Deep Neural Network Architectures for Prognostics and Health Management of Industrial Systems

Von der Fakultät für Ingenieurwissenschaften
Abteilung Elektrotechnik und Informationstechnik
der Universität Duisburg-Essen

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

von

Gavneet Singh Chadha
aus
Saharanpur, Indien

1. Gutachter: Prof. Dr.-Ing. Steven X. Ding
2. Gutachter: Prof. Dr.-Ing. Andreas Schwung

Tag der mündlichen Prüfung: 12. Juli 2024

Acknowledgments

This cumulative thesis includes a collection of publications which have been published during my stay at the South Westphalia University of Applied Sciences between 2017 and 2021. Being a research assistant in the Department of Automation Technology at the South Westphalia University of Applied Sciences and joining the cooperative Ph.D. program student at the Institute for Automatic Control and Complex Systems (AKS) at the University of Duisburg-Essen was lots of fun and joining it was one of the best decisions that I have ever made. I want to thank my adviser, Prof. Dr.-Ing. Andreas Schwung sincerely for being an excellent mentor and providing me with this special opportunity. He always encouraged me to pursue my own ideas and supported me whenever it was needed. I am very grateful for having been his student and part of the laboratory. I would also like to thank Prof. Dr.-Ing. Steven X. Ding of the Institute for Automatic Control and Complex Systems (AKS) at the University of Duisburg-Essen for his valuable advice and discussions.

Furthermore, I would like to thank my wonderful colleagues from the department of Automation Technology who always offered timely help and suggestions. A special thanks to Fabian Westbrink, Fernando Arévalo, Jan Niclas Reimann and Mohammed Sharafath Abdul Hameed for the valuable discussions. I am fortunate to have been part of a fantastic group and will always look back with joy at the time I spent in Soest. Also, many thanks to the AKS colleagues, Caroline Charlotte Zhu and Dr.-Ing. Chris Jan Louen for valuable discussions.

Finally, I want to express my deepest gratitude to my family in India and Ariane Werner for their unconditional support.

Leinfelden-Echterdingen, 28th July, 2024

Gavneet Singh Chadha

Enclosed Publications

This cumulative thesis includes the following published journal articles:

- Gavneet Singh Chadha, Ambarish Panambilly, Andreas Schwung, Steven X. Ding "Bidirectional deep recurrent neural networks for process fault classification," *ISA Transactions*, vol. 106, pp. 330–342, 2020. doi: 10.1016/j.isatra.2020.07.011

- Gavneet Singh Chadha, Intekhab Islam, Andreas Schwung, Steven X. Ding, "Deep Convolutional Clustering-Based Time Series Anomaly Detection," *Sensors*, vol. 21(16), 5488, 2021. doi: 10.3390/s21165488

- Gavneet Singh Chadha, Utkarsh Panara, Andreas Schwung, Steven X. Ding, "Generalized dilation convolutional neural networks for remaining useful lifetime estimation," *Neurocomputing*, vol. 452, pp. 182–199, 2021. doi: 10.1016/j.neucom.2021.04.109

- Gavneet Singh Chadha, Sayed Rafay Bin Shah, Andreas Schwung, Steven X. Ding, "Shared Temporal Attention Transformer for Remaining Useful Lifetime Estimation," *IEEE Access*, vol. 10, pp. 74244–74258, 2022. doi: 10.1109/ACCESS.2022.3187702

The extended summary of this thesis is based on the above-listed published journal articles.

Abstract

This thesis presents novel deep neural network architectures and training techniques to enhance their prediction capabilities in Prognostics and Health Management. These architectural and algorithmic changes are necessary to the standard deep neural network architectures since they were conceptualized to solve problems in different application fields, such as computer vision and natural language processing. Therefore, this thesis presents techniques to effectively model long-term time dependencies, efficient use of a large number of trainable parameters and productive use of unlabelled training data. Each publication proposes a solution to at least one of the above-stated drawbacks of standard deep neural networks.

First, the proposed bidirectional LSTM architecture offers a new perspective for time-series analysis, which enhances prediction performance for incipient fault categorization. It is easier to retain long-term data in LSTM cells thanks to the sliding window data function, allowing for considering even longer time series patterns. Second, a novel anomaly detection framework using an auxiliary loss function is proposed to learn a hidden representation which is amenable to the task. Based on the K-means clustering loss, the auxiliary loss function is only computed for a subset of the hidden variables. Third, a generalized dilation layer is proposed for adaptive sampling across the temporal and feature variables for multivariate time-series data analysis. Two novel training approaches are presented to make the generalized dilation training process compatible with gradient-based learning. Moreover, finally, two novel transformer neural network architectures are proposed with a focus on the parameter-sharing phenomena for modelling inter-dependency in the temporal and feature domains. The developed training strategies and algorithms have been tested on benchmark datasets available in the Prognostics and Health Management literature. The benchmark datasets' overall results underline the proposed approaches' superior performance. Lastly, because every strategy suggested in this thesis is very modular, it is possible to combine them in different ways to build dependable and robust architectures for system health monitoring.

Keywords

Deep Neural Networks, Convolutional Neural Networks, Recurrent Neural Networks, Transformers

Zusammenfassung

Diese kumulative Dissertation stellt neuartige tiefe neuronale Netzwerkarchitekturen und Trainingstechniken vor, um ihre Vorhersagefähigkeiten im Bereich der Prognose und des Gesundheitsmanagements zu verbessern. Diese architektonischen und algorithmischen Änderungen sind für die Standardarchitekturen tiefer neuronaler Netze notwendig, da sie konzipiert wurden, um Probleme in verschiedenen Anwendungsbereichen wie Computer Vision und Verarbeitung natürlicher Sprache zu lösen. Daher werden in dieser Dissertation Techniken zur effektiven Modellierung langfristiger Zeitabhängigkeiten, zur effizienten Nutzung einer großen Anzahl von trainierbaren Parametern und zur produktiven Nutzung unbeschrifteter Trainingsdaten vorgestellt. In jeder der vier Publikationen wird eine Lösung für mindestens einen der oben genannten Nachteile von standardmäßigen tiefen neuronalen Netzen vorgeschlagen.

Erstens bietet die vorgeschlagene bidirektionale LSTM-Architektur eine neue Perspektive für die Zeitreihenanalyse, die die Vorhersageleistung für die Kategorisierung von beginnenden Fehlern verbessert. Dank der Sliding-Window-Datenfunktion ist es einfacher, Langzeitdaten in LSTM-Zellen zu speichern, wodurch selbst längere Zeitreihenmuster berücksichtigt werden können. Zweitens wird ein neuartiger Anomalieerkennungsrahmen vorgestellt, der eine Hilfsverlustfunktion verwendet, um eine verborgene Darstellung zu lernen, die für die Aufgabe geeignet ist. Basierend auf dem K-Means Clustering Verlust wird die Hilfsverlustfunktion nur für eine Teilmenge der verborgenen Variablen berechnet. Drittens wird eine verallgemeinerte Dilatationsschicht für adaptives Sampling über die Zeit- und Merkmalsvariablen für die multivariate Zeitreihendatenanalyse vorgestellt. Um den generalisierten Dilatationstrainingsprozess mit dem gradientenbasierten Lernen kompatibel zu machen, werden zwei neuartige Trainingsansätze beschrieben. Abschließend werden zwei neuartige neuronale Transformer Netzwerkarchitekturen vorgeschlagen, wobei der Schwerpunkt auf Phänomenen der gemeinsamen Nutzung von Parametern zur Modellierung der gegenseitigen Abhängigkeit in den Zeit- und Merkmalsdomänen liegt. Alle entwickelten Trainingsstrategien und Algorithmen wurden an Benchmark-Datensätzen getestet, die in der Literatur zu Prognose und Gesundheitsmanagement verfügbar sind. Die Gesamtergebnisse der Benchmark-Datensätze unterstreichen die überlegene Leistung der vorgeschlagenen Ansätze. Da jede in dieser Arbeit vorgeschlagene Strategie sehr modular ist, ist es möglich, sie auf unterschiedliche Weise zu kombinieren, um zuverlässige und robuste Architekturen für die Überwachung des Systemzustands aufzubauen.

Contents

Preamble

Publications

List of Tables

List of Figures

List of Notations

Abbreviations

Abbreviation	Expansion
AE	Autoencoder
B-GRU	Bidirectional Gated Recurrent Units
B-LSTM	Bidirectional Long Short Term Memory
C-MAPSS	Commercial Modular Aero-Propulsion System Simulation
CNN	Convolutional Neural Networks
DBN	Deep Belief Network
DCCA	Deep Convolutional Clustering Algorithm
DNN	Deep Neural Networks
FC	Fully Connected
FeaR-STAT	Feature-Represented Shared Temporal Attention Transformer
FFN	Feed-Forward Network
FFNN	Feedforward Neural Networks
GD	Generalized Dilation
GDCNN	Generalized Dilation Convolutional Neural Network
GRU	Gated Recurrent Units
LSTM	Long Short Term Memory
ML	Machine Learning
MLP	Multilayer Perceptron
PE	Positional Encoding
PHM	Prognostics and Health Management

RBM	Restricted Boltzmann Machines
RMSE	Root Mean Square Error
RMSLE	Root Mean Square Log Error
RNN	Recurrent Neural Networks
RUL	Remaining Useful Lifetime
SFMHA	Split-Feature Multi-Head Attention
STA	Shared Temporal Attention
STAT	Shared Temporal Attention Transformer
STMHA	Split-Temporal Multi-Head Attention
TE	Tennessee Eastman
VAE	Variational Autoencoder

Mathematical notations

Notation	Description
$\forall$	For all
$\in$	Belongs to
$\mathbb{R}$	A set of real numbers
$\mathbb{R}^n$	Space of n-dimensional vectors
$\mathbb{R}^{n\times m}$	Space of n by m matrices
$\max\{\cdot\}$	Maximum of $\{\cdot\}$
$\mathbf{X}$	A matrix
$\mathbf{x}$	A vector
x	A scalar

Preamble

1. Introduction

The conventional approach to solving a problem using computer programming is to manually formulate a precise program, which a computer can execute step-by-step. This conventional approach requires a domain expert to explicitly write rules and methods for getting the desired result on a particular problem. However, machine learning (ML) algorithms have enabled computers to provide an unconventional way to solve complex problems. These ML algorithms' mathematical models can automatically learn the underlying rules and methods for solving a particular problem by observing the relevant input and the desired target data. This capability results in a highly flexible and ubiquitous application of ML algorithms.

Deep Learning is a subset of ML which uses Deep Neural Networks (DNN) as function approximators to map inputs to corresponding outputs without the need for explicit programming [1]. Therefore, the term DNN is used interchangeably with Deep Learning. DNNs have become a leading tool in various application fields which can be broadly categorised into computer vision, natural language processing and speech recognition [2]. Within these broad application fields, there are multiple specific tasks in which DNN have become the standard ML method. Some prominent examples of these specific tasks where Deep Learning has excelled in recent years are

- **Computer Vision** - image classification [3], object detection [4], semantic segmentation [5], autonomous driving [6].
- **Natural Language Processing** - sentiment analysis [7], machine translation [8], question answering [9], text summarisation [10].
- **Speech Recognition** - automatic speech recognition [11], speech enhancement [12], speech emotion recognition [13].

In addition to the above application areas, DNNs have also been extensively used recently for the health management of industrial and technical systems [14]. Standard DNN architectures like Multilayer Perceptrons (MLP), Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) are applied for a multitude of tasks in the field of Prognostics and Health Management (PHM). The reason for this widespread adoption is the ability of DNNs to extract high-level features from raw data by a hierarchy of simple nonlinear modules which successively transform the data into a more abstract space. Neural networks have also been proven to be universal function approximators [15], which means that given enough capacity, a single hidden layer neural network can arbitrarily well approximate any arbitrary continuous functions. It must be, however, noted that access to process data is essential for any PHM task for an industrial system. The recording and access to data are made possible with the advancements in the field of Industrial Internet of Things [16], smart manufacturing and the recent research agendas of

"Industry 4.0". Modern industrial systems are usually equipped with a network of sensors constantly monitoring the system state and helping supervise and regulate the respective actuators for safe and sustainable operation. This availability of sensor data makes using data-intensive DNN PHM methods more lucrative than ever before.

Standard DNN architectures in the literature include MLP, CNN, RNN and Transformers [17]. These architectures were originally proposed as structures for specific application domains. However, certain structural and learning style modifications can enhance their capabilities and prove their usefulness in other application domains. Therefore, in this thesis, the author will focus on architectural and algorithmic modifications to the RNN, CNN and Transformer architectures which have been proven to perform well in various ML application fields. All of the modifications, as mentioned earlier, are focused on solving the challenging PHM problems for industrial systems.

1.1 State of the Art - Deep Learning for Prognostics and Health Management of Industrial Systems

This section reviews the different Deep Learning architectures and algorithms presented for the PHM of industrial systems. Note that shallow neural networks have been used for fault detection and diagnosis in industrial processes in previous works such as in [18, 19, 20]. Furthermore, a variety of literature is available on model-based and other data-driven methods for PHM. A model of real-world industrial processes and systems describing their qualitative and quantitative behaviour must be available to use model-based methodologies. This model is used to generate estimates for the outputs of a system and subsequently constructing the disparity between the actual process outputs and their corresponding estimates is referred to as residual generation. The fundamental concept for observer model-based methods for PHM encompasses two key aspects: (i) replacing the process model with an observer or a model capable of furnishing dependable estimates of the process outputs, and (ii) allowing the designer the requisite design flexibility to attain desired decoupling, leveraging well-established observer theory principles [21]. An overview of model-based approaches for the following four categories: fault diagnosis for deterministic faults, fault diagnosis for stochastic faults, and fault diagnosis for discrete-events and hybrid systems, as well as fault detection for networked and distributed systems, is provided in [22].

The data-driven PHM methods depend on the measured process variables. A survey of the data-driven methods, including statistical, non-statistical and joint data-driven analysis tools, is provided in [23]. Among the various data-driven approaches, the multivariate analysis (MVA) technique, exemplified by Principal Component Analysis, Partial Least Squares, Independent Component Analysis, Fisher Discriminant Analysis, and Subspace Aided approaches have been recognized as powerful tools for tackling challenges related to statistical process monitoring and diagnosis [24]. A study comparing the above data-driven PHM approaches on fault diagnosis of the TE process has been provided in [25].

Still, this literature review focuses exclusively on Deep learning approaches. The literature review has been categorised based on the different Deep Learning architectures presented, i.e. Deep Feedforward Neural Networks (FFNN), CNN, RNN and Transformers.

Deep FFNN for Prognostics and Health Management of Industrial Systems: A stacked Autoencoder (AE) approach for automatic feature extraction and intelligent fault classification methodology has been proposed in [26] wherein a layer-wise unsupervised pre-training of AE is performed before a supervised fine-tuning for the overall architecture is done. A similar stacked denoising AE approach with active learning has been proposed in [27]. A fault classification technique in induction motors based on sparse AE has also been presented in [28]. A two-stage DNN methodology with a stacked denoising AE and softmax regression for Remaining Useful Lifetime (RUL) estimation has been presented in [29]. In contrast to the previous unsupervised pre-training approaches, a stacked supervised AE approach is presented in [30] where the fault-relevant features are extracted by training deep AE in a supervised learning manner.

In addition to AE, Restricted Boltzmann Machines (RBM) have also been extensively used as a building block for developing Deep Learning architectures. A Deep Belief Network (DBN), which comprises multiple layers of RBM, for identifying faults in a rolling bearing by fusing multi-sensor data has been presented in [31]. A similar DBN-based fault diagnosis methodology for chemical processes has been presented in [32]. Additionally, DBN has been tested and validated to predict the RUL of gears and bearings in [33], utilising the network's self-taught learning characteristics. A multi-objective DBN ensemble methodology has been proposed in [34] and its prognostic performance evaluated on a turbofan engine RUL estimation dataset, demonstrating outstanding prognostic performance.

RNN for Prognostics and Health Management of Industrial Systems: Usually, the variants of RNN architectures like the Long Short Term Memory (LSTM) networks and Gated Recurrent Units (GRU) have been proposed in the literature for the PHM of Industrial Systems. A fault diagnosis methodology using GRU-based denoising AE of rotating machinery has been presented in [35]. The multivariate time-series vibration data from bearings are modelled to identify the multiple fault cases possible in the machine. An LSTM network-based fault detection and identification methodology using data from track-side monitoring devices has been proposed in [36]. The temporal data comprises the time taken for a train to pass over a section of the track circuit which is taken as the sequence length of an event. A fault diagnosis framework for wind turbines with LSTM networks using multivariate time-series data is presented in [37], where the data includes displacement and acceleration sensor variables.

On the RUL estimation front, a health indicator-based LSTM network for predicting the degradation of an industrial ball bearing system by combining time domain and frequency domain features has been proposed in [38]. In [39], an adaptive kernel spectral clustering methodology is integrated with LSTM networks to predict the health status of ball bearings. The study [40] presents a tendency feature extraction methodology before feeding them to a stacked LSTM network for predicting the RUL of an aero-propulsion system simulation.

CNN for Prognostics and Health Management of Industrial Systems: CNNs have been extensively used in multiple application fields of PHM of industrial processes. Detection and identification of bearing faults using vibration data set have been proposed in [41, 42, 43, 44]. Usually, a slight modification in the data processing or the CNN architecture is present in these studies. Dislocated time-series convolutions are presented in [45] to model the periodic fault information between nonadjacent signals in electric

motors. A multi-scale CNN approach for fault diagnosis of Wind turbine gearbox under different operational conditions is presented in [46]. Fault diagnosis approaches with CNN of a semiconductor manufacturing process and a chemical process has been proposed in [47] and in [48] respectively.

CNNs have also been successfully applied to RUL prediction in various settings. CNNs were first used for RUL prediction in PHM in [49] wherein the convolution operation is applied along the temporal dimension of the multi-channel sensor data. A unit width convolution kernel for sharing kernel weights across raw sensors has been proposed in [50] enhance the network's ability to learn abstract features. A CNN model enhanced with a short-time Fourier transform methodology is presented in [51] to obtain the time-frequency domain information for a ball-bearing data set. Furthermore, the study [52] proposes a multitask learning approach with CNN by simultaneously learning two tasks, i.e. RUL estimation and health condition prediction.

Transformers for Prognostics and Health Management of Industrial Systems: The transformer model has been recently applied for PHM tasks in industrial systems. Transformers were applied for predicting the tool wear from CNC milling machines in [53]. A gated convolutional unit together with a Transformer encoder has been presented in [54] for RUL estimation of the turbofan engine dataset. And finally, a transformer model is applied in [55] for predictive maintenance of rotating machinery and automatic washing equipment.

1.2 Preliminaries

This section details the multivariate time-series problem that is considered in this thesis. The preliminaries are broadly divided into fault detection and identification and Remaining Useful lifetime estimation.

1.2.1 Preliminaries Fault Detection and Identification

In the context of fault detection and fault identification, a nonlinear system with n states, m inputs, p outputs, and k faults is considered. The state vector is denoted as $\mathbf{s}(t) = [s_1(t), s_2(t), \ldots, s_n(t)]^T$, the input vector as $\mathbf{u}(t) = [u_1(t), u_2(t), \ldots, u_m(t)]^T$, the output vector as $\mathbf{y}(t) = [y_1(t), y_2(t), \ldots, y_p(t)]^T$, and the fault vector as $\mathbf{f}(t) = [f_1(t), f_2(t), \ldots, f_k(t)]^T$.

The state equation can be formulated as follows:

$$\mathbf{s}(t+1) = \Theta\big(\mathbf{s}(t), \mathbf{u}(t), \mathbf{q}(t), \mathbf{f}(t)\big), \tag{1.1}$$

where $\Theta(\mathbf{s}(t), \mathbf{u}(t), \mathbf{f}(t))$ represents the dynamics of the system. $\mathbf{q}(t)$ denotes the process noise vector. The output equation can be formulated as follows:

$$\mathbf{y}(t) = \Omega\big(\mathbf{s}(t), \mathbf{u}(t), \mathbf{v}(t), \mathbf{f}(t)\big), \tag{1.2}$$

where $\Omega(\mathbf{s}(t), \mathbf{u}(t), \mathbf{v}(t), \mathbf{f}(t))$ represents the system outputs related to the internal states, inputs, measurement noise and faults. $\mathbf{v}(t)$ denotes the measurement noise vector. It is assumed in this thesis that the dynamics of the system are not known. Specifically, the non-linear function Θ and Ω are unknown complex system dynamics. Therefore, only

the measured data over time from the process, including system inputs, outputs, sensor, actuator and fault data, is considered for time-series analysis. This pure data-driven approach provides the adaptability of the fault detection and diagnosis system that can continuously learn and update itself from new system configurations and improve itself as more data becomes available. Features used as input for ANN are a subset of states, inputs, outputs or a function from each. In anomaly detection, the focus is on identifying any abnormal or faulty behaviour within a system without specifying the nature of the fault in advance. Therefore, anomaly detection does not require prior knowledge of the different fault cases.

Model-Based Residual Generation for Fault Detection and Identification

The objective of the Model-Based Residual Generation fault detection and classification scheme is to detect and isolate faults or abnormalities in the system's behaviour based on the residual vector by comparing the process measurement variables and their estimated state. A full-state observer for a non-linear dynamical system is designed to estimate the system states based on the available measurements. The observer is given by the following equation,

$$\widehat{\mathbf{s}(t+1)} = \Theta\big(\widehat{\mathbf{s}(t)}, \mathbf{u}(t), \mathbf{q}(t)\big) + \mathbf{L}\big(\mathbf{y}(t) - \Omega(\widehat{\mathbf{s}(t)}, \mathbf{u}(t), 0, 0)\big) \tag{1.3}$$

where $\widehat{\mathbf{s}} \in \mathbb{R}^n$ is the estimated state vector, $\mathbf{L} \in \mathbb{R}^{n \times p}$ is the observer gain matrix. The residual vector $\mathbf{r} \in \mathbb{R}^p$ is defined as the difference between the measured outputs and the estimated outputs obtained from the observer:

$$\mathbf{r}(t) = \mathbf{y}(t) - \Omega\big(\widehat{\mathbf{s}(t)}, \mathbf{u}(t), 0, 0\big). \tag{1.4}$$

Fault detection and classification involves analyzing the residuals to determine the presence of faults. In the above case, to perform fault detection and identification, the non-linear system dynamics represented by the functions Θ and Ω are known and the system inputs and output vectors are also known. The system states vectors are estimated via the observer as calculated from Eq. (1.3). The generated residuals are compared to predefined thresholds to identify abnormal conditions or faults in the system.

Data-driven Non-linear System Identification for Fault Detection and Identification

In the approach for data-driven system identification, the idea is to create a model of the system's normal behaviour using system identification techniques and then use deviations between the model predictions and actual measurements to detect faults. Specifically, various system identification techniques can be employed to estimate the non-linear system dynamics represented by the functions Θ and Ω. Usually in literature [56], the state function Θ is estimated by the system identification methodologies like first order and second order optimizers such gradient descent and Newtons Algorithm [57], but it is also possible to estimate the output function Ω. The above-mentioned iterative approach involves determining a nonlinear function $\widehat{\Theta}$ that describes the system dynamics utilizing the input-output data pairs $(\mathbf{s}(t), \mathbf{u}(t), \mathbf{s}(t+1))$ under normal operating conditions. Once the model $\widehat{\Theta}$ for the non-linear system is obtained, it's essential to validate its residual

generation for identifying faults. This can be done by comparing the predicted states $\widehat{\Theta}\big(\mathbf{s}(t), \mathbf{u}(t), \mathbf{q}(t)$ by the identified model with the actual observed state derivatives $(\mathbf{s}(t))$ for the collected data points. The difference between the predicted and observed values is used as the residual calculation:

$$\mathbf{r}(t) = \mathbf{s}(t+1) - \widehat{\Theta}\big(\mathbf{s}(t), \mathbf{u}(t), 0, 0\big). \tag{1.5}$$

To detect faults, the residuals $\mathbf{r}(t)$ are analyzed. Since the model $\widehat{\mathbf{\Theta}}\big(\mathbf{s}(t), \mathbf{u}(t), 0, 0\big)$ is designed to capture the system's normal behaviour, significant deviations between the residuals and zero could indicate the presence of a fault. Faults often result in changes to the system's dynamics, causing the model predictions to deviate from actual measurements. To determine whether a fault is present or not, a thresholding scheme is typically applied to the residuals as mentioned in [58].

