017.11.021.

[6] H. Li, W. Zhao, Y. Zhang, and E. Zio, “Remaining useful life prediction using multi-scale deep convolutional neural network,” *Appl. Soft Comput. J.*, vol. 89, p. 106113, 2020, doi: 10.1016/j.asoc.2020.106113.

[7] A. van den Oord *et al.*, “WaveNet: A Generative Model for Raw Audio,” pp. 1–15, 2016.

[8] Y. Liu, D. K. Frederick, J. A. Decastro, J. S. Litt, and W. W. Chan, “User’s Guide for the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS),” *Nasa/Tm*, vol. 2012–21743, no. March, pp. 1–40, 2012, doi: 10.1093/ilar/ilv006.

[9] R. J. Murry and B. F. Mitchell, “Cost savings from a practical predictive-maintenance program,” 2002, pp. 206–209, doi: 10.1109/rams.1994.291109.

[10] E. Ozkat, “THE COMPARISON OF MACHINE LEARNING ALGORITHMS IN ESTIMATION OF THE COMPARISON OF MACHINE LEARNING ALGORITHMS IN,” *Conf. IXth Int. Maint. Technol. Congr. Exhib. Pamukkale Univ. Denizli, Turkey*, no. September, 2019.

[11] Y. Zhang, P. Hutchinson, N. A. J. Lieven, and J. Nunez-Yanez, “Remaining Useful Life Estimation Using Long Short-Term Memory Neural Networks and Deep Fusion,” *IEEE Access*, vol. 8, pp. 19033–19045, Jan. 2020, doi: 10.1109/access.2020.2966827.

[12] X. S. Si, W. Wang, C. H. Hu, and D. H. Zhou, “Remaining useful life estimation - A review on the statistical data driven approaches,” *European Journal of Operational Research*, vol. 213, no. 1. Elsevier B.V., pp. 1–14, 16-Aug-2011, doi: 10.1016/j.ejor.2010.11.018.

[13] M. V. Cocquempot, M. A. Grall, M. P. Weber, and M. A. Barros, “Automatique-Productique John Jairo MARTINEZ MOLINA Christophe BERENGUER GIPSA-Lab l ’ École Doctorale EEATS M . Olivier SENAME M . John Jairo MARTINEZ MOLINA M . Christophe BERENGUER,” 2016.

[14] P. Paris and F. Erdogan, “A critical analysis of crack propagation laws,” *J. Fluids Eng. Trans. ASME*, vol. 85, no. 4, pp. 528–533, Dec. 1963, doi: 10.1115/1.3656900.

[15] M. Jouin, R. Gouriveau, D. Hissel, M. C. Péra, and N. Zerhouni, “Degradations analysis and aging modeling for health assessment and prognostics of PEMFC,” *Reliab. Eng. Syst. Saf.*, vol. 148, pp. 78–95, Apr. 2016, doi: 10.1016/j.ress.2015.12.003.

[16] G. J. Kacprzynski, G. J. Roemer, G. Modgil, A. Palladino, and K. Maynard, "Enhancement of physics-of-failure prognostic models with system level features," in *IEEE Aerospace Conference Proceedings*, 2002, vol. 6, pp. 2919–2925, doi: 10.1109/AERO.2002.1036131.

[17] J. Z. Sikorska, M. Hodkiewicz, and L. Ma, "Prognostic modelling options for remaining useful life estimation by industry," *Mech. Syst. Signal Process.*, vol. 25, no. 5, pp. 1803–1836, Jul. 2011, doi: 10.1016/j.ymssp.2010.11.018.

[18] Y. Qian and R. Yan, "Remaining Useful Life Prediction of Rolling Bearings Using an Enhanced Particle Filter," *IEEE Trans. Instrum. Meas.*, vol. 64, no. 10, pp. 2696–2707, Oct. 2015, doi: 10.1109/TIM.2015.2427891.

[19] M. Ibrahim, N. Y. Steiner, S. Jemei, and D. Hissel, "Wavelet-Based Approach for Online Fuel Cell Remaining Useful Lifetime Prediction," *IEEE Trans. Ind. Electron.*, vol. 63, no. 8, pp. 5057–5068, Aug. 2016, doi: 10.1109/TIE.2016.2547358.

[20] C. J. Lu and W. Q. Meeker, "Using Degradation Measures to Estimate a Time-to-Failure Distribution," *Technometrics*, vol. 35, no. 2, p. 161, May 1993, doi: 10.2307/1269661.