Statistical and Multivariate Analysis for Fault Detection

The conventional statistical and multivariate analysis approaches for fault detection involve projecting or transforming process data from the measurement subspace to another subspace. Principal component analysis (PCA), dynamic principal component analysis and canonical variate analysis are the most common techniques for transforming process data. PCA is the most widely used example of one of these techniques and is widely accepted as a typical statistical fault detection technique.

In PCA, the idea is to transform the original variables into a new set of uncorrelated variables called principal components. These components capture the maximum variance in the data. The first few principal components usually represent most of the variability in the dataset [24]. Consider a dataset matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$ consisting of n samples and p variables is normalized by reducing it to zero mean and unit standard deviation. This dataset matrix is a subset of states, inputs, outputs or a function from each. For numerical stability, especially when dealing with large datasets, the Singular Value Decomposition (SVD) approach is used for performing PCA. The SVD on the process data $\mathbf{X}$ is executed as

$$\frac{1}{\sqrt{n-1}}\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T, \tag{1.6}$$

where $\mathbf{U} \in \mathbb{R}^{n \times n}$ represents the orthonormal basis in the sample space, $\mathbf{V} \in \mathbb{R}^{p \times p}$ represents the orthonormal basis in the variable space and $\mathbf{S} \in \mathbb{R}^{n \times p}$ contains the non-negative real *singular* values. The loading vectors $\mathbf{P}$ are the column vectors in the matrix $\mathbf{V}$ which are subsequently used for calculating the Hotelling's T^2 statistic defined as

$$T^2 = \mathbf{x}^T\mathbf{P}\mathbf{S}_a^{-2}\mathbf{P}^T\mathbf{x}, \tag{1.7}$$

where $\mathbf{x} \in \mathbb{R}^m$ is an observation vector, $\mathbf{P}$ contains the loading vectors corresponding to the a largest singular values in the $\mathbf{S}$ matrix, $\mathbf{S}_a$ contains the first a rows and columns of the $\mathbf{S}$ matrix, i.e. $\mathbf{S}_a$ and $\mathbf{P}$ represent the truncated matrices from $\mathbf{S}$ and matrix V respectively. Depending on the number of loading vectors a (the principal components), the threshold for normal operating condition can be calculated for a significant level α as

$$T_\alpha^2 = \frac{(n^2-1)a}{n(n-1)}F_\alpha(a, n-a), \tag{1.8}$$

where $F_\alpha(a, n-a)$ denotes the upper $100\alpha\%$ critical point of the F-distribution with a and n-a degrees of freedom. An observation vector with a Hotelling's T^2 statistic outside the defined threshold indicates that a fault has taken place.

Deep Neural Networks for Fault Detection

DNN architectures such as CNN, RNN and Transformers are responsible for the feature extraction from raw input data and therein indirectly modelling the non-linear system dynamics for fault detection and identification. In comparison to the statistical and multivariate methods described in the previous section, DNN excels at capturing long-term temporal dependencies and complex degradation patterns from condition monitoring input data. More specifically, fault detection and identification using deep neural networks can be mathematically formulated as a supervised learning problem. For simplicity, consider a solitary time-series sequence input data as a matrix $\mathbf{X} \in \mathbb{R}^{t \times p}$ where t is the number of time steps, and p is the number of condition monitoring features. This sequential input dataset is mainly used from feedforward dynamic systems and not from dynamic systems in a closed feedback setting.

Consider n as the total time-series samples available for the training dataset. In the supervised learning case, each time series sample has a corresponding fault label associated with it. Specifically, the corresponding fault labels is a vector $\mathbf{y} \in \mathbb{R}^n$ where each element $\mathbf{y_i}$ represents the fault class of the i-th time-series sample. Consider that the DNN is represented by the function λ_w parameterized by w, which takes the input data $\mathbf{X}$ and predicts the fault labels $\mathbf{y}$. This can be represented mathematically as

$$\mathbf{y}' = \lambda_w(X), \tag{1.9}$$

where $\mathbf{y}'$ is the predicted fault labels. The main goal of fault detection and identification using DNN is to determine the function λ_w using condition monitoring data and labels. In contrast to the model-based and the data-driven methods mentioned above, the DNN learns to transform input data into meaningful representations that enable it to accurately classify or predict the fault labels directly. The hidden layers of the DNN learn to approximate non-linear functions which are necessary to distinguish between different classes which are characterized by features in the input data. These non-linear function approximations take into consideration the system dynamics during the training process by adjusting the parameters w of the DNN via gradient descent.

DNNs like CNNs and RNNs, can approximate the highly nonlinear functions and dynamical systems directly because of the following properties.

- DNN use nonlinear activation functions, such as Rectified Linear Unit (ReLU) [59], sigmoid, or tanh within each computation unit allowing the architecture to model complex relationships between inputs, states and output data.

- Composition of multiple computational layers in a hierarchical manner in a DNN creates a hierarchy of features and non-linear transformations. These temporal feature identification plays an important part in the subsequent fault identification and classification task.

- The convolution operation in CNN performs local operations on the input data, allowing the network to identify temporal patterns. Stacking these temporal convolutional layers allows the CNN architecture to capture long temporal dependencies in a dynamical system.

- The recurrent connections in RNN allow them to maintain an internal hidden state, which captures information from previous time steps. This hidden state is updated at each time step using nonlinear transformations. In this way, RNNs excel at capturing non-linear temporal dependencies, such as those found in dynamical systems.

Due to the above properties, DNNs like CNN and RNN can do a direct function approximation without the need for residual generation as explained in section 1.2.1 and 1.2.1. Therefore, in this thesis, various DNN architectures are identified for fault detection which can approximate the dynamical and non-linear nature of systems using condition monitoring data and labels. The process of this function determination is usually referred to as training. The DNN is usually trained to minimize a loss function that quantifies the error between the predicted labels $\mathbf{y}'$ and the true labels $\mathbf{y}$. For fault detection and identification, A common choice is the categorical cross-entropy loss such as

$$L(w) = -\sum_{i=1}^{n}\sum_{j=1}^{c} \mathbf{y}_{ij} \log(\mathbf{y'}_{ij}) \tag{1.10}$$

where c is the number of fault classes, $\mathbf{y}_{ij}$ is a binary indicator (0 or 1) of whether the i-th data sample belongs to the j-th class, and $\mathbf{y'}_{ij}$ is the predicted probability that the i-th data sample belongs to the j-th class. The training goal is to find the optimal set of parameters w that minimizes the loss function $L(w)$ such as

$$w_{t+1} = \arg\min_{w_t} L(w_t) \tag{1.11}$$

where w_{t+1} are the updated parameters at iteration $t+1$ and $L(w_t)$ denote the loss at iteration t. Usually, a gradient-based optimization algorithm like stochastic gradient descent updates the parameters w iteratively by computing gradients of the loss with respect to the parameters and adjusting the parameters in the direction that minimizes the loss. Specifically, the DNN parameters w are usually updated with gradient descent as

$$w_{t+1} = w_t - \eta \nabla L(w_t) \tag{1.12}$$

where η is the learning rate and $\nabla L(w_t)$ denotes the gradient of the loss function L with respect to the parameters w_t for a specific training example. Once the DNN is trained until a predetermined number of iteration steps or a minimum loss value is achieved, it can be used for fault detection and classification by feeding new data samples through the trained model. The output of the model $\mathbf{y}'$ can be interpreted as probabilities belonging to each fault class, and the class with the highest probability can be selected as the predicted fault class.

Since the trained DNN model performs a classification task, the selection of the right evaluation metric is crucial when evaluating the fault detection performance. The performance metric should provide an unbiased and neutral evaluation of the model's predictions

to avoid favouring the dominant class. This is especially relevant for fault detection and diagnosis tasks where usually the normal operating condition is the dominant class as compared to the faulty condition. Therefore, the F_1 score is an appropriate measure since it ensures an even balance between false positives and false negatives in the classified samples. Moreover, it serves as a versatile metric for general multi-class classification. The F_1 score for the binary classification problem of fault detection utilizes the True Positives ρ, the False Positives σ and the False Negatives τ. These are calculated by comparing the predictions made by the classifier with the ground truth labels $\mathbf{y}$. The elements within the prediction vector $\mathbf{y}'$ from the DNN represent the likelihood of the corresponding instance belonging to the positive or specifically the faulty class. These likelihood values are transformed to obtain the evaluation vector $\mathbf{z}' \in \{0, 1\}^n$ from the prediction vector of a neural network and a threshold value θ. For a binary classification case, the threshold value is typically taken as 0.5. Instances with a likelihood greater than or equal to this threshold are calculated as 1, representing the faulty class, while instances with a probability less than the threshold are calculated as 0 representing the non-faulty class. Specifically,

$$\mathbf{z_i'} = \begin{cases} 1, & \text{if } \mathbf{y_i'} \geq \theta \\ 0, & \text{otherwise} \end{cases} \tag{1.13}$$

where $\mathbf{z_i'}$ is the class assignment to either the faulty case or the non-faulty case for the i^{th} time-series instance. The Eq. (1.13) essentially applies a binary threshold to convert the probabilities outputted by the neural network into binary predictions. The evaluation vector and ground vector are used for calculating the ρ, σ and τ as,

$$\rho = \sum_{i=1}^{n} (z_i' \cdot y_i), \tag{1.14}$$

$$\sigma = \sum_{i=1}^{n} (z_i' \cdot (1 - y_i)), \tag{1.15}$$

$$\tau = \sum_{i=1}^{n} ((1 - z_i') \cdot y_i). \tag{1.16}$$

Subsequently, the F_1 score is defined as

$$F_1 = \frac{2\rho}{2\rho + \sigma + \tau}, \tag{1.17}$$

where in binary classification case, the true positive counts ρ are the correctly identified faulty samples, false positive counts σ are non-faulty samples that were misclassified as faulty samples and false negative counts τ are the fault examples that were misclassified as normal operating condition samples.

1.2.2 Preliminaries Remaining Useful Lifetime Estimation

RUL aims to estimate the remaining time until a system or component will no longer perform its intended function adequately. A non-linear state-space model for RUL estimation can be derived to model the complex relationships and dynamics within the system. In the

case of regression-based RUL estimation, a set of state vectors that capture relevant information about the system's degradation are considered. A nonlinear system with n states, m inputs is considered. The state vector is denoted as $\mathbf{s}(t) = [s_1(t), s_2(t), \ldots, s_n(t)]^T$, the input vector as $\mathbf{u}(t) = [u_1(t), u_2(t), \ldots, u_m(t)]^T$ and the output as $y(t)$. The state equation can be formulated as follows:

$$\mathbf{s}(t+1) = \mathbf{\Gamma}\big(\mathbf{s}(t), \mathbf{u}(t), \delta(t), \mathbf{q}(t)\big), \tag{1.18}$$

$$\delta(t+1) = \mathbf{\Pi}\big(\mathbf{s}(t), \delta(t)\big), \tag{1.19}$$

where $\mathbf{s}(t)$ is a vector of state variables at time t, $\mathbf{u}(t)$ represents any inputs vector, $\delta(t)$ represent the system degradation variable that affects the degradation process, $\mathbf{q}(t)$ is the process noise that captures uncertainties, $\mathbf{\Pi}$ is the non-linear degradation function and $\mathbf{\Gamma}$ is a non-linear function that describes the non-linear dynamic nature of the system and its evolution over time. The features modelled above are inputs to a regression model that predicts the RUL. The observation equations can be written as:

$$y(t) = \mathbf{\Xi}\big(\mathbf{s}(t), \mathbf{u}(t), \mathbf{v}(t)\big), \tag{1.20}$$

where $y(t)$ is the output vector at time t, $\mathbf{v}(t)$ represents the measurement noise that accounts for uncertainties in the RUL estimation and $\mathbf{\Xi}$ is a non-linear function that maps the state variables, inputs and the system outputs. Assuming that X_t be the random variable representing the lifetime of an asset, then the probability density function of the random variable representing RUL at time t is $F(X_t) \mid Y_t)$ where Y_t is the history of the operational environment and the health of the system up to time t [60]. The estimation of the probability density function F is performed by the below methods of model-based and statistical prognostic methods. The regression-based machine learning methods do not estimate a probability density function of the RUL. The core idea is that the condition of systems can be represented using essential condition monitoring variables. By tracking, analyzing, and forecasting these variables, it becomes possible to estimate the RUL of the systems.

Model-Based prognostics for Remaining Useful Lifetime estimation

The approaches based on model-based prognostics presume the existence of a precise mathematical representation of the system under consideration. As observed from Eq. (1.18) and Eq. (1.19) the mathematical model of a system includes the Degradation Variable which needs to be estimated. In literature, the estimation of the system degradation variable $\delta(t)$ is performed depending on the use case since it requires specific mechanistic knowledge relevant to the system. Some examples of degradation estimation and the connection of the state variables that describe the condition of a system and its RUL probability density function through mechanistic modelling are [61, 62]. In [61] a stiffness-centered degradation model for bearing systems using vibration analysis and damage mechanics is presented. Similarly, other use cases such as for fatigue crack dynamics derive other non-linear models for RUL estimation.

Stochastical Prognostics Methods for Remaining Useful Lifetime estimation

Since modelling from first principles is not always possible, data-driven statistical methods use their ability to transform high-dimensional noisy data into lower-dimensional information for RUL estimation. These statistical models are developed by aligning them with existing data using a probabilistic framework, all without dependence on any underlying physics or engineering principles. The basic idea is that random fluctuations influence the degradation process, and the RUL can be estimated based on how degraded the component is at a given time. In literature, these random fluctuations or degradation are modelled via stochastic processes such as Wiener [63] and Gamma processes [64] which are used for estimating the RUL probability density function. Specifically, the degradation process via a Wiener process can be represented as $Y(t) = \lambda t + \sigma B(t)$ where λ is the drift (mean rate of degradation) and σ is the diffusion coefficient (volatility of degradation) and $W(t)$ be a Wiener process (standard Brownian motion). The X_t, is when the degradation level reaches a certain failure threshold. Specifically, $X(t)$ can be defined as $X(t) = \inf\{t > 0 : Y(t + RUL(t)) \geq w \mid Y_t \leq w$ where w is the failure threshold [60]. The probability distribution function of the $X(t)$ is given by the inverse Gaussian distribution as

$$F(X(t)) = \frac{w - Y(t)}{\sqrt{2\pi\sigma^2 t^3}} \exp\left(\frac{w - Y(t) - \lambda t^2}{2\sigma^2 X_t}\right). \tag{1.21}$$

Regression Methods for Remaining Useful Lifetime estimation

In regression-based approaches, the aim is to create mathematical models directly from collected condition monitoring data, rather than relying on intricate system physics and degradation models. These approaches utilize historical records to generate predictions solely in terms of condition monitoring data. The data can be calorimetric, power, vibration and acoustic signal, temperature, pressure, oil debris, currents, voltages and spectrometric data [65]. Let $\mathbf{C(t)}$ denote the condition monitoring features of the system at time t, $r(t)$ denote the actual remaining useful life of the system at time t, which is the time left until the system fails, and $\hat{r}(t)$ denote the predicted remaining useful life at time t obtained from the regression model. These condition monitoring features can be a combination of state variables $\mathbf{s_t}$, system input variables $\mathbf{u_t}$, system output variables $\mathbf{y_t}$.

The goal of regression-based RUL estimation is to find a mapping function H_w parameterized by w, that relates the condition monitoring features $\mathbf{C(t)}$ to the remaining useful life $r(t)$. To do this, the RUL is predicted as

$$\hat{r}(t) = H_w(\mathbf{C(t)}). \tag{1.22}$$

The common techniques for identifying regression functions for RUL estimation include support vector regression, DNN and evolutionary computation techniques [66]. In contrast to the model-based and the stochastical methods mentioned above for RUL estimation, the DNN learns to transform input data into meaningful representations that enable it to accurately predict the RUL directly. The hidden layers of the DNN learn to approximate non-linear functions which are necessary to map the condition monitoring input data to a continuous output. These non-linear function approximations take into consideration the system dynamics during the training process by adjusting the parameters w of the DNN

via gradient descent. The approximation of these functions is performed via the stochastic gradient descent as explained in Eq. (1.12). As mentioned in section 1.2.1, DNN have the following properties which make them suitable for identifying the regression function $\mathbf{H}$:

- Usage of nonlinear activation functions ReLU, sigmoid, or tanh for each hidden unit allows the architecture to model complex relationships between inputs, states, degradation variables and output data.

- Composition of multiple computational layers in a hierarchical manner in a DNN creating a hierarchy of time-series features. These temporal features form the basis for modelling the degradation, state evolution and output function from the system.

- The convolution operation in CNN performs local operations on the input data, allowing the network to identify temporal patterns. Stacking these temporal convolutional layers allows the CNN architecture to capture long temporal dependencies in a dynamical system.

- The self-attention mechanism enables Transformers to capture long-range non-linear dependencies from the system condition monitoring data. Furthermore, the multi-head mechanism allows the model to attend to various temporal features and patterns in the condition monitoring data simultaneously.

The regression model is trained to minimize the loss functions such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and Root Mean Squared Log Error (RMSLE). This regression function $\mathbf{H}$ is learned from historical data, where the n observed pairs $(\mathbf{C_t}, r_t)$ are used to minimize the RMSE loss function as

$$\min_{w} L_w(\mathbf{C_t}, r_t) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\big(r(t) - \hat{r}(t)\big)^2} \tag{1.23}$$

where w are the parameters of the regression function $\mathbf{H}$. The loss function defined in the Eq. (1.23) is the evaluation metric for RUL estimation performance. A qualitative performance evaluation is also sometimes conducted as depicted in Eq. (2.54). Once the DNN is trained until a predetermined number of iteration steps or loss value is achieved, it can be used for RUL estimation by feeding new data samples through the trained model. The output of the model $\hat{r}(t)$ can be interpreted as the RUL at time t.

1.3 Objectives of the Thesis

The main objective of this thesis is to identify the weaknesses of standard DNN architectures and develop new DNN architectures for better prediction capabilities. DNN architectures are a good fit for data-based condition monitoring because they have been proven to identify underlying data distributions directly from raw data rather than the hand-engineered features. However, most of these standard architectures that consider the multivariate time-series problem cannot effectively model long-term time dependencies, do not efficiently use many trainable parameters, are not interpretable for individual input

features and do not effectively use unlabelled training data. Therefore, to our best knowledge, although standard DNN architectures can be helpful for the multivariate time-series problem, certain modifications are necessary to capture the long temporal dependencies in high dimensional real-world non-linear processes. A summary of the objectives and goals of this thesis can be stated as follows:

- Identify the limitations of standard deep neural network architectures for multivariate time-series analysis, especially for prognostics and health management of industrial systems, specifically feedforward dynamic systems and not from dynamic systems in a closed feedback setting.
- Identify and develop DNN architectures such as CNN, RNN and Transformers that overcome the limitations of the standard DNN architectures.
- Development and evaluation of suitable learning and data preparation methodologies to model the long-term time dependencies of fault development in industrial systems, specifically feedforward dynamic systems and not from dynamic systems in a closed feedback setting.
- Development and evaluation of training algorithms to efficiently use the trainable parameters by selecting the time samples and the sensor channels most relevant for the prediction task.
- Incorporate unlabelled data for training DNN for efficient anomaly detection in industrial systems.
- Development and evaluation of parameter-efficient Transformer architectures which can better correlate the degradation patterns of the machine in multiple instances.

1.4 Structure of the Thesis

The thesis consists of two parts, the Preamble and Publications. Each chapter of the Preamble is briefly described in the following.

Chapter 1: Introduction: This chapter introduces and describes the motivation for this work. Furthermore, this work's main objectives, desired contributions and organization are defined.

Chapter 2: Summary of the Publications: In this chapter, a summary is provided of all the publications which are included in this thesis. The developed novel methodologies and algorithms are fundamentally explained along with the outcomes.

Chapter 3: Conclusions and Future Work: This chapter includes the overall conclusion and findings from the entire work. It also comprises avenues where future work could be done.

Section: Publications: This section includes all the publications which are part of this thesis.

Chapter I: Bidirectional Deep Recurrent Neural Networks for Process Fault Classification: This chapter includes the first publication, which tackles the challenge of fault detection and classification over long time horizons. The proposed methodology

is based on a Bidirectional Long Short Term Memory Network and is evaluated on the TE process.

Chapter II: Deep Convolutional Clustering-Based Time Series Anomaly Detection: In this chapter, a novel anomaly detection approach using convolutional AE and K-means clustering is presented. The developed approach is evaluated and compared with other state-of-the-art methodologies on the TE benchmark dataset.

Chapter III: Generalized dilation convolutional neural networks for remaining useful lifetime estimation: This chapter is dedicated to the novel Generalized Dilation approach for CNN. The proposed approach can model time-series data over a long time horizon with the ability to select the relevant features for the RUL prediction task.

Chapter IV: Shared Temporal Attention Transformer for Remaining Useful Lifetime Estimation: This chapter comprises the publication of a novel transformer-based RUL estimation framework. Two Transformer architectures are presented, which can model the degradation patterns across the input features in a parameter-efficient manner.

2. Summary of Publications

This chapter provides a summary of the contributions entailed in this thesis. Section 2.1 summarises the contribution of fault classification in industrial processes over long time horizons. Furthermore, the section details the methodology for unsupervised and semi-supervised learning-based anomaly detection. Finally, the proposed methodologies of Generalized Dilation and Shared Temporal Attention for RUL estimation are explained in Section 2.2.

2.1 Fault classification with Deep Recurrent Neural Networks and Anomaly Detection with Clustering augmented Deep Convolutional Neural Networks

This section details the time series-based and fault classification and unsupervised learning-based anomaly detection based on bidirectional recurrent neural networks and 1D-convolutional neural network-based deep AE, respectively. The proposed approaches are applied to the Tennessee Eastman (TE) benchmark process to test the effectiveness of the mentioned deep architectures and provide a detailed comparative analysis of the different architectural settings and fault situations. The content of this section is based on the publications described in Chapter I and Chapter II

2.1.1 Introduction

Several methods are presented in the literature for fault detection and classification of dynamical systems. The typical Deep Learning models like Deep AE and DBN do not capture this dynamic nature of industrial processes where incipient faults can occur gradually over a more extended period. Therefore the fault detection and classification model should be able to model this dynamic nature of modern industrial processes to compensate for their behaviour. Therefore, the approach presented in this section provides a novel system architecture using Bidirectional-LSTM (B-LSTM), which operates directly on the raw sensor data and models information over a longer time horizon. I also compare the prediction capabilities of other recurrent architectures and report their strengths and weaknesses in detail.