[21] M. Moghaddam, Q. Chen, and A. V. Deshmukh, "A neuro-inspired computational model for adaptive fault diagnosis," *Expert Syst. Appl.*, vol. 140, 2020, doi: 10.1016/j.eswa.2019.112879.

[22] G. A. Whitmore and F. Schenkelberg, "Modelling accelerated degradation data using Wiener diffusion with a time scale transformation.," *Lifetime Data Anal.*, vol. 3, no. 1, pp. 27–45, 1997, doi: 10.1023/a:1009664101413.

[23] X. Si, T. Li, Q. Zhang, and C. Hu, "Prognostics for linear stochastic degrading systems with survival measurements," *IEEE Trans. Ind. Electron.*, vol. 67, no. 4, pp. 3202–3215, Apr. 2020, doi: 10.1109/TIE.2019.2908617.

[24] K. Le Son, M. Fouladirad, A. Barros, E. Levrat, and B. Iung, "Remaining useful life estimation based on stochastic deterioration models: A comparative study," *Reliab. Eng. Syst. Saf.*, vol. 112, pp. 165–175, Apr. 2013, doi: 10.1016/j.ress.2012.11.022.

[25] J. P. KHAROUFEH and S. M. COX, "Stochastic models for degradation-based reliability," *IIE Trans.*, vol. 37, no. 6, pp. 533–542, Jun. 2005, doi: 10.1080/07408170590929009.

[26] F. Camci and R. B. Chinnam, "Health-state estimation and prognostics in machining processes," *IEEE Trans. Autom. Sci. Eng.*, vol. 7, no. 3, pp. 581–597, Jul. 2010, doi: 10.1109/TASE.2009.2038170.

[27] E. Ramasso and T. Denoeux, "Making use of partial knowledge about hidden states in HMMs: An approach based on belief functions," *IEEE Trans. Fuzzy Syst.*, vol. 22, no. 2, pp. 395–405, 2014, doi: 10.1109/TFUZZ.2013.2259496.

[28] L. Zhang, J. Lin, B. Liu, Z. Zhang, X. Yan, and M. Wei, "A Review on Deep Learning Applications in Prognostics and Health Management," *IEEE Access*, vol. 7, pp. 162415–162438, 2019, doi: 10.1109/ACCESS.2019.2950985.

[29] R. Huang, L. Xi, X. Li, C. Richard Liu, H. Qiu, and J. Lee, "Residual life predictions for ball bearings based on self-organizing map and back propagation neural network methods," *Mech. Syst. Signal Process.*, vol. 21, no. 1, pp. 193–207, Jan. 2007, doi: 10.1016/j.ymssp.2005.11.008.

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

[31] C. Zhang, P. Lim, A. K. Qin, and K. C. Tan, "Multiobjective Deep Belief Networks Ensemble for Remaining Useful Life Estimation in Prognostics," *IEEE Trans. Neural*

Networks Learn. Syst., vol. 28, no. 10, pp. 2306–2318, 2017, doi: 10.1109/TNNLS.2016.2582798.

[32] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Science (80-.).*, vol. 313, no. 5786, pp. 504–507, Jul. 2006, doi: 10.1126/science.1127647.

[33] G. Hinton *et al.*, “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups,” *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, 2012, doi: 10.1109/MSP.2012.2205597.

[34] L. Ren, J. Cui, Y. Sun, and X. Cheng, “Multi-bearing remaining useful life collaborative prediction: A deep learning approach,” *J. Manuf. Syst.*, vol. 43, pp. 248–256, Apr. 2017, doi: 10.1016/j.jmsy.2017.02.013.

[35] O. Bektas, J. A. Jones, S. Sankararaman, I. Roychoudhury, and K. Goebel, “A neural network filtering approach for similarity-based remaining useful life estimation,” *Int. J. Adv. Manuf. Technol.*, vol. 101, no. 1–4, pp. 87–103, Mar. 2019, doi: 10.1007/s00170-018-2874-0.

[36] G. Sateesh Babu, P. Zhao, and X.-L. Li, “Deep Convolutional Neural Network Based Regression Approach for Estimation of Remaining Useful Life.”

[37] X. Li, W. Zhang, and Q. Ding, “Deep learning-based remaining useful life estimation of bearings using multi-scale feature extraction,” *Reliab. Eng. Syst. Saf.*, vol. 182, pp. 208–218, Feb. 2019, doi: 10.1016/j.ress.2018.11.011.

[38] X. Li, W. Zhang, and Q. Ding, “Cross-domain fault diagnosis of rolling element bearings using deep generative neural networks,” *IEEE Trans. Ind. Electron.*, vol. 66, no. 7, pp. 5525–5534, Jul. 2019, doi: 10.1109/TIE.2018.2868023.