The second approach proposed in this chapter is the k-means clustering augmented 1D-convolutional neural network-based deep AE architecture for anomaly detection. Since a labelled dataset with all the possible operating conditions of a system is not always possible, unsupervised or semi-supervised learning-based data-driven machine learning methods are the only alternative for anomaly detection. The standard unsupervised learning-based Deep Learning methods are usually encompassed under the AE framework,

namely Denoising AE [67], Variational AE (VAE) [68] and Adversarial AE [69]. In contrast to the existing approaches, the proposed clustering augmented AE framework presented in this thesis splits the latent representation into two sets. This split of the latent space in discriminative and reconstructive latent variables allows for superior anomaly detection capabilities for multivariate time-series datasets.

2.1.2 Recurrent Neural Networks Architectures for Fault Classification

In contrast to the standard feedforward neural networks, RNN architectures have been proposed in this work because these can efficiently model the temporal dependencies in a dynamic process. RNNs possess the ability to model sequential information over time, which their feedforward counterparts are not able to do. The different RNN architectures tested in this work are briefly explained in this section.

Vanilla Recurrent Neural Networks Vanilla RNN includes a chain of neurons which receive input over a time window. Each neuron receives input from the current time step and the hidden representation from the previous time step. Formally, the hidden representation of an RNN cell at time step t yields

$$\mathbf{h_t} = \psi(\mathbf{a_t}) = \psi(\mathbf{W_x x_t} + \mathbf{W_h h_{t-1}} + \mathbf{b}), \tag{2.1}$$

where $\mathbf{W_x} \in \mathbb{R}^{d_h \times d_x}$ denotes the weight matrix from the input to hidden states, $\mathbf{W_h} \in \mathbb{R}^{d_h \times d_h}$ denotes the hidden state to hidden state weight matrix, $\mathbf{b} \in \mathbb{R}^{d_x}$ denote the input biases, $\mathbf{x_t} \in \mathbb{R}^{d_x}$ denote the input vector at time-step t and $\mathbf{h_{t-1}} \in \mathbb{R}^{d_h}$ denote the hidden state at time step $t-1$ or previous time step. The activation function is denoted as $\psi(\cdot)$ and is usually a sigmoid activation function described as:

$$\psi(\mathbf{a_t}) = \frac{1}{1 + e^{-\mathbf{a_t}}}. \tag{2.2}$$

Long-Short Term Memories The LSTM network [70] consists of LSTM cells as the building block for sequential modelling. The information at each time step t in an LSTM cell flows through a gated architecture, namely, the forget $\mathbf{f_t}$, the input $\mathbf{i_t}$ and output gate $\mathbf{o_t}$. Each of these gates performs an independent function, as their name suggests, to determine how much information the cell should carry forward to the next time step and how much information should be forgotten. The information flow is controlled through the sigmoid activation function as explained in Eq. (2.2). In addition to the hidden state $\mathbf{h_t}$, an LSTM cell also has a cell state $\mathbf{c_t}$, which is passed onto the next time step. The function of the cell state is to act as an information highway, as no activation functions are used for it. Formally, the vector equations for an LSTM cell at time step t yields

$$\tilde{\mathbf{c_t}} = \gamma(\mathbf{W_z x_t} + \mathbf{R_z h_{t-1}} + \mathbf{b_z}), \tag{2.3}$$
$$\mathbf{i_t} = \sigma(\mathbf{W_i x_t} + \mathbf{R_i h_{t-1}} + \mathbf{b_i}), \tag{2.4}$$
$$\mathbf{f_t} = \sigma(\mathbf{W_f x_t} + \mathbf{R_f h_{t-1}} + \mathbf{b_f}), \tag{2.5}$$
$$\mathbf{c_t} = \tilde{\mathbf{c_t}} \odot \mathbf{i_t} + \mathbf{c_{t-1}} \odot \mathbf{f_t}, \tag{2.6}$$
$$\mathbf{o_t} = \sigma(\mathbf{W_o x_t} + \mathbf{R_o h_{t-1}} + \mathbf{b_o}), \tag{2.7}$$
$$\mathbf{h_t} = \psi(\mathbf{c_t}) \odot \mathbf{o_t}, \tag{2.8}$$

where $\mathbf{W_z}, \mathbf{W_i}, \mathbf{W_f}, \mathbf{W_o} \in \mathbb{R}^{d_h \times d_x}$ denote the weight matrices for the input, $\mathbf{R_z}, \mathbf{R_i}, \mathbf{R_f}, \mathbf{R_o} \in \mathbb{R}^{d_h \times d_h}$ denote the weight matrices for the hidden state, $\mathbf{b_z}, \mathbf{b_i}, \mathbf{b_f}, \mathbf{b_o} \in \mathbb{R}^{d_x}$ denote the bias vectors, $\odot$ refers to pointwise multiplication, $\tilde{\mathbf{c_t}}$ is the candidate cell state, σ, γ and ψ are point-wise activation functions. In the standard setting, the sigmoid function is used for σ, and the hyperbolic tangent function is used for γ and ψ. Since the hidden state of the LSTM cell is calculated using the cell state and the cell state is a linear combination of the input and forget gate, the vanishing gradient problem is mitigated to a certain degree.

Gated Recurrent Units A GRU unit is the basic building block of the Gated Recurrent Network [71] and is a simplified version of the LSTM network. A GRU cell consists of a reset gate r_t and an update gate z_t. The output of these two gates is eventually used to calculate the hidden state h_t of the cell at time step t. After that, this hidden state is passed onto the next cell at time step $t+1$ for further modelling. Since the GRU cell combines the forget and input gates of the LSTM cell into a solitary update gate, the network has fewer training parameters than an LSTM network. This decrease in the number of trainable parameters helps in faster training. Formally, the vector equations for a GRU cell at time step t yields

$$\mathbf{r_t} = \sigma(\mathbf{W_r x_t} + \mathbf{R_r h_{t-1}} + \mathbf{b_r}), \tag{2.9}$$

$$\mathbf{z_t} = \sigma(\mathbf{W_z x_t} + \mathbf{R_z h_{t-1}} + \mathbf{b_z}), \tag{2.10}$$

$$\mathbf{h_t} = (1 - \mathbf{z_t}) \odot \mathbf{h_{t-1}} + \mathbf{z_t} \odot \psi(\mathbf{W_h x_t} + \mathbf{R_h}(\mathbf{r_t} \odot \mathbf{h_{t-1}}) + \mathbf{b_h}), \tag{2.11}$$

where $\mathbf{W_z}, \mathbf{W_r}, \mathbf{W_h} \in \mathbb{R}^{d_h \times d_x}$ are the input weight matrices, $\mathbf{R_z}, \mathbf{R_r}, \mathbf{R_h} \in \mathbb{R}^{d_h \times d_h}$ are the hidden weight matrices, $\mathbf{b_z}, \mathbf{b_r}, \mathbf{b_h} \in \mathbb{R}^{d_x}$ are the bias vectors and $\odot$ refers to pointwise product.

Bidirectional RNN Standard RNN architectures read a sequence of input by processing first the past and then the future. Specifically, the input is an ordered set of vectors represented as $\{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_t}\}$ where t the length of the input signal. The idea of Bidirectional RNN [72] is to train two separate networks that read the input signal in opposite directions and eventually combine their hidden representation, feeding into the deeper layers for processing. Particularly, the input of the first RNN is $\{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_t}\}$ and the second RNN is $\{\mathbf{x_t}, \mathbf{x_{t-1}}, \ldots, \mathbf{x_1}\}$. Specifically, as is illustrated in Fig. 2.1, each node $\mathbf{A_t}$ and $\mathbf{A'_t}$ represents one cell of the bidirectional RNN layer and the sequence of hidden states $\mathbf{h_t}$ and $\mathbf{h'_t}$ for both the RNN in matrix form yields

$$\mathbf{h_t} = \psi(\mathbf{A_t}) = \psi(\mathbf{W_x x_t} + \mathbf{W_h h_{t-1}} + \mathbf{b}) \tag{2.12}$$

$$\mathbf{h'_t} = \psi(\mathbf{A'_t}) = \psi(\mathbf{W'_x x_t} + \mathbf{W'_h h'_{t-1}} + \mathbf{b'}), \tag{2.13}$$

where $\mathbf{h_0}$=$\mathbf{h'_t}$ and $\mathbf{h_t}$=$\mathbf{h'_0}$. In this thesis, the bidirectional RNN idea is used in the context of LSTM (B-LSTM) and GRU (B-GRU) networks, where the hidden state and cell states of the networks are concatenated. Consequently, the size of the hidden and cell states in the LSTM and GRU networks doubles. The advantage of using B-LSTM and B-GRU networks is that the fault detection and classification system can simultaneously process information in both temporal directions. Essentially, the main difference between an LSTM and a B-LSTM is the viewpoint of the network on the input data signal, based on which the prediction is made. In a vanilla LSTM model, the viewpoint is at the end of the considered sequence, i.e. the model has to refer to the past. On the contrary, as

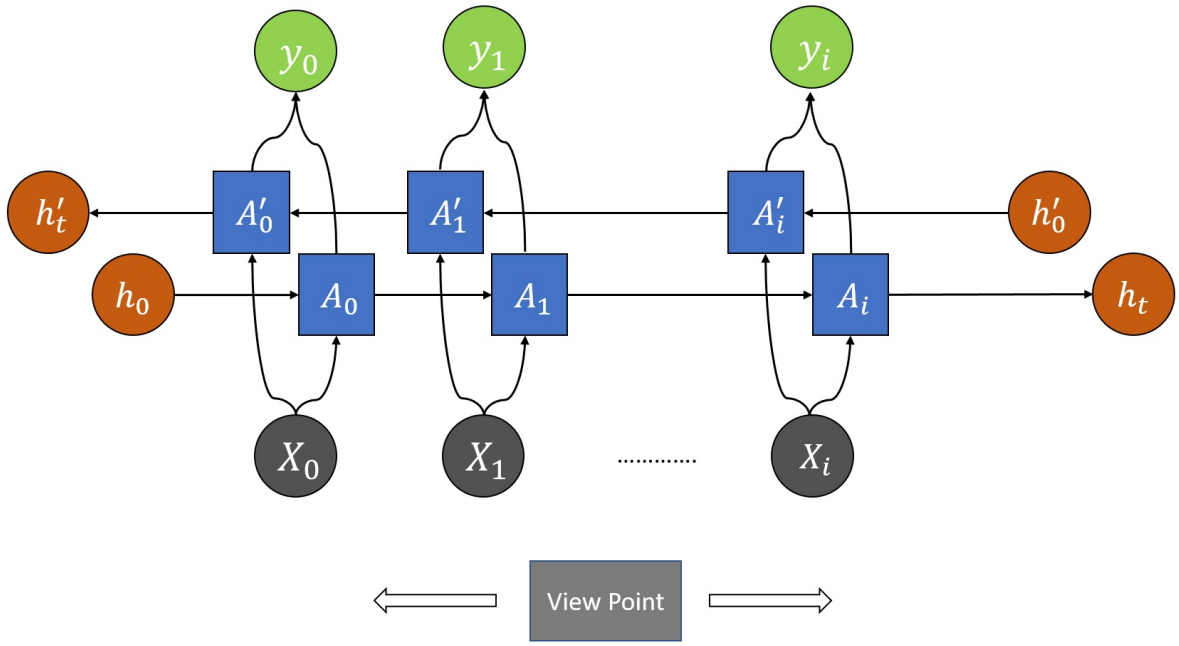


Figure 2.1: Structure and Viewpoint of Bidirectional RNN network.

illustrated in Fig. 2.1, a B-LSTM model has the viewpoint at the middle of the sequence from where the complete sequence can be considered.

Problem Statement Fault Classification

The problem for fault detection and classification can be summarized as a supervised learning problem where given labelled data samples $(\mathbf{x_t}, \mathbf{f_t^i})$ from a time series data set of a process denoting the faulty and normal operating conditions. In the above notation, $t = 0, \ldots T$ denotes the sequence number, i is the type of fault, and f^0 indicates normal operation. A B-LSTM model, $g(\cdot)$ is trained to minimize the negative log-likelihood loss function L as

$$\min_{\theta} L_\theta(\mathbf{x}, \mathbf{f}) = -\sum_{t=0}^{T} \mathbf{f_t} log(g_\theta(\mathbf{x_t})), \tag{2.14}$$

where θ are the trainable parameters of the model.

Data Preprocessing for Deep Recurrent Architectures and Efficient Training

This thesis proposes a novel data preprocessing technique for efficiently training the Deep RNN models. The overall methodology helps keep the model complexity under check and create a better discriminative representation of the training dataset. The methodology can be explained in a two-step procedure as follows:

1. **Sequence Generation:** Fig. 2.2 illustrates an example of time-series signals from two independent classes, i.e. No fault and Faulty. This approach can be, however, extended to any number of classes. From the raw input signal, sequences of length

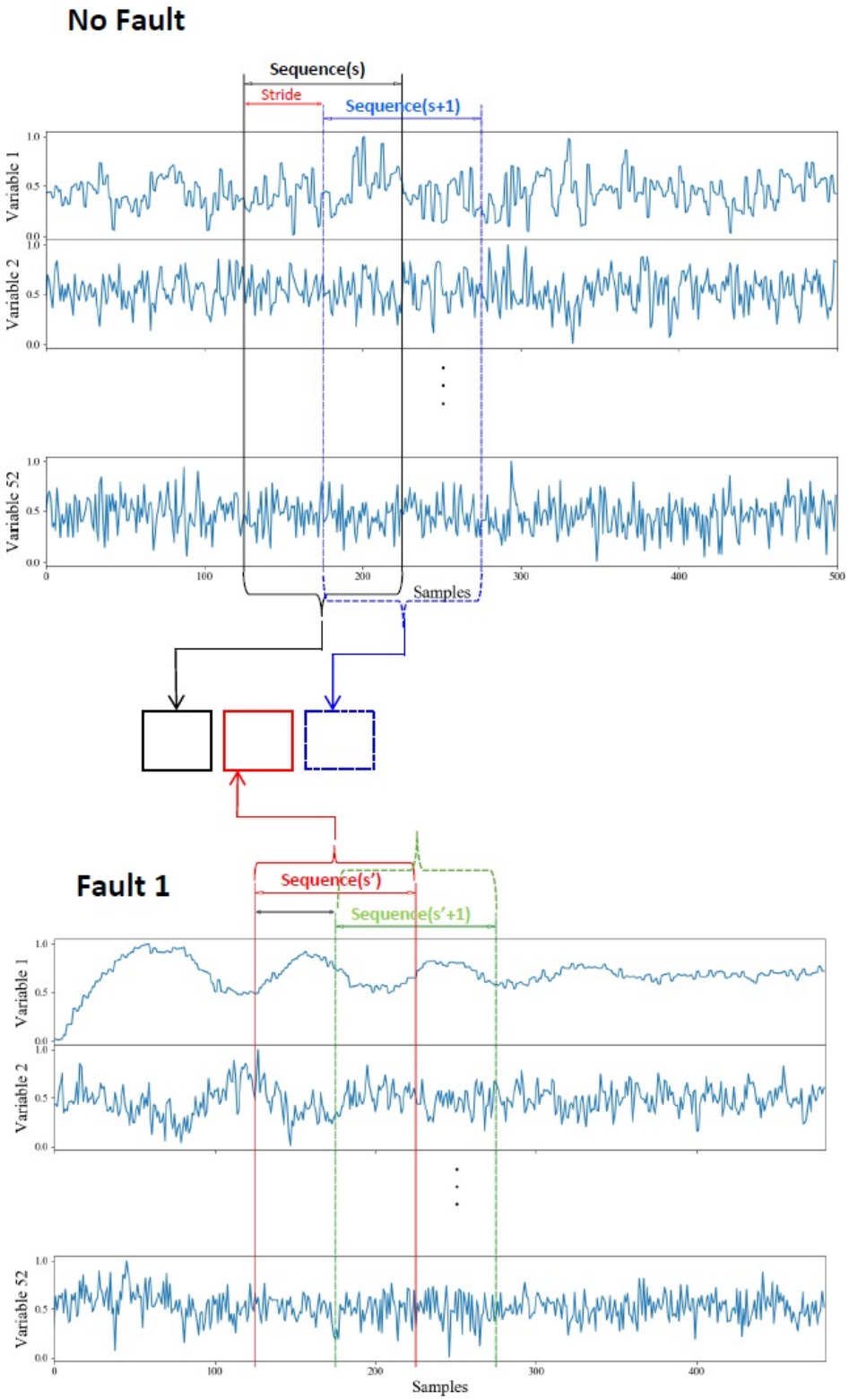
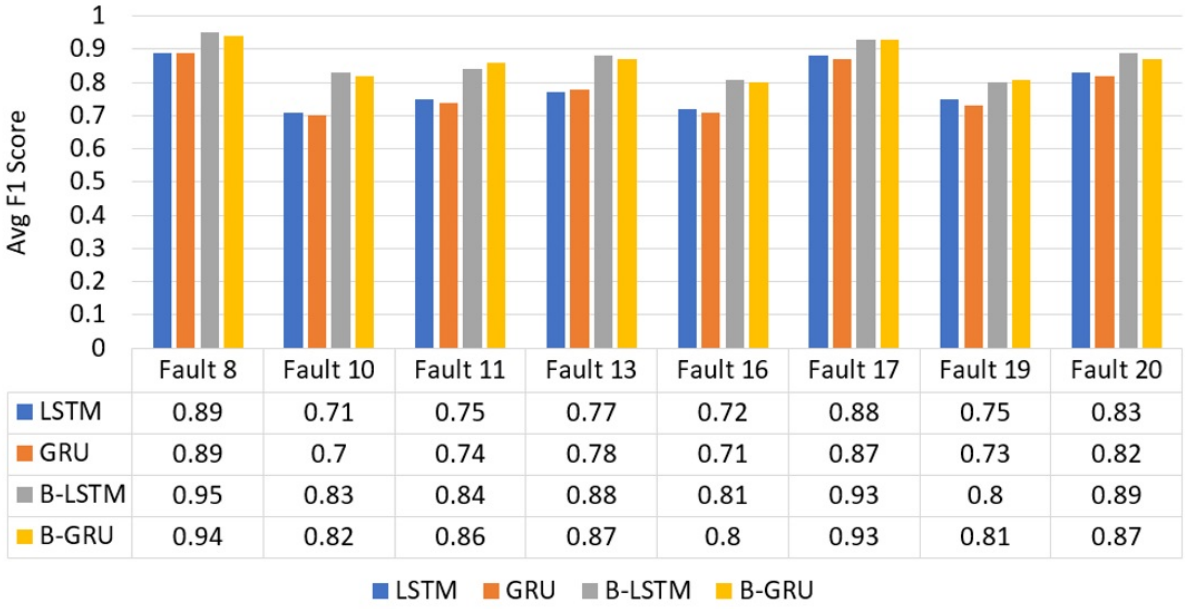


Figure 2.2: Sliding Window Approach for Data Representation



	Fault 8	Fault 10	Fault 11	Fault 13	Fault 16	Fault 17	Fault 19	Fault 20
LSTM	0.89	0.71	0.75	0.77	0.72	0.88	0.75	0.83
GRU	0.89	0.7	0.74	0.78	0.71	0.87	0.73	0.82
B-LSTM	0.95	0.83	0.84	0.88	0.81	0.93	0.8	0.89
B-GRU	0.94	0.82	0.86	0.87	0.8	0.93	0.81	0.87

Figure 2.3: Comparison of all RNN architectures for Faults 8, 10, 11, 13, 16, 17, 19 and 20

T (a hyperparameter) from both the multivariate raw signals are extracted. In the Fig. 2.2, two sample sequences for each class dataset are denoted by sequence s and sequence $s+1$ in the No fault case and the sequence s' and sequence $s'+1$ in the Fault 1 case. This process of dividing the raw input signals into sequences is performed iteratively over the complete training samples. This step leads to the maximum use of information in the training since each time step is used multiple times to approximate the trend in the input space.

2. **Dataset Restructuring:** The extracted sequences in the previous step are aggregated alternatively from the different classes available in the dataset for dataset restructuring. This alternative stacking is represented by the black, red and blue square boxes in Fig. 2.2. This step helps the model learn a discriminative representation between the different classes due to the sequential information flow about the difference in each class. Additionally, for longer sequence lengths, the model can see both the class information in one sequence.

Experimental Results and Comparison Study on the Tennessee Eastman Process with Deep Recurrent Neural Network

The TE [73] process is a benchmark dataset for evaluating fault detection and classification systems. The process consists of 21 pre-programmed process faults as proposed in [74]. The goal of the experiments is to analyse the fault detection and classification capabilities of the proposed Deep RNN architectures using the F_1 score as an appropriate evaluation metric. The F_1 score [75] is chosen in this thesis since it considers an even balance for the false positive and false negative classified samples. Fig. 2.3 illustrates the better generalization capabilities of the B-LSTM models compared to the other architectures. It must be noted here that both the bidirectional architectures perform better

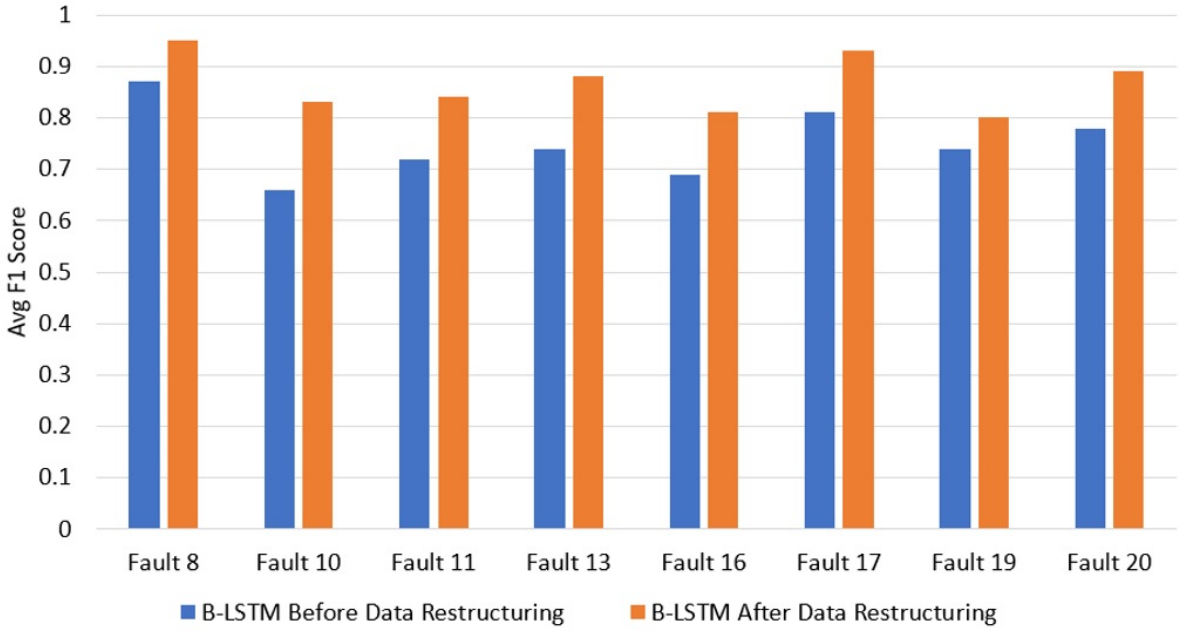


Figure 2.4: B-LSTM Comparison before and after data partitioning for Medium Faults

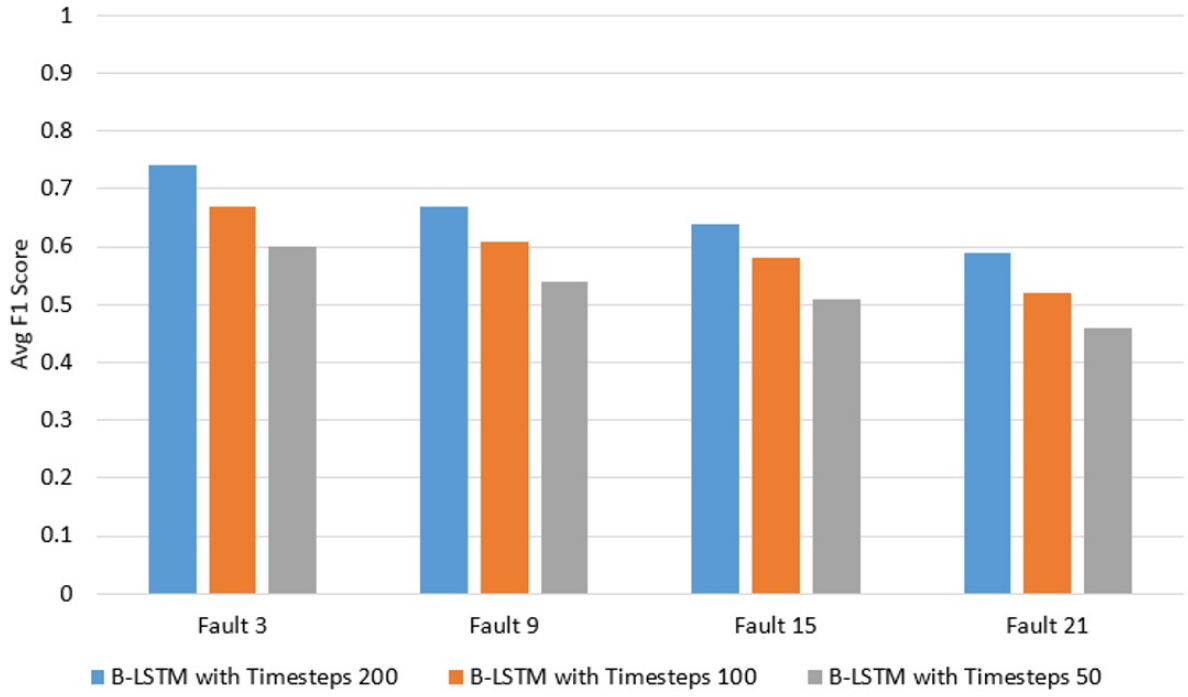


Figure 2.5: Effect of Sequence length on the Model's performance

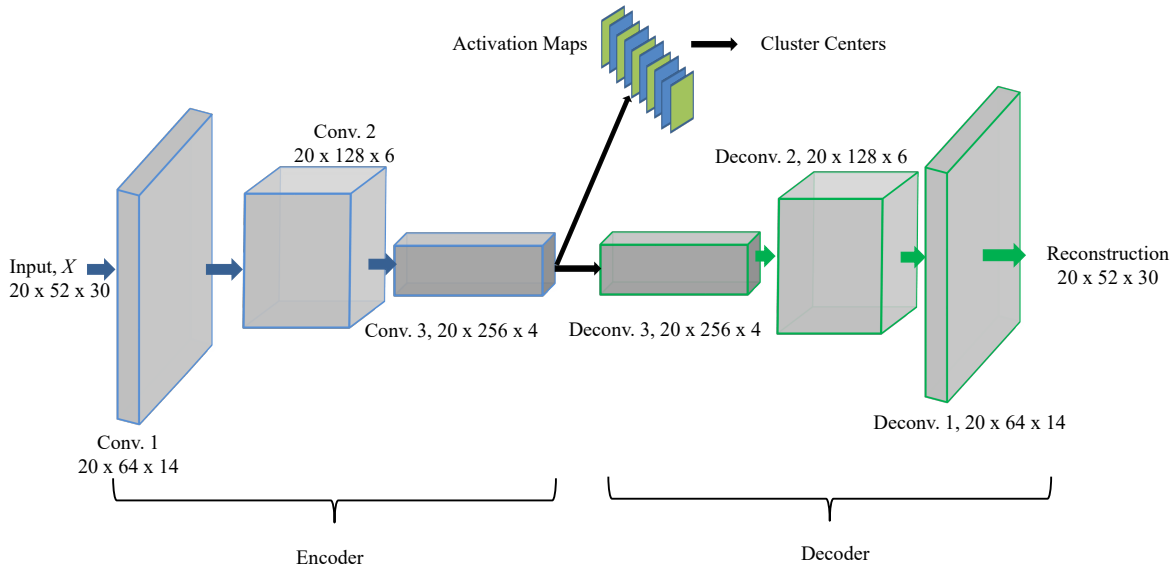


Figure 2.6: Proposed architecture of the clustering augmented deep autoencoder for anomaly detection.

than their unidirectional counterparts, strongly supporting the proposed hypothesis. The architecture of all the models is the same, with an input sequence length $T = 200$ and a hidden size of 64 neurons. To illustrate the improvement in the performance of the proposed methodology, Fig. 2.4 shows the drastic improvement in some of the fault cases of the TE process. Similar behaviour was also observed in the other fault cases. A detailed comparison can be found in the publication I.

In addition to the data restructuring, a larger sequence length yielded far better results than the models with a shorter sequence length. This was especially the case in incipient fault cases as shown in Fig. 2.5, where models with input time-step 200 performed superior as compared to the models with input time-steps of 100 and 50.

2.1.3 Anomaly Detection with Clustering augmented Deep Convolutional Neural Networks

The proposed architecture for Deep Convolutional Clustering Algorithm (DCCA) is shown in Fig. 2.6. The architecture consists of 3 convolution layers at the encoder and 3 deconvolution layers at the decoder with their respective activation functions. Since the bottleneck representation is what the decoder uses for reconstructing the input, it is imperative to force this representation to be as discriminative as possible for the anomalous and non-anomalous data samples. The clustering module is integrated at the bottleneck representation of the encoder to allow for additional discriminative representation. The size of the tensors shown in the image follows the naming convention as $(Batch - Size \times Number - of - Input - Channels \times Sequence - Length)$. The convolution layers incorporate the 1D convolution operation as the input to the system is a

multivariate time-series signal. The multivariate time series signal $\{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_T}\}$ where $\mathbf{x_i} \in \mathbb{R}^m$ is the input for the anomaly detection task, with m denoting the number of variables and T the length of the signal. The convolution operation is performed on the part of the input, also commonly known as the receptive field. The size of the receptive field is denoted as $n_r \times m$, which strides over the input $T \times m$ sequences, accounting for each of the variables. The pth convolution 1D kernel in the first layer can be denoted with a 2-dimensional tensor as $\mathbf{K^{(p)}} = [k_{i,j}^{(p)}] \in \mathbb{R}^{n_r \times m}$. The weights in these convolution 1D kernels are adjusted during the training process with the backpropagation algorithm. The indices i, j denote the dimension along the time and variable axis, respectively. Formally, the convolution 1D operation can be summarized as follows:

$$\begin{aligned} \mathbf{h_{i,p}} = (x * k)_i = \sum_{g=1}^{n_r} \sum_{f=1}^{m} \mathbf{x_{i+g-1,f}} \cdot \mathbf{k^p_{g,f}} \\ \forall i \in \{1, \ldots, T - n_r + 1\} \\ \forall p \in \{1, \ldots, d_{q+1}\}, \end{aligned} \tag{2.15}$$

where $\mathbf{h_{i,p}}$ denotes the output of the $(i)^{th}$ receptive field and the pth convolution kernel, $x_{i+g-1,f}$ are the elements in the receptive field of the input variable, $k_{g,f}$ is the convolution kernel and d_{q+1} denotes the number of convolution kernels in the given layer.

Problem Statement Anomaly Detection

The problem can be stated as follows: The goal is to train a Convolutional AE structure $f_\theta(\mathbf{x})$ so that the learned latent representation z can best distinguish between normal z_{no} and anomalous behaviour z_{ano}. The aim is to find an optimal separation between normal and anomalous data using only unlabelled data.

Top-K Deep Convolutional Clustering Algorithm

The idea behind the Top-K DCCA algorithm is to force the latent representation of the input data to be according to the anomaly detection downstream task that needs to be performed. Therefore, the overall latent space $\mathcal{Z}$, is split into two subsets, i.e. $\mathcal{Z}_c \subseteq \mathbb{R}^{n_c}$ and $\mathcal{Z}_r \in \mathbb{R}^{n_{rec}}$ which are termed as clustering and reconstruction friendly latent variables respectively. The optimization of the following cost function calculates the cluster centres:

$$\begin{aligned} \min_{M_j \in \mathbb{R}^{n_c \times k}, s_i \in \{0,1\}^K} \sum_{i=1}^{N} ||\mathbf{z_j^i} - \mathbf{M_j s_i}|| \\ \text{s.t.} \quad \mathbf{1}^T s_i = 1 \ \forall i, \\ \forall z_j \in \mathcal{Z}, \end{aligned} \tag{2.16}$$

where the columns $m_{k,j}$ in the matrix $\mathbf{M}$ denotes the k^th cluster center in the n_c-dimensional space and s_i is the cluster assignment of the i^{th} data points latent representation. The splitting criterion is the K-means algorithm wherein the cluster centres with the maximum Euclidean distance are chosen to identify the Top-n_c latent variables forming the set $\mathcal{Z}_c$. Specifically, for an anomaly detection task two clusters are assumed with centres defined as $m_{no,j}$ and $m_{ano,j}$ indicating normal and anomalous operation, respectively.

Subsequently, the Euclidean distance between the cluster centers are is calculated as

$$\max_{j \in \mathcal{Z}_c} d(m_{no,j}, m_{ano,j}) = \max_{j \in \mathcal{Z}_c} ||m_{no,j} - m_{ano,j}||^2, \tag{2.17}$$

to identify the Top-n_c latent variables which have the maximum distance between them, forming the set $\mathcal{Z}_c$.

End-to-end training of the clustering augmented AE

As illustrated in Fig. 2.6, the encoder f_θ and the decoder g_ψ are trained in an end-to-end based manner via the gradient descent algorithm using the reconstruction loss. Specifically, the reconstruction can be formally stated as

$$L_{\mathrm{AE}}(\theta, \psi) = \sum_{i=1}^{N_B} ||\mathbf{x^i} - g_\psi(f_\theta(\mathbf{x^i}))||_2^2, \tag{2.18}$$

where N_B is the minibatch size. In addition to the reconstruction loss, to determine the split between the clustering and the reconstruction-friendly latent variables, the clustering loss defined as

$$L_{j,\mathrm{CL}}(\theta) = \sum_{i=1}^{N_B} ||\mathbf{z_j^i} - \mathbf{M_j s^i}||_2^2 = \sum_{i=1}^{N_B} ||f_{j,\theta}(\mathbf{x^i}) - \mathbf{M_j s^i}||_2^2, \tag{2.19}$$
$$z_j \in \mathcal{Z}_c,$$

is also fed back, which subsequently affects the trainable parameters of the encoder too. Therefore, the total loss to training the Convolutional AE is

$$L \;=\; \alpha \sum_{j=1}^{z_j} L_{j,\mathrm{CL}}(\theta) \;+\; (1 \;-\; \alpha) L_{\mathrm{AE}}(\theta, \psi) \tag{2.20}$$

where the value of α ranges between 0.6 to 1 which was found empirically. The encoder and decoder parameters are denoted as $\chi = (\theta, \psi)$. The whole training process is divided into two steps. The first step, termed pre-training, comprises training with only the reconstruction loss, i.e. with the value of α set to 0. For the second step, a fixed value of α is set, and the Convolutional AE is trained with the total loss as in Eq. (2.20). Furthermore, the clusters are updated only in a certain interval of update steps. This *Cluster Update Interval C* is a hyperparameter which is set before the training. The complete algorithm for the training is defined in Algorithm 1, where a model is trained for N epochs.

Algorithm 1 Top-K Deep Convolutional Clustering Algorithm

procedure INITIALIZATION(Perform N epochs over the data)
    P = Number of pre-training epochs
    C = Cluster update interval
    for epoch = 1 to P + 1 **do**
        Reconstruct the data, extract latent representation $f_\theta(x^i)$
        Compute gradients $\nabla \chi L^i$ with $\alpha = 0$
        Update network parameters χ
        if epoch = P + 1 **then**
            Perform *K-Means* by optimising the Eq.(2.16)
            Return centers $m_{no,j}$ and $m_{ano,j}$ and center assignments $M_j s_i$
            Rank latent representation layer channels by Eq. (2.17)
            Return Top K ranked channels
    for epoch = P + 1 to N **do**
        Reconstruct the data, extract latent representation $f_\theta(x^i)$
        Compute gradients $\nabla \chi L^i$ with $\alpha = 0$
        Update top K ranked channel parameters
        Zero the gradients
        Compute gradients $\nabla \chi L^i$ with α
        Update rest of the channel parameters
        if epochmodC = 0 **then**
            Perform *K-Means* by optimising the Eq.(2.16)
            Return centers $m_{no,j}$ and $m_{ano,j}$ and center assignments $M_j s_i$
            Rank latent representation layer channels by Eq. (2.17)
            Return Top K ranked channels

Experimental Results on the Tennessee Eastman Process with Clustering augmented Deep Convolutional Neural Networks

The proposed DCCA and Top-K DCCA algorithm is tested on the benchmark TE process for evaluating their applicability to the anomaly detection task. The F_1 score is chosen as the evaluation metric to balance the predicted false positive and false negative samples. The proposed methodology showed superior fault detection performance compared to pure unsupervised learning-based k-means augmented CNN methodology. Previous works [76] are followed in dividing fault classes into subgroups based on how challenging the faults are to detect. The 21 faults are divided into easy, medium, and hard-to-detect faults.

The proposed architecture is compared to the Vanilla model in terms of the F_1 score, with the results shown in Fig. 2.7. The proposed architecture performs much better than the baseline model for all fault categories in both the 2-layer and 3-layer configurations. The 3-layer configuration performs slightly better than the 2-layer one in all cases, so it is used for the rest of the analysis. The t-SNE [77] plots in Fig. 2.8 show some of the activation maps to better visualize the discriminative capability in the latent representation. The model has learned that there are two distinct regions, normal and anomalous, and the boundaries of the two regions can be seen clearly. After that, the results of the semi-supervised training setup are presented, where the encoder of the Top-K DCCA architecture is pre-trained with unlabelled data, as per Algorithm 3. Two fully connected

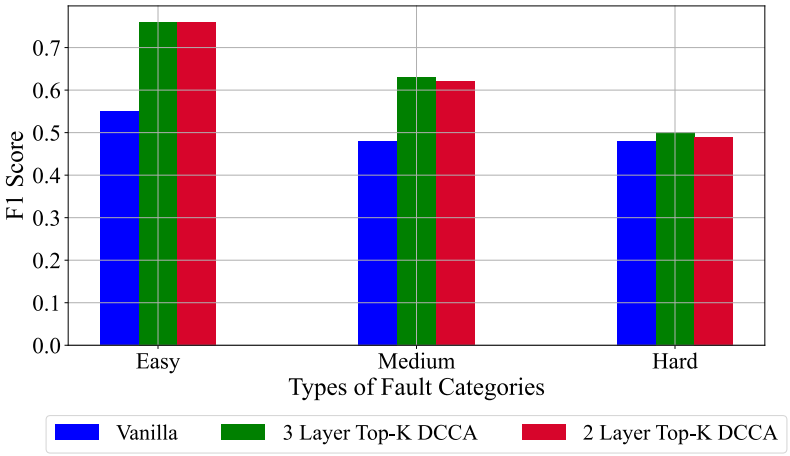
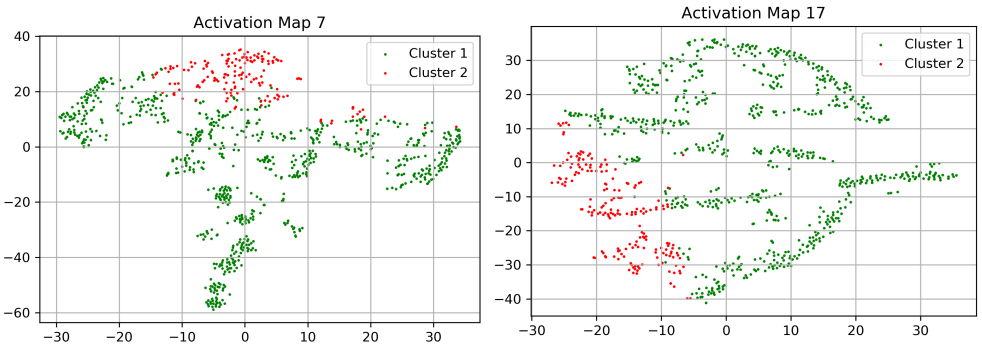
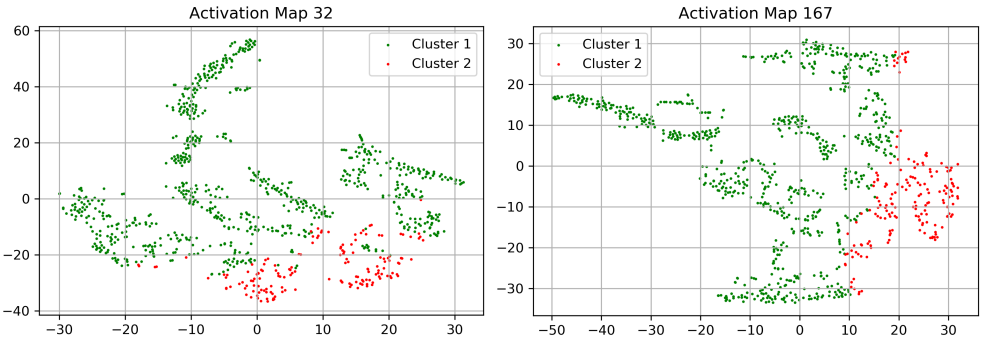


Figure 2.7: F_1 score obtained by the Vanilla and the Top-K DCCA approach with different layers for anomaly detection task in an unsupervised learning setup



(a) Activation Map 7

(b) Activation Map 17



(c) Activation Map 32

(d) Activation Map 167

Figure 2.8: t-SNE Visualization of a sample of the activation maps with Top-K DCCA Approach on Tennessee Eastman Data

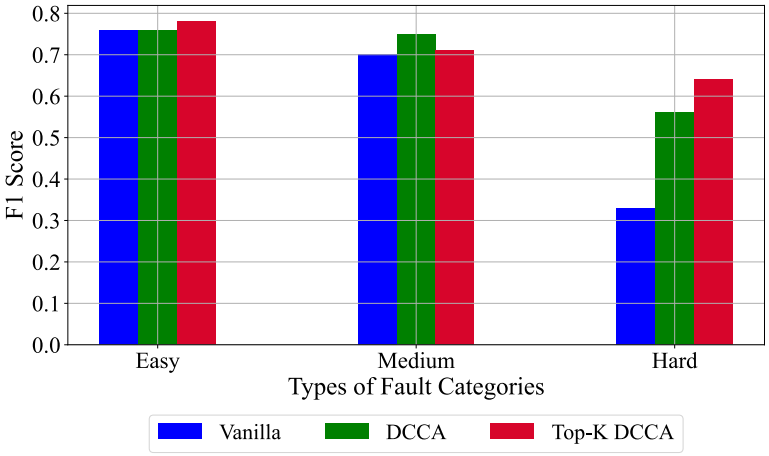


Figure 2.9: F_1 score obtained by the Vanilla, DCCA and the Top-K DCCA approach for the anomaly detection task in a semi-supervised learning setup

layers with 300 and 2 hidden units, respectively, are trained in a supervised manner using labelled data. The convolutional encoder's weights and biases are frozen during the fine-tuning stage.

The average F_1 score for the Vanilla, DCCA, and Top-K DCCA approaches on different fault categories in a semi-supervised learning setup is shown in Fig. 2.9. The proposed Top-K DCCA approach outperforms the other two models in the Easy and Hard fault categories. The standard DCCA only marginally performs better in the medium category; however, the proposed methodology works better than the Vanilla model in all three fault categories.

2.1.4 Contribution

This section's contribution, based on the publications I and II, are first, a novel system architecture with bidirectional LSTM networks for fault detection and classification based on raw sensor data. Secondly, a novel approach is presented for unsupervised training-based anomaly detection focusing on time series data sets. The fault classification system proposes novel sequence generation and data restructuring procedures, enabling Deep RNN models to learn discriminative features for fault classification tasks efficiently. The proposed methodology outperforms unidirectional and other standard architectures reported in the literature.

The anomaly detection approach combines a deep 1D-CNN-based AE with a clustering loss on a subset of the latent variable space, which increases the discriminative power within the latent variable space without sacrificing too much reconstruction performance on the data set. The approach is end-to-end trainable by backpropagating the clustering and the reconstruction objective through the network. The approach is tested on the TE benchmark data set with satisfactory results.

2.2 Remaining useful estimation with Generalized Dilation and Temporal Attention Neural Networks

This section presents two novel algorithms for remaining useful estimation of industrial components by analysing multivariate time series datasets. The two approaches are Generalized Dilation CNN and Shared Temporal Attention Transformers. The proposed approaches are tested on two benchmark datasets, the PRONOSTIA Bearing Dataset and the C-MAPSS Aircraft Engine Dataset, both challenging for RUL prediction. The content of this section is based on publications in Chapter III and Chapter IV.

2.2.1 Introduction

Combining condition monitoring and predictive maintenance with data-driven tools and techniques can significantly optimise production processes. Consequently, data-based maintenance can be used to predict the availability or degradation of an asset based on multivariate time series analysis approaches. Neural networks, specifically CNN [78], and Transformer Networks [79] are particularly effective for this purpose. This chapter proposes two novel Deep Learning architectures for multivariate time series analysis. The first architecture is based on the concept of dilation in CNN. It can learn to extract features over a prolonged time horizon and ignore certain input features. Consequently, the proposed Generalized Dilation (GD) layer allows for arbitrary dilation structures and can learn which part of the input is relevant for the prediction task and which is not.

The second architecture proposed in this chapter is the Shared Temporal Attention Transformer (STAT) and the Feature-Represented Shared Temporal Attention Transformer (FeaR-STAT) for predicting sequence RUL values instead of a single RUL for a specific cycle. The proposed transformer architectures consist of a Shared Temporal Attention methodology along with Split-Temporal Multi-Head attention and Split-Feature Multi-Head attention blocks to enforce a higher emphasis on the sequence of the encoder input signal. Both of these architectures, the Generalized Dilation Convolutional Neural Network (GDCNN) and the Transformer architecture, are applied to the C-MAPSS aircraft engine data set [80] to underline their prediction performance.

2.2.2 Generalized Dilation Neural Networks

Convolutional layers are similar to standard filter algorithms in signal processing, where the filters are trainable. 2D convolutions are appropriate when there is a spatial relationship between the input channels, for example, in images. 1D convolutions are appropriate when there is no such relationship, for example, in time-series data. Therefore, in 1D convolution, the kernel ranges over the entire sensor channel size. Considering a multivariate time series signal for RUL estimation, $\{x_t\}_{t=1}^{T}$ with $x_t \in \mathbb{R}^m$ where m is the number of sensors measuring the state of an asset and T the length of the time series. The standard convolution kernel yields

$$\mathbf{h_i} = (\mathbf{X} * \mathbf{K})_i = \sum_{f=0}^{n_r-1} x_{i+f} \cdot k_{i+f} + b, \tag{2.21}$$

where $\mathbf{h_i}$ denotes the output of the $(i)^{th}$ receptive field in the input, x_{i+f} are the elements in the receptive field of the sequence, k_{i+f} are the elements in the convolution kernel, b denotes the bias for the convolution kernel. The size of each convolution yields $\mathbf{K} \in \mathbb{R}^{n_r \times m}$. The weight-sharing capabilities of the convolution kernel are severely weakened when the number of sensors measuring the state of an asset is high. Furthermore, modelling information over a long time horizon requires a deep network increasing parameters. Consequently, dilated convolution [81] mitigates the above drawbacks by expanding the receptive field size of a convolution kernel. The 1D dilated convolution operation yields

$$\mathbf{h_i} = (\mathbf{X} *_\mathbf{l} \mathbf{K})_i = \sum_{f=0}^{n_r-1} x_{i+l*f} \cdot x_{i+f} + b \tag{2.22}$$

where l is the dilation factor. The dilation factor and its structure are kept constant during training in the original study. Furthermore, standard dilations cannot express several dilation structures within a single receptive field. It would be advantageous to eliminate these restrictions, especially for time series data analysis. Because of this, this thesis proposes additional flexibility by making the dilation structure trainable and flexible throughout the receptive field. This concept may be even more comprehensive to accommodate broader patterns in the receptive field.