[39] A. Graves, A. Mohamed, and G. Hinton, “Speech Recognition with Deep Recurrent Neural Networks,” *ICASSP, IEEE Int. Conf. Acoust. Speech Signal Process. - Proc.*, pp. 6645–6649, Mar. 2013.

[40] A. Malhi, R. Yan, and R. X. Gao, “Prognosis of defect propagation based on recurrent neural networks,” *IEEE Trans. Instrum. Meas.*, vol. 60, no. 3, pp. 703–711, Mar. 2011, doi: 10.1109/TIM.2010.2078296.

[41] N. Gugulothu, V. Tv, P. Malhotra, L. Vig, P. Agarwal, and G. Shroo, “Predicting Remaining Useful Life using Time Series Embeddings based on Recurrent Neural Networks *-ing Useful Life using Time Series Embeddings based on Recurrent Neural Networks,” *Canada (Tor).*, vol. 10, 2017, doi: 10.1145/nnnnnnn.nnnnnnn.

[42] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, “Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies.”

[43] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” 2013.

[44] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[45] Y. Wu *et al.*, “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation,” Sep. 2016.

[46] J. Wang, G. Wen, S. Yang, and Y. Liu, “Remaining Useful Life Estimation in Prognostics Using Deep Bidirectional LSTM Neural Network,” in *Proceedings - 2018 Prognostics and System Health Management Conference, PHM-Chongqing 2018*, 2019, pp. 1037–1042, doi: 10.1109/PHM-Chongqing.2018.00184.

[47] R. Zhao, R. Yan, J. Wang, and K. Mao, “Learning to monitor machine health with convolutional Bi-directional LSTM networks,” *Sensors (Switzerland)*, vol. 17, no. 2, Feb. 2017, doi: 10.3390/s17020273.

[48] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, 1997, doi: 10.1109/78.650093.

[49] A. Al-Dulaimi, S. Zabihi, A. Asif, and A. Mohammadi, “Hybrid Deep Neural Network Model for Remaining Useful Life Estimation,” in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2019, vol. 2019-May, pp. 3872–3876, doi: 10.1109/ICASSP.2019.8683763.

[50] J. Li, X. Li, and D. He, “A Directed Acyclic Graph Network Combined With CNN and LSTM for Remaining Useful Life Prediction,” *IEEE Access*, vol. 7, pp. 75464–75475, 2019, doi: 10.1109/ACCESS.2019.2919566.

[51] Y. Song, G. Shi, L. Chen, X. Huang, and T. Xia, “Remaining Useful Life Prediction of Turbofan Engine Using Hybrid Model Based on Autoencoder and Bidirectional Long Short-Term Memory,” *J. Shanghai Jiaotong Univ.*, vol. 23, pp. 85–94, 2018, doi: 10.1007/s12204-018-2027-5.

[52] A. Van Den Oord, N. Kalchbrenner, and K. Kavukcuoglu, “Pixel recurrent neural networks,” in *33rd International Conference on Machine Learning, ICML 2016*, 2016, vol. 4, pp. 2611–2620, doi: 10.4249/scholarpedia.1888.

[53] A. Saxena, K. Goebel, D. Simon, and N. Eklund, “Damage propagation modeling for aircraft engine run-to-failure simulation,” in *2008 International Conference on Prognostics and Health Management, PHM 2008*, 2008, doi: 10.1109/PHM.2008.4711414.

[54] E. Ramasso and A. Saxena, “Performance benchmarking and analysis of prognostic methods for CMAPSS datasets,” *Int. J. Progn. Heal. Manag.*, vol. 5, no. 2, 2014.

[55] L. Jayasinghe, T. Samarasinghe, C. Yuenv, J. C. Ni Low, and S. Sam Ge, “Temporal convolutional memory networks for remaining useful life estimation of industrial machinery,” *Proc. IEEE Int. Conf. Ind. Technol.*, vol. 2019-Febru, pp. 915–920, 2019, doi: 10.1109/ICIT.2019.8754956.

[56] A. Botchkarev, “Performance Metrics (Error Measures) in Machine Learning Regression, Forecasting and Prognostics: Properties and Typology.”

[57] W. Yu, I. Y. Kim, and C. Mechefske, “Remaining useful life estimation using a bidirectional recurrent neural network based autoencoder scheme,” *Mech. Syst. Signal Process.*, vol. 129, pp. 764–780, 2019, doi: 10.1016/j.ymssp.2019.05.005.

[58] S. Bai, J. Z. Kolter, and V. Koltun, “An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling,” Mar. 2018.