It is conceivable to think of the dilation process as a standard convolution operation with a receptive field size $n_d \times n_d$ with $n_r \leq n_d$. This results in a convolution weight matrix $\mathbf{W} \in \mathbb{R}^{n_d \times n_d}$, where the elements of $(n_d \times n_d) - (n_r \times n_r)$ are fixed to zero so that the active weights (weights unequal zero) add to $n_r \times n_r$. To do this, vectors $\psi_\mathbf{l} \in \{0,1\}^{n_d}$ and $\psi_\mathbf{r} \in \{0,1\}^{n_d}$ together with matrices $\mathbf{\Psi_l}$ and $\mathbf{\Psi_r}$, are defined in such a way that

$$\text{diag}(\mathbf{\Psi_l}) = \psi_\mathbf{l}, \quad \text{diag}(\mathbf{\Psi_r}) = \psi_\mathbf{r}. \tag{2.23}$$

The new convolution weight matrix is then defined as

$$\tilde{\mathbf{W}} = \mathbf{\Psi_l} \cdot \mathbf{W} \cdot \mathbf{\Psi_r}, \tag{2.24}$$

which offers a generalization to the dilation layer that was briefly discussed in part before. It should be noted that arbitrary dilation-like patterns in the weight matrix $\tilde{W}$ may be produced by pushing the total number of ones per ψ_l and ψ_r to n_r while also arbitrarily setting the components of ψ_l and ψ_r to zero and one, respectively. Formally, this results in imposing constraints

$$\psi_\mathbf{l}^\mathbf{T} \cdot \mathbf{1} \leq n_r, \quad \psi_\mathbf{r}^\mathbf{T} \cdot \mathbf{1} \leq n_r, \tag{2.25}$$

where $\mathbf{1}$ denotes the all-one vector.

The dilation procedure can be further enhanced by creating a new masking matrix $\mathbf{\Psi} \in \{0,1\}^{n_d \times n_d}$ and define a new convolution weight matrix as

$$\tilde{\mathbf{W}} = \mathbf{W} \odot \mathbf{\Psi} \tag{2.26}$$

where the $\odot$ symbol stands for element-wise multiplication. The constraint on the new masking matrix has to be modified as

$$\mathbf{1}^T \cdot \mathbf{\Psi} \cdot \mathbf{1} \leq n_r^2. \tag{2.27}$$

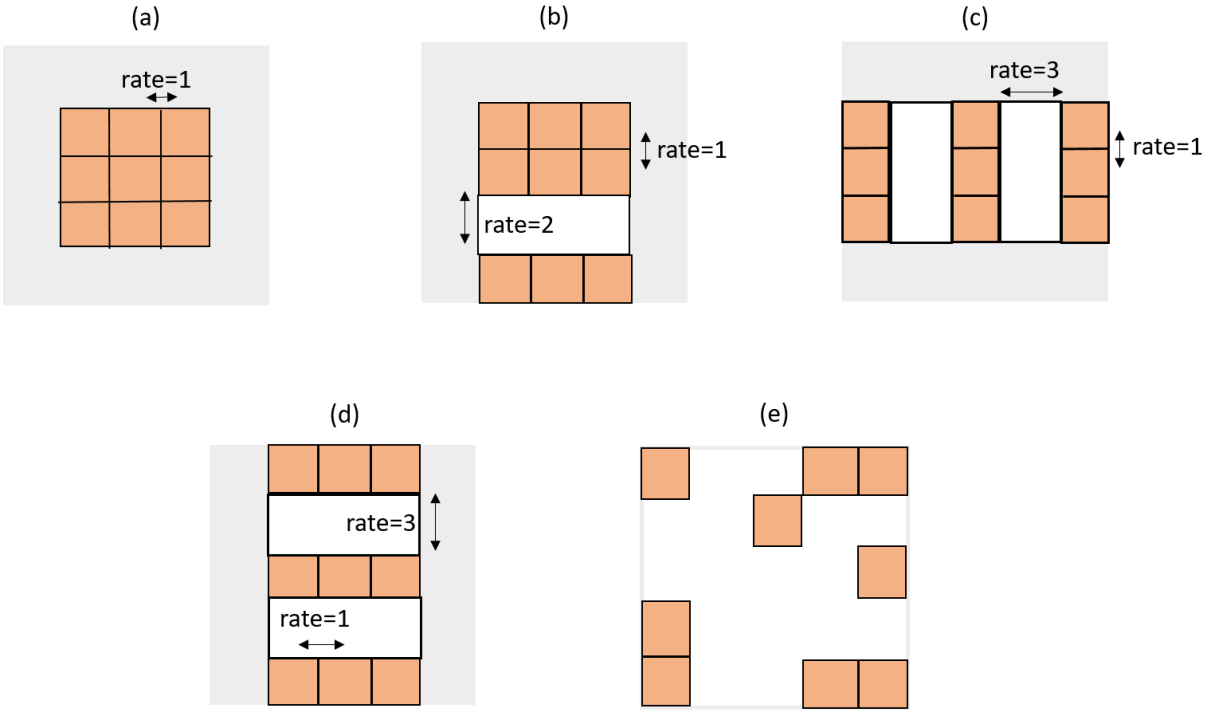


Figure 2.10: Different configurations of the parameters: (a) Original convolution, (b) Dilation with varying dilation rates; (c) Dilation in horizontal dimension only; (c) Dilation in vertical dimension only; (e) Arbitrary dilation kernel.

There are generalized dilation-like patterns possible with the above reparameterization of the convolution weight matrix $\tilde{W}$. Fig. 2.10 provides samples of a few of the various potential patterns. By setting the parameters of Ψ as

$$\mathbf{\Psi} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{bmatrix}, \tag{2.28}$$

it is possible to produce arbitrary dilation patterns, as depicted in Fig. 2.10 (e)

End-to-end Training of Dilated Neural Networks

Since binary variables have to be optimized for finding an optimal masking weight matrix and vector described in the previous section, the learning issue becomes combinatorial. This combinatorial learning problem does not directly permit end-to-end gradient-based training. To overcome this issue, a soft binary variable is generated by applying a sigmoidal activation function resulting in continuous vectors $\tilde{\psi}_l, \tilde{\psi}_r \in \mathbb{R}^{n_d}$ and matrices $\tilde{\Psi} \in \mathbb{R}^{n_d \times n_d}$. Formally, the softening of binary parameters is done such that

$$\tilde{\psi}_{\mathbf{l}} = \sigma(\psi_{\mathbf{l}}), \quad \tilde{\psi}_{\mathbf{r}} = \sigma(\psi_{\mathbf{r}}), \quad \tilde{\mathbf{\Psi}} = \sigma(\mathbf{\Psi}). \tag{2.29}$$

As training progresses, the sigmoid function trends toward its boundaries of $(0, 1)$. The masking weights are bound to the interval $(0, 1)$ using the sigmoidal function because it

is necessary for choosing input samples from the multivariate time-series sensor inputs. Next, the additional constraints on the masking parameter $\tilde{\Psi}$ provided in Eq. (2.25) and Eq. (2.27) must be taken into consideration. This work proposes two approaches for implementing these constraints: a.) a differentiable barrier functions approach, which contributes to the loss function only if the constraints are not satisfied, and b.) $Top-K$ sampling approach uses integer values to assign integer values to the binary masking parameters.

Barrier Function: The overall loss of the model is defined as $L_s(\omega, \tilde{\Psi}) + L_b(\tilde{\Psi})$ where L_b is the barrier function loss and $L_s(\omega, \tilde{\Psi})$ is a standard loss like cross-entropy loss for classification or root mean squared error for regression tasks. One may approximate the barrier loss directly as the gradient of the barrier loss $\nabla_\Psi L_b$ since the barrier loss only applies to the masking vectors and parameters, $\tilde{\Psi}$ and $\tilde{\psi}_l, \tilde{\psi}_r$ respectively. Formally, the gradient of the barrier loss can be defined as

$$\nabla_{\tilde{\Psi}} L_b = b_c(\tilde{\mathbf{\Psi}}) + b_r(\tilde{\mathbf{\Psi}}) + b_a(\tilde{\mathbf{\Psi}}), \tag{2.30}$$

where b_c, b_r and b_a stand for various barrier functions that were derived using varying degrees of penalties for satisfying the constraint. The barrier functions used in this work are

$$b_c(x) = \left(e^{\alpha_1 \cdot (x - n_r)} - \alpha_2 \cdot (x - n_r)\right) - \alpha_3, \tag{2.31}$$

$$b_r(x) = \max(e^{\alpha_1 \cdot (x - n_r)} \cdot \alpha_2 \cdot (x - n_r), \alpha_3), \tag{2.32}$$

$$b_a(x) = \max(e^{\alpha_1 \cdot \left(x^2 - {n_r}^2\right)} \cdot \alpha_2 \cdot \left(x^2 - {n_r}^2\right), \alpha_3), \tag{2.33}$$

where we test several values for α_1, α_2 and α_3 the barrier function as shown in Fig. 2.11 for the barrier function $b_c(x)$. According to Fig. 2.11, the barrier function has a built-in propensity to yield greater penalties if the boundary condition $(x - n_r)$ is exceeded, independent of the parameters (α_1, α_2 and α_3). Therefore, if $\tilde{\Psi}_{ij}$ has a big sum of its components at the start of training, there will be greater penalties and consequently larger gradients. The gradient for end-to-end training of the masking parameter $\tilde{\Psi}_{ij}$ results in

$$\nabla_{\tilde{\mathbf{\Psi}}_{\mathbf{ij}}} = \frac{\partial L_s(\omega, \tilde{\mathbf{\Psi}})}{\partial \tilde{\mathbf{\Psi}}_{\mathbf{ij}}} + \nabla_{\tilde{\Psi}_{ij}} L_b. \tag{2.34}$$

The amount by which the masking parameters deviate from the constraint and their contribution to the classification or regression loss $L_s(\omega, \tilde{\Psi})$ are added together to form the gradient of the masking parameters.

Top-K Sampling: In this method, the top-K values or the maximum K values from the matrix $\tilde{\mathbf{\Psi}}$ are set to 1 and the remainder to 0. Specifically, the masking matrix's components are updated as follows:

$$\hat{\mathbf{\Psi}}_{\mathbf{ij}} = \begin{cases} 1, & \text{if } \tilde{\mathbf{\Psi}}_{ij} \in \mathcal{Y}_{max-K} \\ 0, & \text{else .} \end{cases} \tag{2.35}$$

where $\mathcal{Y}_{max-K}$ denoted the K max elements in the masking matrix $\tilde{\Psi}$. The masking matrix $\hat{\mathbf{\Psi}}$ is given integer binary values in the forward pass by doing the above operation. When updating the binary parameters or performing a backward pass, the gradient estimates

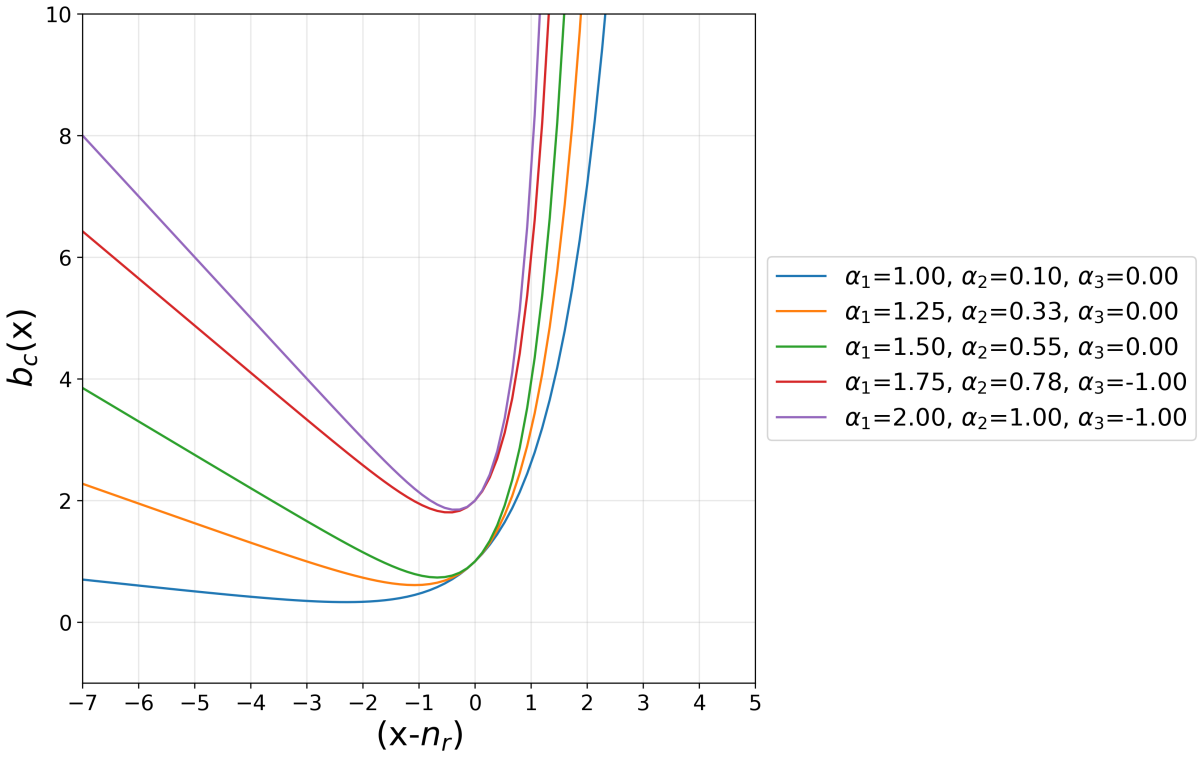


Figure 2.11: Barrier functions b_c with different parameters α_1, α_2 and α_3.

are used as if the forward pass had been performed using the masking forward pass Eq. (2.29), and we update the parameters using the gradient masking Eq. (2.34). In the forward pass, this results in integer binary values; however, the backward pass uses their sigmoidal values. The top-K operation has the benefit of requiring the network to choose the *exact* weights for the prediction job and loss computation. In contrast, the barrier function strategy would have chosen a scaled version of the weights because of the sigmoid function.

Experimental Results with Generalized Dilation Neural Networks

This section presents the experimental results achieved from the GDCNN architecture. The proposed model is tested on two datasets, namely the PRONOSTIA [82] dataset and the Commercial Modular Aero-Propulsion System Simulation (C-MAPPS) [80] dataset. PRONOSTIA is an experimental platform designed to produce accelerated degradation in ball bearings for testing and validating bearing diagnostics, prognostics and fault detection. The challenge is to take data from three different operating conditions and use it to predict the RUL of 11 bearings under the same conditions. There are six training cases, two for each operating condition. The Root Mean Square Log Error (RMSLE) is proposed in this thesis for RUL training since the large loss values cause the weights to be updated too much, which makes the training process unstable. The RMSLE loss yields

$$RMSLE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\big(\log(r_i+1) - \log(\widehat{r}_i+1)\big)^2}. \tag{2.36}$$

Table 2.1: Performance of proposed architecture on Bearing data set

Bearing Name	**Actual RUL (s)**	**Predicted RUL (s)**	**%Er**	**Score**
Bearing 1-3	5730	4086.31	28.68	
Bearing 1-4	339	174	48.67	
Bearing 1-5	1610	3004.42	-86.61	
Bearing 1-6	1460	1458.76	0.084	
Bearing 1-7	7570	2159.85	71.46	
Bearing 2-3	7530	6737.35	10.53	0.38
Bearing 2-4	1390	4245.71	-205.4	
Bearing 2-5	3090	3084.25	0.18	
Bearing 2-6	1290	2478.83	-92.16	
Bearing 2-7	580	4411.95	-660.68	
Bearing 3-3	820	783	4.51	

where r_i is the actual RUL and $\widehat{r}_i$ is the predicted RUL for the i^{th} data instance.

The Fig. 2.12 shows how well the GDCNN model predicts the RUL of a bearing. The model does an excellent job of predicting the RUL near the end of the bearing's life span when the bearing is about to fail. Fig. 2.13 shows the impact of sequence length on the loss for the test dataset and the training time required. It can be observed that if the sequence length is too small, then the model will not be able to capture all the valuable information as the historical data is too less. If the sequence is too long, it will take a long time to train the model. Therefore, a sequence length of 2560 samples provides an optimum result, with a further increase in sequence length resulting in over-fitting. The prediction performance of the proposed model on all the test bearings and the overall score per the standard scoring defined in the literature is summarized in Table 2.1.

The C-MAPSS dataset is a set of data from 21 sensors on three different types of engines in different stages of degradation. The dataset is split into four sub-datasets, FD001-FD004, each split into training and testing data. Each of the sub-datasets represents a different operating condition. Two different types of convolutional architectures for the C-MAPPS data to try to improve our predictions are presented in this thesis. The first convolutional architecture is the standard one-dimensional CNN architecture. The second architecture is called the shared kernel CNN-1D architecture. The motivation for the shared kernel approach is that the feature space for the C-MAPPS dataset is much larger than the bearing dataset, so the standard CNN-1D architecture would need to be more efficient. Two distinct training methods given in Sec. 2.2.2 to incorporate the restrictions in the GD training, in addition to the two different architectures, are presented below.

When the Top-100 elements from the receptive field are sampled as 1, and the remainder are sampled as 0, Top-K Sampling significantly improves prognostic performance. The integer binary values of the masking parameters benefit from this rigorous sampling. Fig. 2.14a and Fig. 2.14b show the distribution of initial and learnt masking parameters within the range of (0, 1) for the barrier function and Top-K sampling approaches, respectively. The figures show how the Top-K sampling strategy achieves hard limits instead of the Barrier function's soft constraints. The author believes that the Top-K approach's

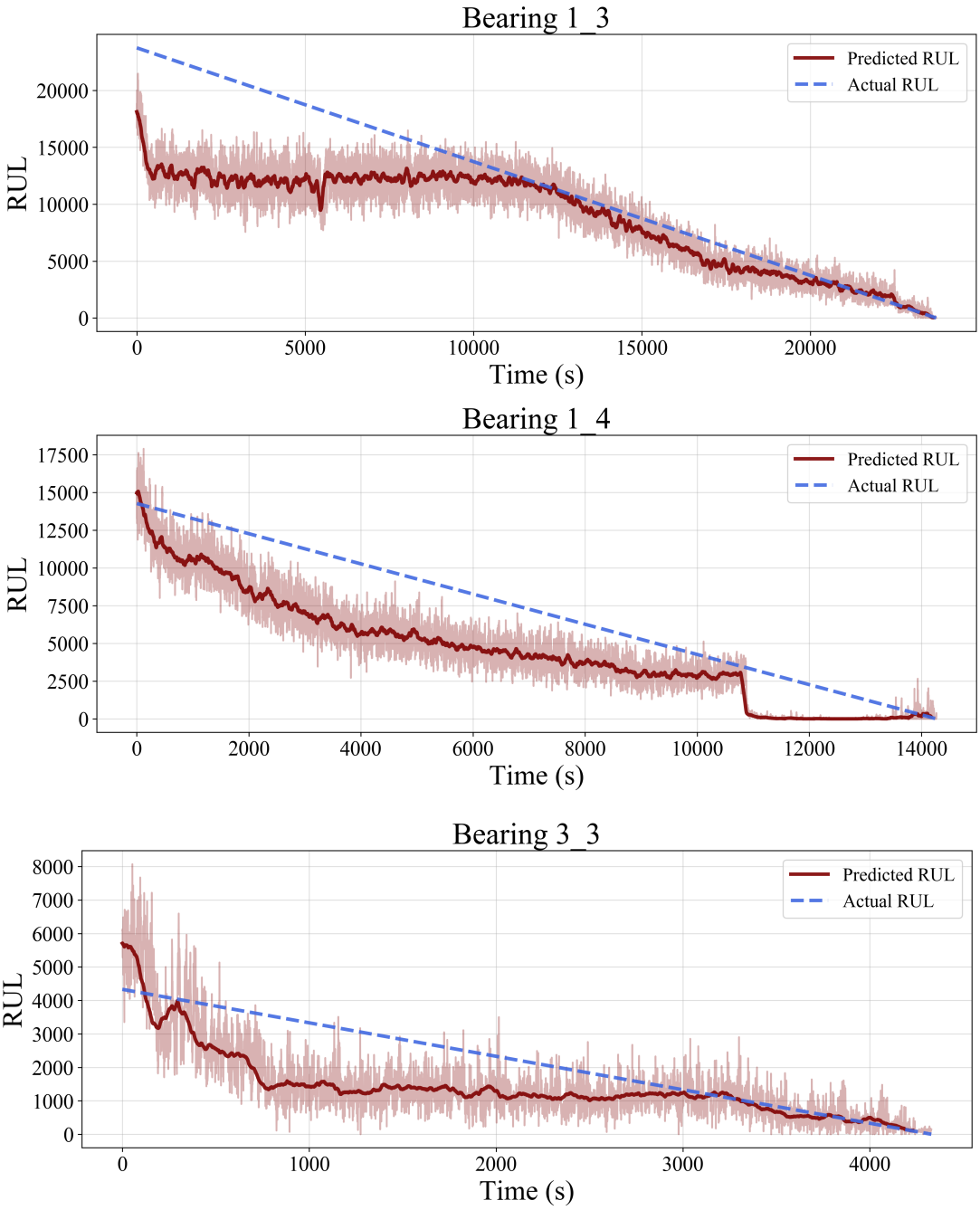


Figure 2.12: Qualitative Generalization Performance on the Bearing data set

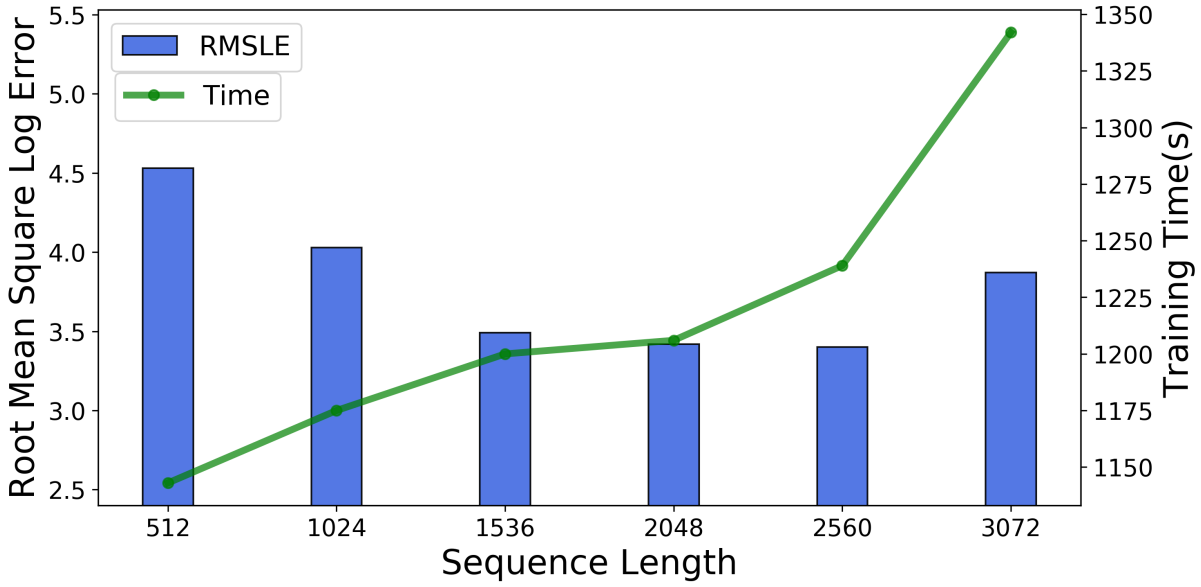


Figure 2.13: Effect of sequence length on prognostics performance and training time of operating condition-1 test dataset

Table 2.2: Performance comparison of the proposed architectures on C-MAPSS Dataset

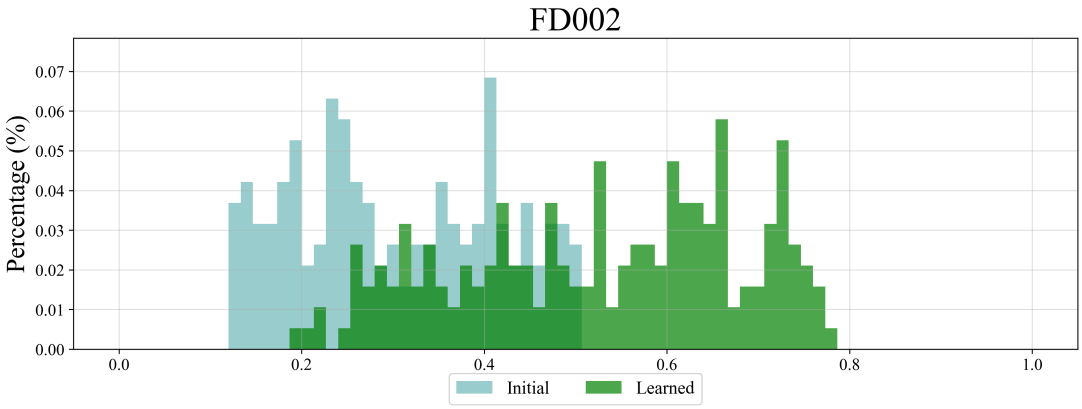
Method	FD001	FD002	FD003	FD004
Shared Kernel CNN-1D	317.7	11053	336.7	8122
Shared Kernel GDCNN-1D with Barrier	316.8	10273	304.4	7250
Shared Kernel GDCNN-1D with Top-K	274.6	10256.6	285.1	7213.8
GDCNN-1D with Top-K	267.8	10868.9	307.8	7111.1

more extraordinary performance is due to the strict constraint it imposes. As a result, only the Top-K sampling strategy was used in the GDCNN-1D tests. Fig. 2.15, Fig. 2.16, Fig. 2.17 and Fig. 2.18 show the prognostic performance of the proposed GDCNN-1D.

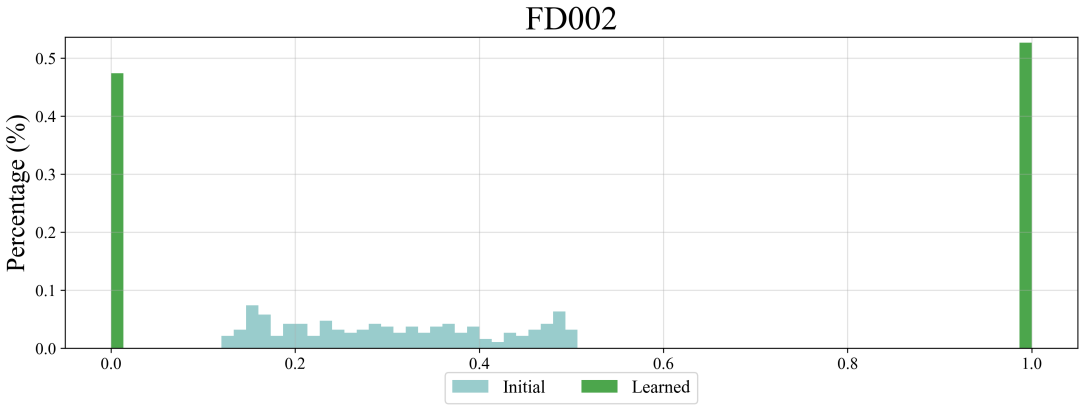
The prognostic performance of the proposed architectures for each sub-dataset using all the proposed techniques is presented in Table 2.2. It is worth noting that the combined score of all proposed architectures beats several previously published papers. It is clear that GDCNN techniques outperform ordinary CNN approaches and that the Top-K sampling approach for GDCNN performs better than its barrier function counterpart.

2.2.3 Temporal Attention-based Transformer Networks

The self-attention mechanism used by the transformer architecture suggested in [79] computes the significance of each input with the other inputs. To accomplish the intra-attention action, the architecture leverages parallel inputs rather than the sequential input approach used by RNN/LSTM structures. Self-attention necessitates transforming the raw input signal $\mathbf{X}$ into query, key and value $(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ matrices through three dif-



(a) Distribution of initial and learned masking parameters ($\sigma(\Psi)$) in the case of Barrier Function



(b) Distribution of initial and learned masking parameters ((Ψ)) in the case of Top-K Sampling

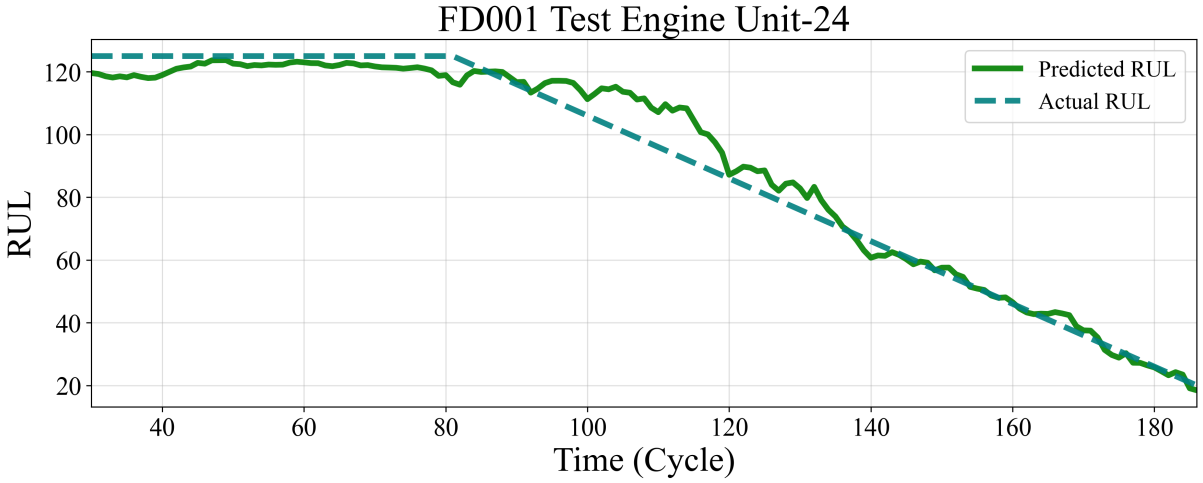


Figure 2.15: Prognostic performance from the GDCNN-1D architecture with Top-K sampling on a test engine from FD001

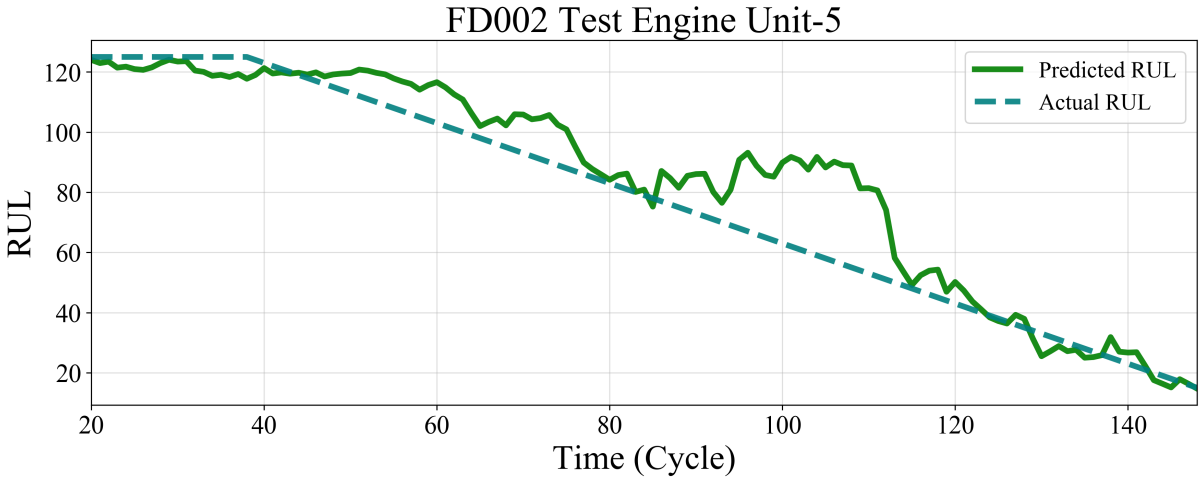


Figure 2.16: Prognostic performance from the GDCNN-1D architecture with Top-K sampling on a test engine from FD002

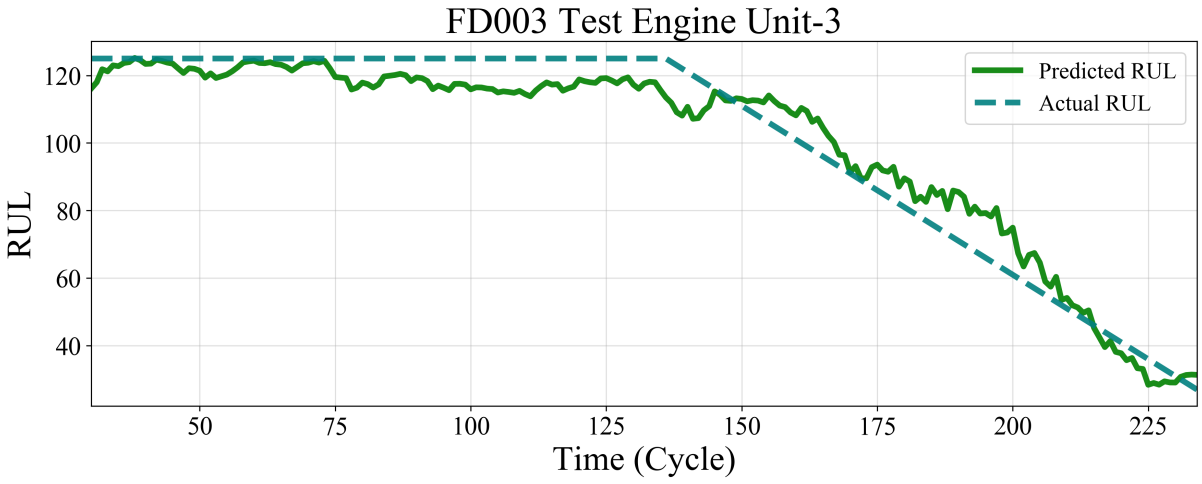


Figure 2.17: Prognostic performance from the GDCNN-1D architecture with Top-K sampling on a test engine from FD003

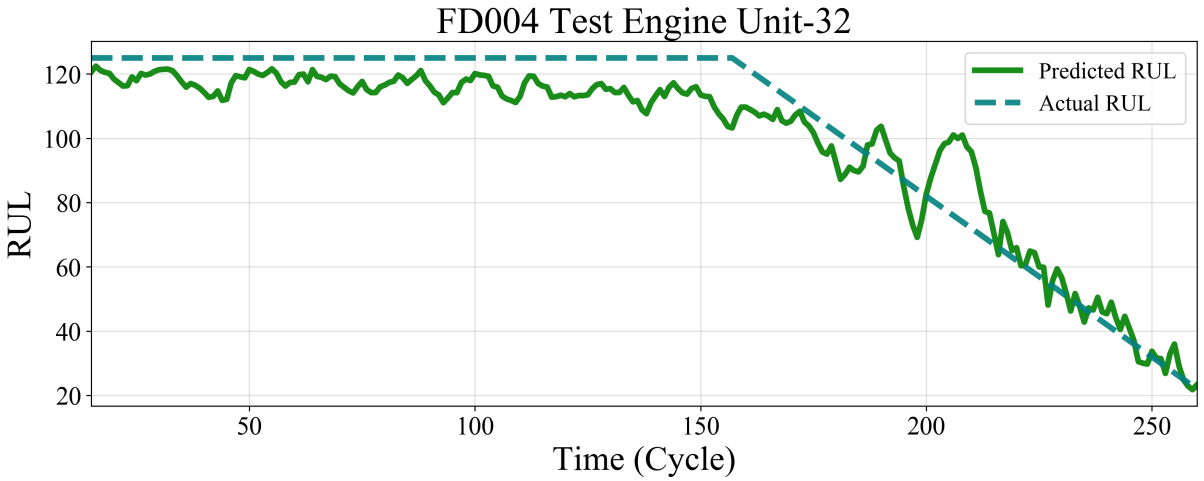


Figure 2.18: Prognostic performance from the GDCNN-1D architecture with Top-K sampling on a test engine from FD004

ferent weight matrices. In the original transformers proposed in [79], the multi-headed self-attention is achieved by transforming the multivariate input signals into as many $(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ matrices as set by the amount heads as shown in Fig. 2.19. The steps for calculating the attention are briefly explained as follows.

$$\mathbf{Q} = \mathbf{W_q} \cdot \mathbf{X} \tag{2.37}$$

$$\mathbf{K} = \mathbf{W_k} \cdot \mathbf{X} \tag{2.38}$$

$$\mathbf{V} = \mathbf{W_v} \cdot \mathbf{X} \tag{2.39}$$

where $\mathbf{W_q}, \mathbf{W_k}, \mathbf{W_v} \in \mathbb{R}^{d_i \times d}$, d_i is the no. of input features and d is the corresponding latent dimension in the layer. Self-attention matrix output is then computed with the help of these three matrices as follows

$$\mathbf{Z^{attn}} = \text{softmax}\left(\frac{\mathbf{Q} \cdot \mathbf{K^T}}{\sqrt{d_k}}\right) \cdot \mathbf{V} \tag{2.40}$$

where d_k is the key dimension in a specific layer. The softmax function t takes a vector of arbitrary real-valued scores and transforms them into a probability distribution over multiple classes. When dealing with matrices, the softmax function is applied row-wise to normalize the scores across each row. Precisely

$$\text{softmax}(Z_{ij}) = \frac{e^{Z_{ij}}}{\sum_{k=1}^{n} e^{Z_{ik}}} \tag{2.41}$$

where Z_{ij} is the i^{th} row and j^{th} column element of the input matrix $\mathbf{Z}$. In the classical transformer architecture, the attention from all h self-attention blocks are concatenated and passed through a weight matrix $\mathbf{W_a} \in \mathbb{R}^{d_k \times d}$ to produce the final attention. As shown in fig. 2.19, the individual self-attention heads collaborate to attend to information

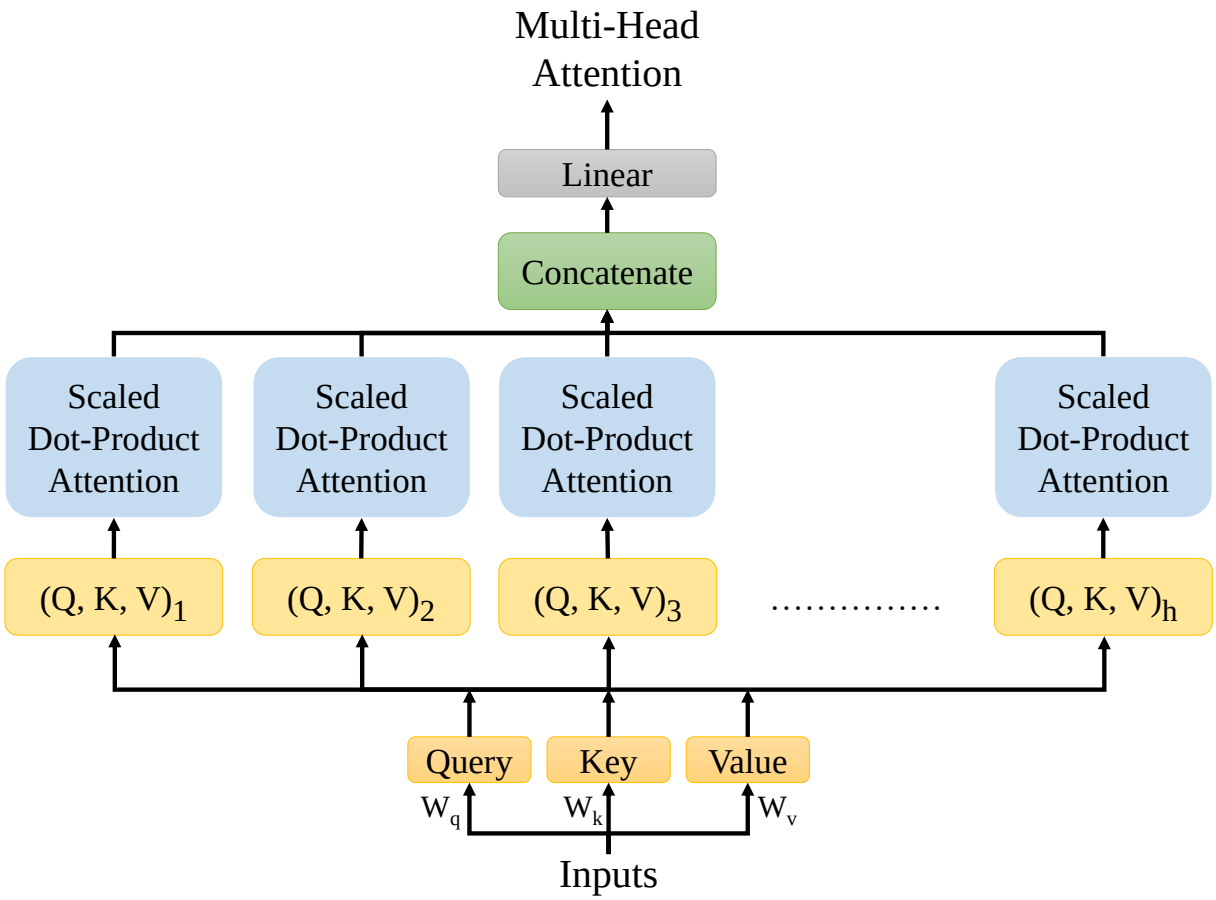


Figure 2.19: Multi-Head Attention

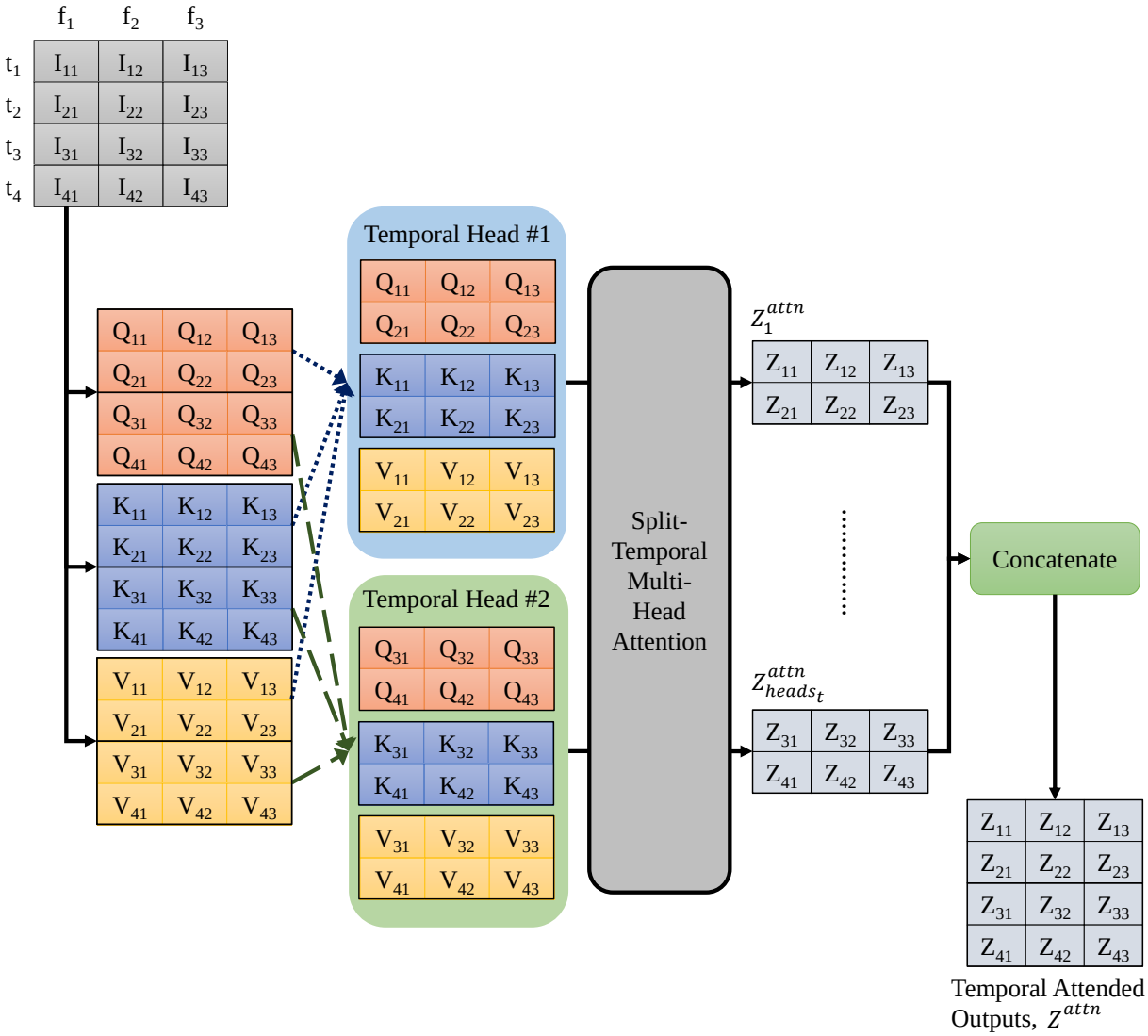


Figure 2.20: Split-Temporal Multi-Head Attention for Input with *no. of timesteps, T= 4, no. of features, i= 3 and no. of temporal heads, $heads_t$= 2.*

from multiple representation subspaces, form a context, and map it to a combined representation subspace. However, the author proposes in this thesis a Multi-head attention mechanism which is used on both the feature and temporal dimensions independently. Specifically, for feature attention, instead of passing the Query, Key and Value matrices through multiple linear layers, the dimensions are reduced by splitting them into multiple feature heads and then calculating the attention. In the case of temporal attention, the Query, Key and Value matrices are split along the temporal dimension to produce matrices with a shorter time length but with full feature size.

Split-Temporal Multi-Head Attention

Temporal heads are formed by dividing an input signal into a divisible number of shorter signals. The full process of Split-Temporal Multi-Head Attention (STMHA) is depicted in Fig. 2.20. A sample multivariate time-series signal $\mathbf{X} \in \mathbb{R}^{T \times i}$, where T is the length of the

time-series signal, and i is the size of input feature dimension. The query, key and value $(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ matrices are calculated by using the weight matrices $\mathbf{W_q}, \mathbf{W_k}, \mathbf{W_v} \in \mathbb{R}^{i \times d_e}$ where d_e is the encoder hidden dimension size as described in Equations (2.37) (2.38) (2.39). The query, key and value matrices are then separated along the time dimension to generate temporal heads. In Fig. 2.20 $T = 4$, $i = 3$, d_e= 3 and $heads_{\mathrm{t}} = 2$ have been assumed for simplicity. Self-attention is then performed on each of the temporal heads to produce individual attended matrices as described in Eq. (2.40) resulting in attention matrices as $\{\mathbf{Z_1^{attn}}, \mathbf{Z_2^{attn}}, ..., \mathbf{Z_{heads_t}^{attn}}\} \in \mathbb{R}^{heads_t \times d_e}$. Subsequently, the concatenated split temporal multi-headed attention output is $\mathbf{Z_{stmha}^{attn}} \in \mathbb{R}^{T \times d_e}$.

Shared Temporal Attention Block for Transformer Encoder

This section introduces a novel concept of raw feature segmentation and univariate STMHA, intending to decrease noise or interference from nearby sensor data while attending to timesteps in a single input feature. A multivariate input signal is divided into univariate signals, and each of those signals passes through a Shared Temporal Attention (STA) block, as illustrated in Figure 2.21. The STMHA mechanism functions as explained in the previous section, with the input feature dimension in the STA block set to $i = 1$ for segmented univariate signals. A position-wise Feed-Forward Network (FFN) layer, a residual addition, and a layer normalization are applied after the univariate temporal attended output has been processed. Specifically, the FFN operation for a univariate temporal attended matrix Z_i^{attn} can be defined as

$$FFN(\mathbf{Z_i^{attn}}) = max(0, (\mathbf{Z_i^{attn}} \cdot \mathbf{W_1} + \mathbf{b_1})) \cdot \mathbf{W_2} + \mathbf{b_2} \tag{2.42}$$

where $\mathbf{W_1} \in \mathbb{R}^{d_e \times (m.d_e)}$, $\mathbf{W_2} \in \mathbb{R}^{(m.d_e) \times d_e}$, $\mathbf{b_1} \in \mathbb{R}^{(m.d_e)}$ and $\mathbf{b_2} \in \mathbb{R}^{d_e}$ are the weights and biases of the two feed-forward layers. $m \in Z^+$ is a multiplier that increases the inner dimensionality of the FFN layer. The corresponding equations for layer normalization [83] are

$$\mu^l = \frac{1}{d_e} \sum_{i=1}^{d_e} \mathbf{Z_i^{attn}} \tag{2.43}$$

$$\sigma^l = \sqrt{\frac{1}{d_e} \sum_{i=1}^{d_e} (\mathbf{Z_i^{attn}} - \mu^l)^2} \tag{2.44}$$

where μ^l and standard deviation σ^l are calculated across all the hidden nodes in the l^{th} layer.

$$\hat{\mathbf{Z}}_\mathbf{i}^{\mathbf{attn}} = \gamma \left(\frac{\mathbf{Z_i^{attn}} - \mu_l}{\sigma_l + \epsilon} \right) + \beta \equiv \mathrm{LayerNorm}_{\gamma,\beta}(\mathbf{Z_i^{attn}}) \tag{2.45}$$

where γ and β are learnable parameters. ϵ is for numerical stability in case the denominator becomes zero by chance. The FFN output and the original layer output are residually added and normalized as

$$\mathbf{Z_i^{norm}} = \mathrm{LayerNorm}_{\gamma,\beta}(\mathbf{Z_i^{attn}} + FFN(\mathbf{Z_i^{attn}})). \tag{2.46}$$

Finally, the shared temporal-attended univariate signal is calculated as

$$\mathbf{Z_i} = \mathbf{W_{flat}} \cdot \mathbf{Z_i^{norm}} + \mathbf{b_{flat}} \tag{2.47}$$

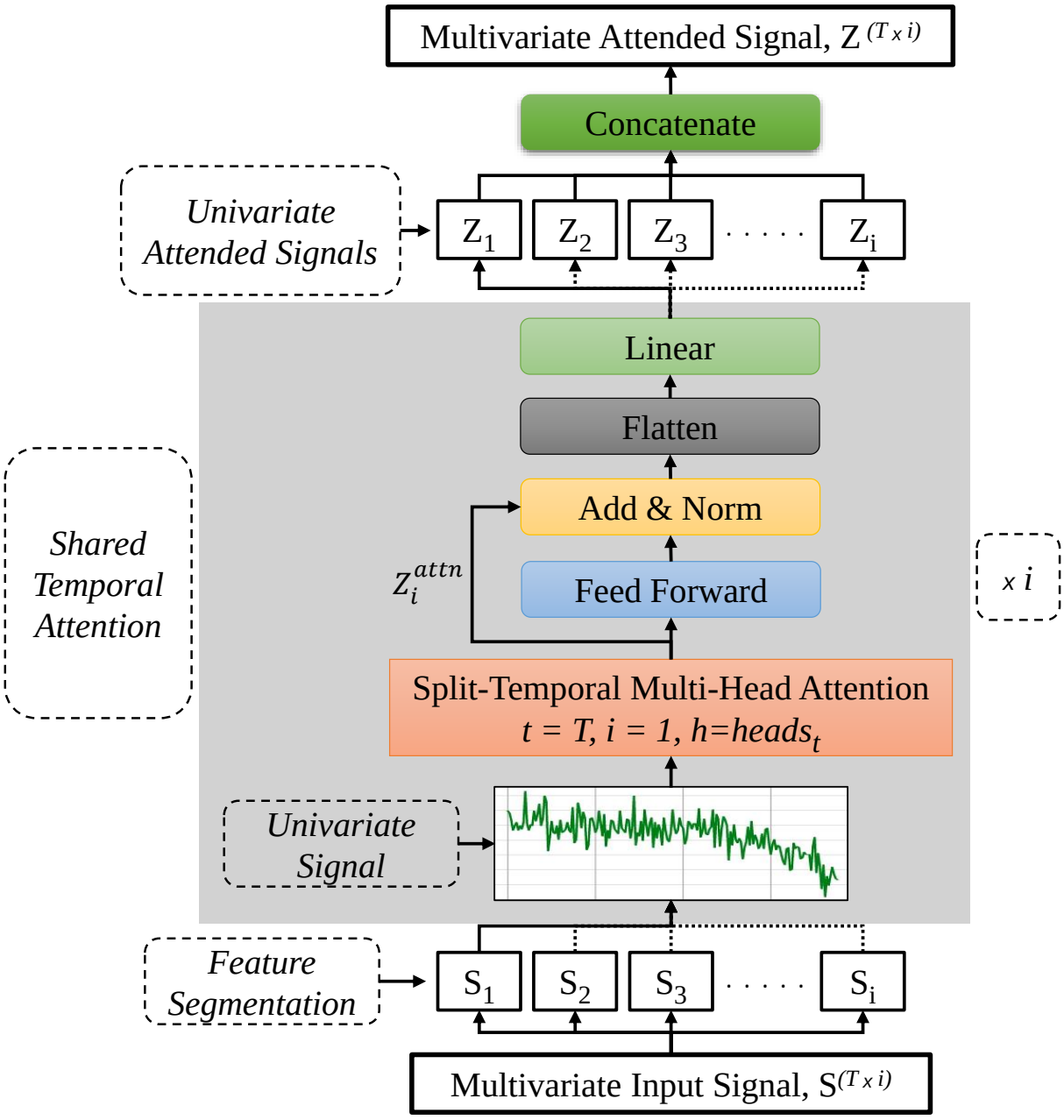


Figure 2.21: Shared Temporal Attention Block for Transformer Encoder

where $\mathbf{W_{flat}} \in \mathbb{R}^{(T.d_e)\times T}$, $\mathbf{b_{flat}} \in \mathbb{R}^T$ are the weight and biases in the flattening layer to produce $\mathbf{Z_i} \in \mathbb{R}^T$. The STMHA block runs for a i number of segmented sensor signals for a multivariate raw input signal to create shared univariate attended signals as $\{\mathbf{Z_1}, \mathbf{Z_2}, ..., \mathbf{Z_i}\} \in \mathbb{R}^T$. A multivariate shared-temporal attended output is created by concatenating these signals along the feature dimension as

$$\mathbf{Z} = (\mathbf{Z_1}, \mathbf{Z_2}, \mathbf{Z_3}, ..., \mathbf{Z_i}) \equiv \text{concat}(\mathbf{Z_i}). \tag{2.48}$$

The word "shared" denotes that the weights of each layer in the STA block are distributed evenly across the segmented raw signals. As a result, the model learns the links between each sensor while also paying attention to the pattern of sensor deterioration over time. The encoder of both the suggested transformer architectures has this STA block.

Split-Feature Multi-Head Attention

Similar to Split Temporal Multi-Head Attention, a novel idea of separating input features to create feature heads is presented and shown in Fig. 2.22, namely the Split-Feature Multi-Head Attention (SFMHA). The SFMHA substitutes the traditional multi-head attention technique in the decoder part of the proposed transformer architectures. Additionally, SFMHA is included in the encoder part of one of the proposed models. The process of splitting the features, as shown in Fig. 2.22, is as follows. The $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ matrices are formed by passing the raw input $\mathbf{X} \in \mathbb{R}^{T\times i}$ through weight matrices $\mathbf{W_q}, \mathbf{W_k}, \mathbf{W_v}$ $\in \mathbb{R}^{i\times d_d}$ where d_d is the decoder hidden dimension size. The $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ matrices are then separated into $heads_f$ number of feature heads along the feature dimension, with the newly split vectors having a dimension of $i_{heads} = \frac{d_d}{heads_f}$. For ease of display, the values in Fig. 2.22 have been set to $T = 3, i = 3, d_d = 6$, and $heads_f = 2$. In order to create the associated attended matrices $\{\mathbf{Z_1^{attn}}, \mathbf{Z_2^{attn}}, .., \mathbf{Z_{heads_f}^{attn}}\} \in \mathbb{R}^{T\times i}$, self-attention is applied to the split feature heads. Concatenating the distinct feature head attention outputs along the feature dimension results in the creation of a feature-attended output vector $\mathbf{Z_{sfmha}^{attn}}$ $\in \mathbb{R}^{T\times d_d}$.

Shared Temporal Attention Transformer *(STAT)*

This thesis introduces the Shared Temporal Attention Transformer (STAT), a new transformer design. The encoder and decoder components of the architecture are built of a stack (represented as $N\times$) of encoder and decoder layers as illustrated in Fig. 2.23. There is no weight sharing across the various encoder and decoder layers. Before the first encoder and first decoder layers, positional encoding (PE) [79] is connected to provide information about the relative or absolute position of each input data point.

Encoder: A multivariate input signal's features are first divided up into separate univariate signals at the encoder layer such that the input matrix $\mathbf{X} \in \mathbb{R}^{T\times i}$ is converted into individual vectors as $\{\mathbf{i_1}, \mathbf{i_2}, .., \mathbf{i_i}\} \in \mathbb{R}^{T\times 1}$. To create a multivariate attended signal $\mathbf{Z_{stmha}^{norm}} \in \mathbb{R}^{T\times i}$, these segmented univariate signals are transmitted through the Shared Temporal Attention (STA) block. By sharing weights across all the segmented features in the initial attention phase, the encoder can learn high-level feature representations across the sensors in the input data while at the same time learning the intrinsic deterioration pattern within each sensor, independent of other sensors. The shared attention method

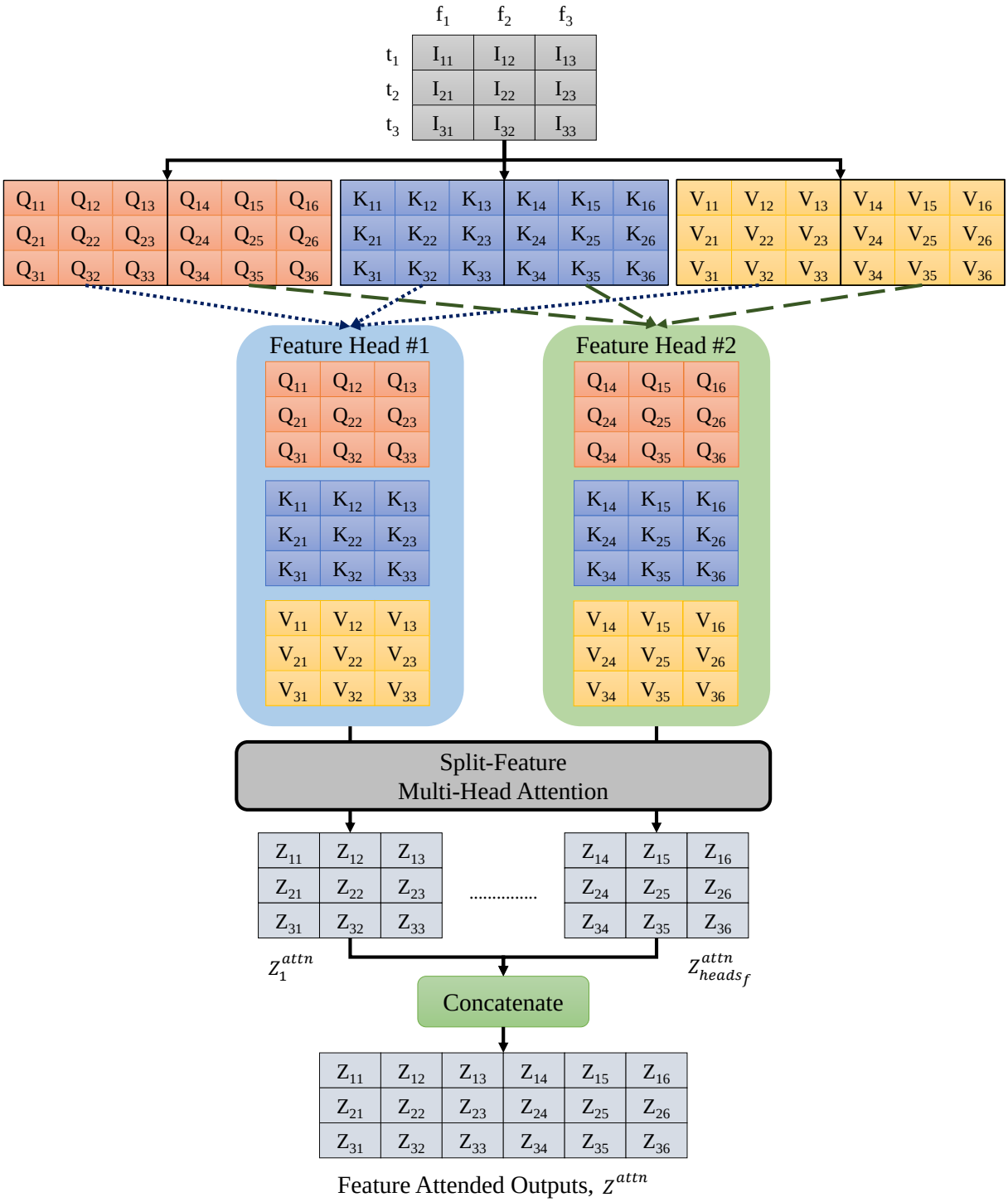


Figure 2.22: Split-Feature Multi-Head Attention for Input with *no. of timesteps*, $T = 3$, *no. of features*, $i = 6$ *and no. of features heads*, $heads_f = 2$.

improves the coherence of the multivariate attended signal over the complete feature set. The multivariate attended signal after that provided the second phase of attention via STMHA. The primary distinction between the first and second attention phases is that the first attention phase performs STMHA on each of the i segmented feature sensors in a shared recurrent way. In contrast, the second attention phase does STMHA on the multivariate signal as a whole. The STMHA enables coherent feature interaction while executing self-attention along the temporal axis. Following this layer are a residual connection, an FFN layer, and a layer normalization. The equations involved in these layers for any multivariate STMHA output are as follows:

$$FFN(\mathbf{Z^{attn}_{stmha}}) = max(0, (\mathbf{Z^{attn}_{stmha}} \cdot \mathbf{W_3} + \mathbf{b_3})) \cdot \mathbf{W_4} + \mathbf{b_4} \tag{2.49}$$

$$\mathbf{Z^{norm}_{stmha}} = \text{LayerNorm}_{\gamma,\beta}(\mathbf{Z^{attn}_{stmha}} + FFN(\mathbf{Z^{attn}_{stmha}})) \tag{4.6}$$

where $\mathbf{W_3} \in \mathbb{R}^{i\times(m\cdot i)}$, $\mathbf{W_4} \in \mathbb{R}^{(m\cdot i)\times i}$, $\mathbf{b_3} \in \mathbb{R}^{m\cdot i}$ and $\mathbf{b_4} \in \mathbb{R}^{i}$ are the weights and biases of the feedforward layers. $m \in \mathbb{Z}^+$ is the dimensionality multiplier of the FFN layer. The subsequent encoder layer performs the attention mentioned earlier while receiving the normalized output from its previous encoder layer. The N^{th} encoder layer is reached after N iterations of this operation. The key and value, which are the inputs for the decoder layers, are created by multiplying the output of the N^{th} encoder layer by weight matrices $\mathbf{W^k_{ED}}$, $\mathbf{W^v_{ED}} \in \mathbb{R}^{i\times d_d}$.

Decoder: A tensor of zeros acts as the start-of-sequence to start the first decoder layer in the stack of decoders. The SFMHA is the initial stage inside a decoder layer. The feature-attended output from this layer proceeds through layer normalization after passing via a residual connection. The query matrix, Q, for the second attention stage of the decoder layer, is the normalized output of the first attention stage. The output of the Nth encoder layer is the source of the key and value matrices. These three matrices initiate the second attention phase, which is an SFMHA form of attention. A residual connection, layer normalization, and an FFN layer are then applied. Two Fully Connected (FC) layers make up the FFN layer in the decoder, and the first FC layer is followed by non-linear Leaky ReLU activation [59]. For any normalized input $\mathbf{Y^{norm1}_D} \in \mathbb{R}^{T_o\times d_e}$ to the FFN layer, followed by another add and normalization layer the equations can be summarised as

$$FFN(\mathbf{Y^{norm1}_D}) = max(0, (\mathbf{Y^{norm1}_D} \cdot \mathbf{W_5} + \mathbf{b_5})) \cdot \mathbf{W_6} + \mathbf{b_6} \tag{2.50}$$

$$\mathbf{Y^{norm2}_D} = \text{LayerNorm}_{\gamma,\beta}(\mathbf{Y^{norm1}_D} + \mathbf{FFN}(\mathbf{Y^{norm1}_D})) \tag{2.51}$$

where $\mathbf{W_5} \in \mathbb{R}^{d_d\times(m\cdot d_d)}$, $\mathbf{W_6} \in \mathbb{R}^{(m\cdot d_d)\times d_d}$ are the weights and biases of the two linear layers in the decoder FFN. $m \in \mathbb{Z}^+$ is the dimensionality multiplier of the FFN layer and d_d is the decoder hidden dimension. A linear layer with weight matrix $\mathbf{W_{RUL}} \in \mathbb{R}^{d_e\times d_{RUL}}$ that predicts a sequence output with a dimension of $d_{RUL} = 1$ is the last step in a decoder layer. This predicted output is sent on to the following decoder layer, and the cycle is repeated until the N^{th} decoder layer generates the sequence RUL labels. Like the encoder layers, the N decoder layers do not share weights. To execute the second attention phase with the query matrix vector produced from the masked SFMHA attention phase, each decoder layer gets the identical key and value matrices from the N^{th} encoder layer.

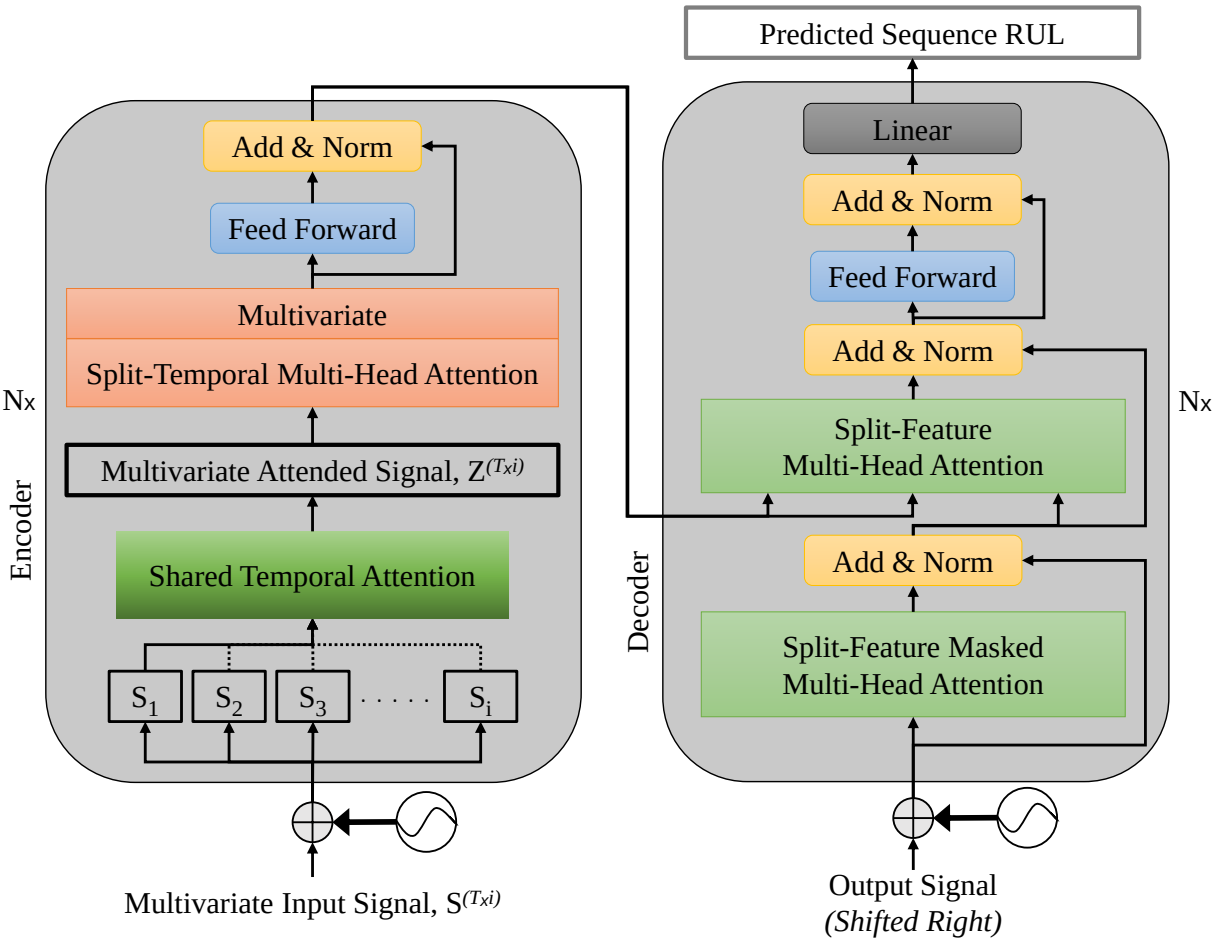


Figure 2.23: Shared Temporal Attention Transformer *(STAT)*.

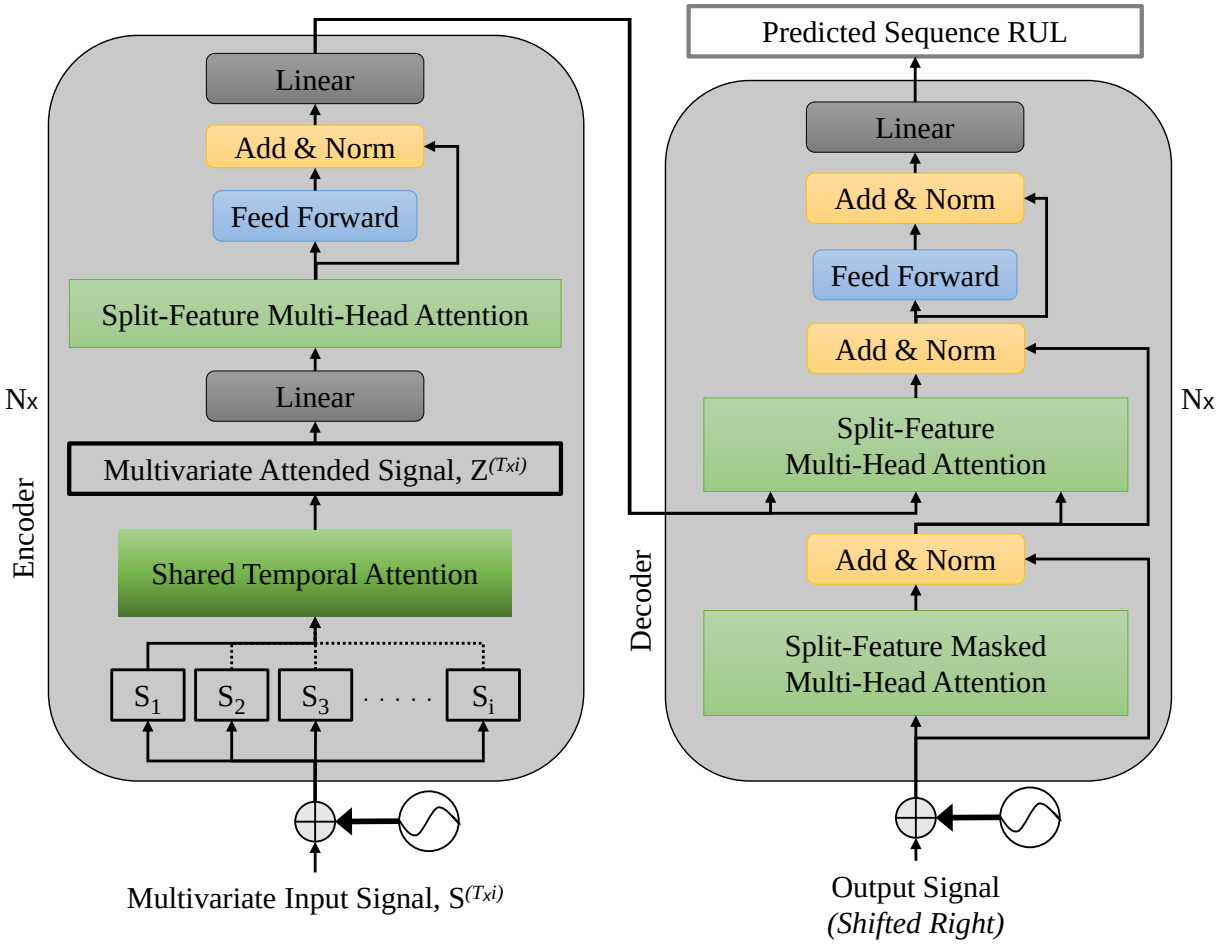


Figure 2.24: Feature-Represented Shared Temporal Attention Transformer *(FeaR-STAT)*.

Feature-Represented Shared Temporal Attention Transformer *(FeaR-STAT)*

This section presents a novel Feature-Represented Shared Temporal Attention Transformer (FeaR-STAT). Except for the encoder layer, the design is similar to the proposed STAT architecture as shown in Fig. 2.24. In contrast to the STAT architecture, an SFMHA form of attention is used for the second attention phase in the encoder, which eliminates the encoder's absolute dependence on temporal attention. The individually attended features can execute self-attention throughout the feature dimension for the whole length of the input signal by applying SFMHA to the STA multivariate output. This improves the encoder's capacity to pay attention to the intricate deterioration pattern affecting all the features across the whole input signal time period.

Encoder: Since the working principle of the FeaR-STAT decoder is similar to that of the STAT decoder, only the encoder part is explained in detail here. The segmentation of sensor features from a multivariate input signal is the first step in the FeaR-STAT encoder $\mathbf{X} \in \mathbb{R}^{T \times i}$. The segmented univariate input signals $\{\mathbf{x_1}, \mathbf{x_2}, .., \mathbf{x_i}\} \in \mathbb{R}^{T}$ are passed through the STA layer to create a multivariate attended signal $\mathbf{Z} \in \mathbb{R}^{T \times i}$. Subsequently, the multivariate attended signal is passed through a linear layer with a weight matrix, $\mathbf{W_{expand}} \in \mathbb{R}^{T \times d_d}$ to create a new attended output $\mathbf{Z} \in \mathbb{R}^{T \times d_d}$. The expansion in feature size is required since in SFMHA, the input signal $\mathbf{Z}$ is split across the feature domain

to create $heads_f$ feature heads, each of length T and number of features, $i_{heads} = \frac{d_d}{heads_f}$ where i_{heads}, d_d and $heads_f \in \mathbb{Z}^+$. The feature-attended output passes to a linear layer, a residual connection and layer normalization. The equations to these operations can be summarised as

$$FFN(\mathbf{Z^{attn}_{sfmha}}) = max(0, (\mathbf{Z^{attn}_{sfmha}} \cdot \mathbf{W_7} + \mathbf{b_7})) \cdot \mathbf{W_8} + \mathbf{b_8} \tag{2.52}$$

$$\mathbf{Z^{norm}_{sfmha}} = \text{LayerNorm}_{\gamma,\beta}(\mathbf{Z^{attn}_{sfmha}} + FFN(\mathbf{Z^{attn}_{sfmha}})) \tag{2.53}$$

where $\mathbf{W_7} \in \mathbb{R}^{d_d \times (m \cdot d_d)}$, $\mathbf{W_8} \in \mathbb{R}^{(m \cdot d_d) \times d_d}$, $\mathbf{b_7} \in \mathbb{R}^{m \cdot d_d}$ and $\mathbf{b_8} \in \mathbb{R}^{d_d}$ are the weights and biases of the feedforward layers. $m \in \mathbb{Z}^+$ is the dimensionality multiplier of the encoder FFN layer. The final step in a FeaR-STAT encoder layer is passing the $\mathbf{Z^{norm}_{sfmha}}$ output through a linear layer having weights $\mathbf{W_{reduce}} \in \mathbb{R}^{d_d \times i}$ to create an encoder output $\mathbf{Z^{norm}_{sfmha}} \in \mathbb{R}^{T \times i}$. The FeaR-STAT encoder operation is recurrent, i.e., the output from one layer is passed on to the next for N times until the N^{th} encoder layer generates a latent representation for the decoder layers.

Experimental Results with Temporal Attention Transformer Networks

The experimental findings of the presented transformer architectures are presented in this section. The Turbofan Engine Degradation Dataset, also known as the C-MAPSS dataset presented in 2.2.2, is utilized in this study to assess the model performances. The performance evaluation metric as defined in [80]

$$s_{\text{i}} = \begin{cases} e^{\frac{d_i}{a_1}} - 1, & d_i < 0 \\ e^{\frac{d_i}{a_2}} - 1, & d_i \geq 0 \end{cases} \tag{2.54}$$

$$S = \sum_{i=1}^{N} s_i \tag{2.55}$$

where, $a_1 = 13$, $a_2 = 10$, $d_i = \text{Predicted RUL} - \text{Target RUL}$ for the i^{th} engine, $s_i =$ score for the i^{th} engine, N are the total engines in a test sub-dataset and S is the final performance score for that sub-dataset.

Fig. 2.25 and Fig. 2.26 represent sample RUL estimation plots by the proposed STAT and FeaR-STAT models, respectively. An analysis of STAT and FeaR-STAT performance on the C-MAPSS dataset using various hyperparameter settings, including the number of temporal and feature heads, encoder and decoder hidden sizes, and the number of encoder and decoder layers, is presented here.

Fig. 2.27 and Fig. 2.28 illustrate the effects of varying the number of Temporal Heads, $heads_t$, in the STAT and FeaR-STAT models, respectively. A $heads_t$ count of 4 results in the overall best performance across all the sub-datasets in both scenarios, holding all other hyperparameters constant. Further increasing the temporal headcount reduces performance across the board for the STAT and FeaR-STAT designs.

However, it is evident by analysing the impact of the number of Feature Heads,$heads_f$ in the STAT model from Fig. 2.29 that the overall prediction performance improves when $heads_f$ is increased up to 16. The overall results are adversely affected by a rise in $heads_f$,

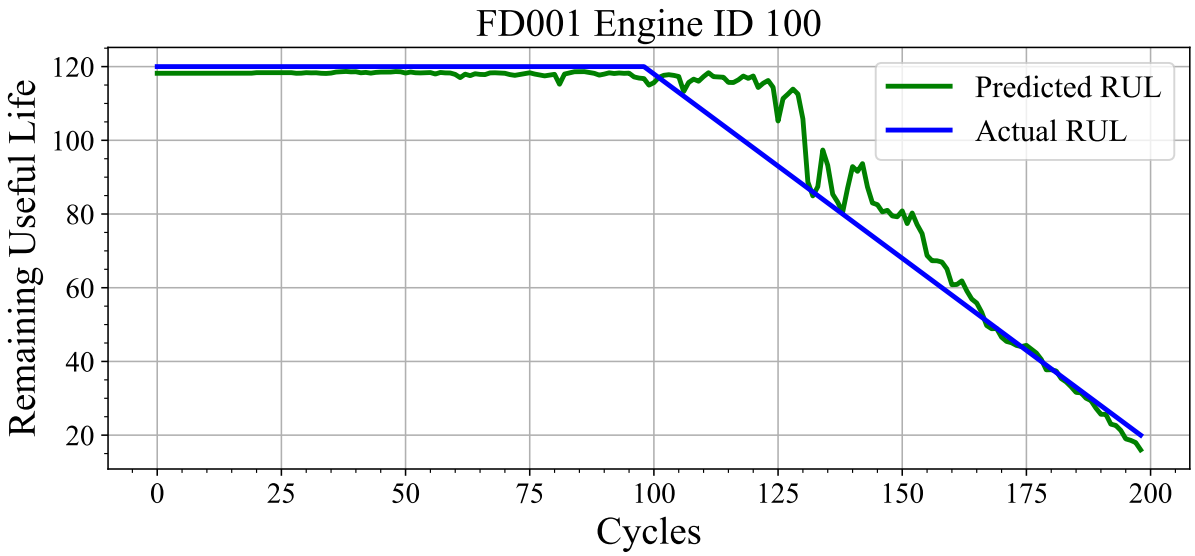


Figure 2.25: Remaining Useful Lifetime (RUL) estimation Plots of FD001 Engine ID 100 by proposed STAT architecture.

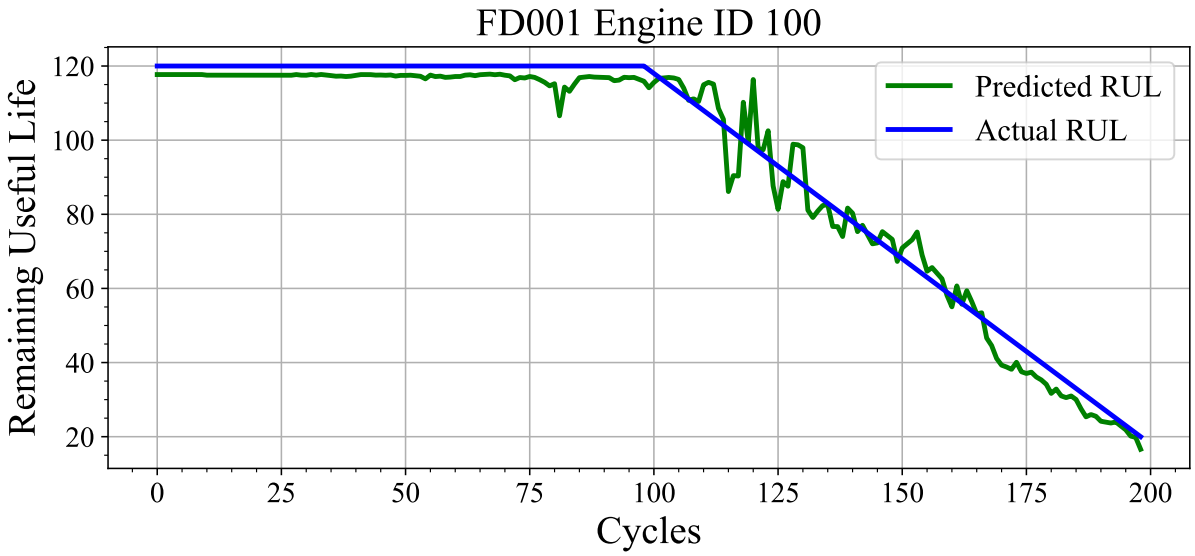


Figure 2.26: Remaining Useful Lifetime (RUL) estimation plots of FD001 Engine ID 100 by proposed FeaR-STAT architecture.

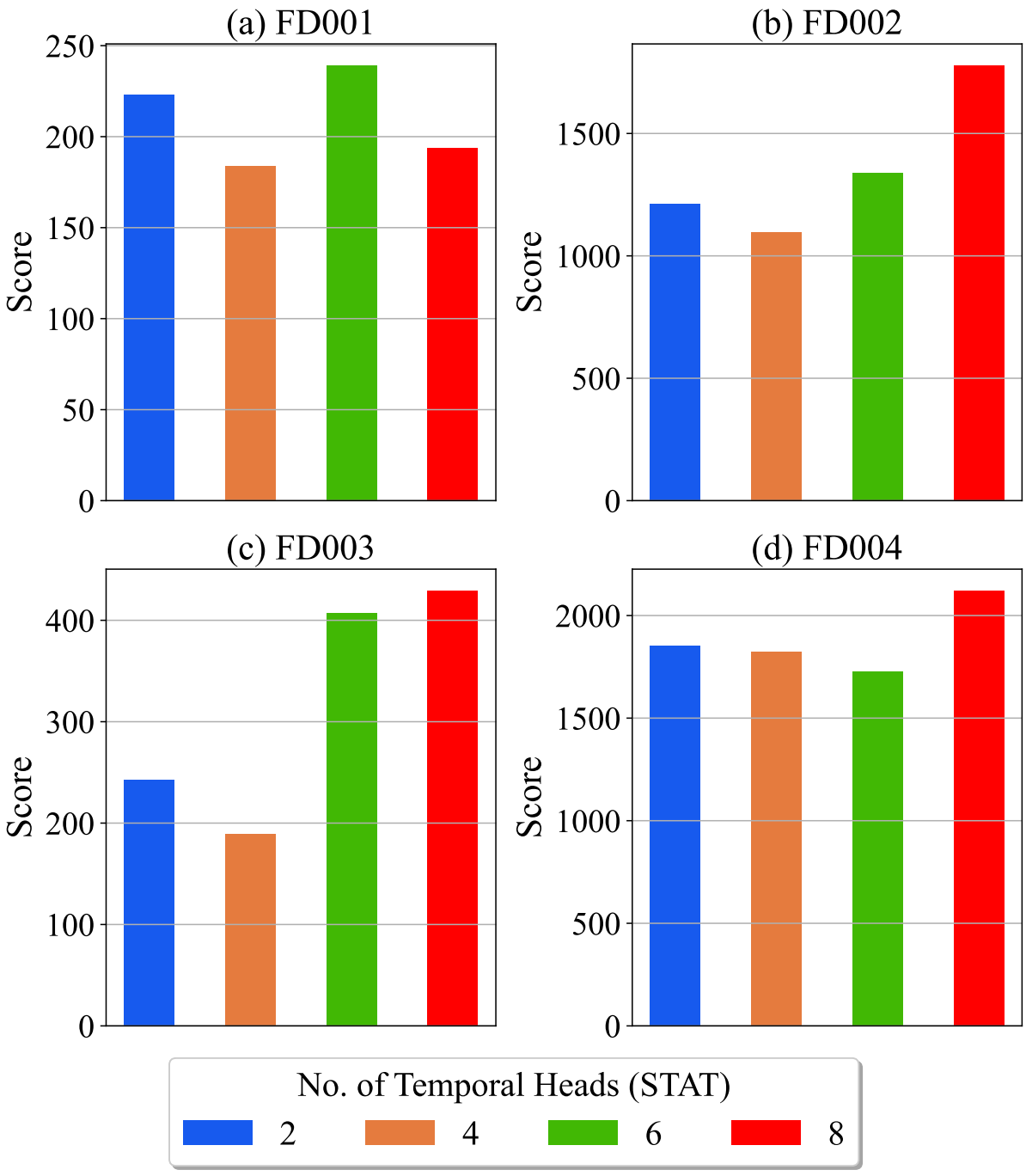


Figure 2.27: Effect of Temporal Heads *(heads$_t$)* in proposed STAT Architecture.

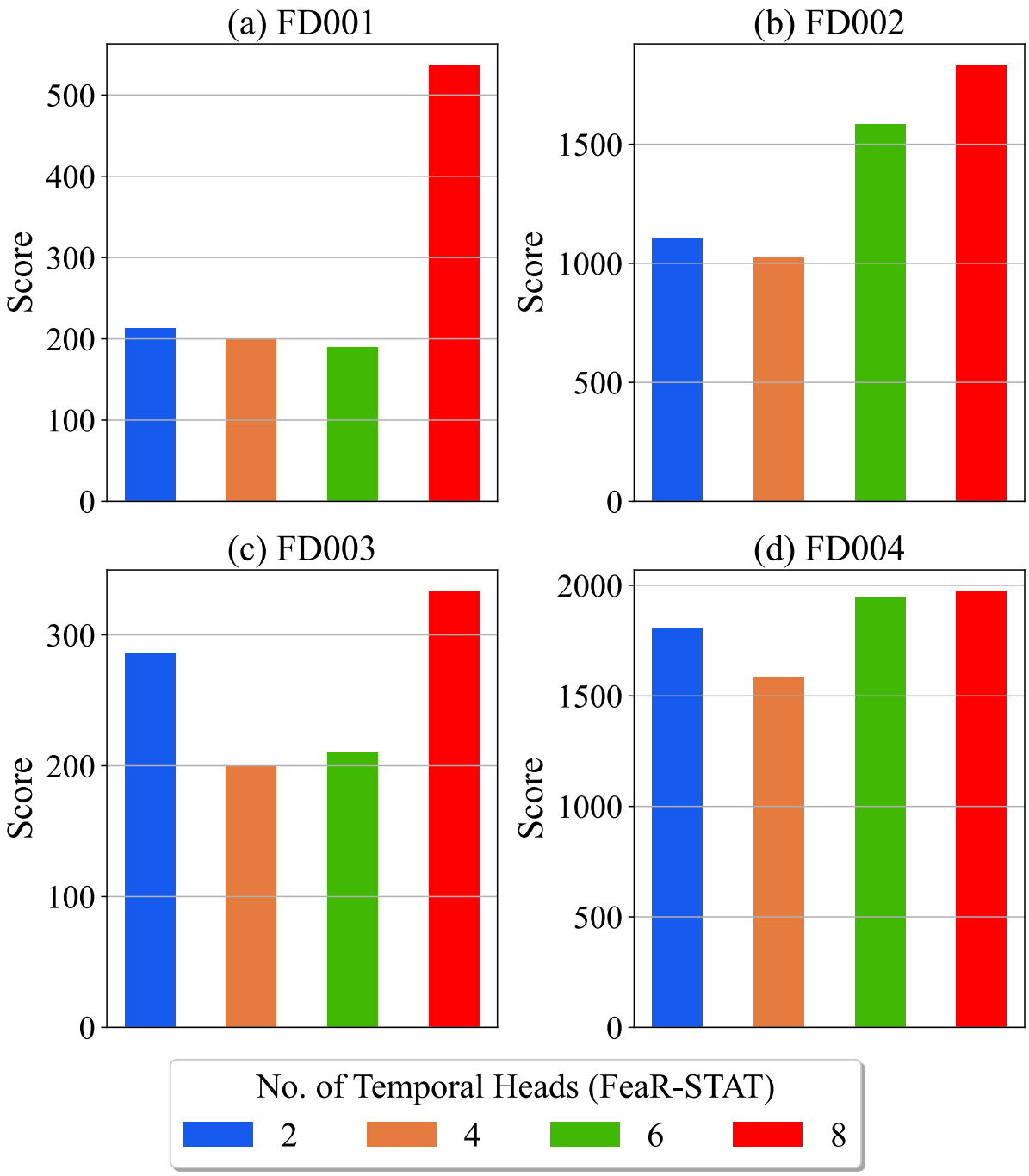


Figure 2.28: Effect of Temporal Heads *(heads$_t$)* in proposed FeaR-STAT Architecture.

Table 2.3: Evaluated Model Performance on *(RMSE)*

Model Description	**RMSE**			
	FD001	**FD002**	**FD003**	**FD004**
STAT	12.1	15.2	10.6	15.54
FeaR-STAT	12.01	15.5	10.9	15.03

notably in the FD001 and FD003 sub-datasets. The influence of the number of Feature Heads, $heads_f$ parameter in the FeaR-STAT model is demonstrated in Fig. 2.30. The model's performance does not improve with an increase in the number of Feature Heads. Instead, maintaining them at 8 yields the best overall performance across all sub-datasets.

As per Fig. 2.31 and Fig. 2.33, increasing the encoder and decoder hidden sizes (d_e, d_d) decreases the STAT model's performance in all four sub-datasets. With the help of STAT's deep architecture, complex patterns may be learned from input data, and matching RULs are produced. Therefore, it's possible that greater hidden sizes won't be essential to enhance model performance. Similar behaviour can be seen in the Fear-STAT model from Fig. 2.32 and Fig. 2.34 where performance degrades as the encoder and decoder hidden size increases.

As illustrated in Fig. 2.35, the STAT model delivers the highest average performance by stacking the fewest encoder and decoder layers. Due to the stacked layers' lack of weight sharing, increasing the number of layers proportionately increases the number of trainable parameters and eventually causes the model performance to decline. One of the essential characteristics of the STAT design is the use of relatively less trainable parameters. In the case of the FeaR-STAT model, as shown in Fig. 2.36, the performance decline is much more pronounced due to adding additional stacked encoder and decoder layers.

The comparison of the proposed models based on their Root Mean Square Error (RMSE) losses during the testing phase is shown in Table 2.3. The RMSE loss shows the model's ability to provide precise RUL forecasts throughout an engine's life. This illustrates how accurately a model can depict the pattern of degradation from the beginning to the conclusion of the life of any equipment. The proposed transformer RMSEs surpass current literature in all operating circumstances, demonstrating the effectiveness of the techniques.

2.2.4 Contribution

The contribution of this section which is based on papers III and IV, are novel learnable dilated CNN and transformer neural networks. Both of these approaches are specifically designed for multivariate time series data analysis. The standard dilated CNNs are generalized in the section 2.2.2, which enables the universal representation of any dilation patterns. Two end-to-end training strategies for the Generalized dilated CNNs architecture leveraging the barrier function and Top-K sampling techniques are presented. The shared temporal and split-feature attention blocks presented in section 2.2.3 help discover deterioration patterns for RUL estimation. The proposed architectures and their respective variants are tested on the C-MAPSS benchmark dataset outperforming current state-of-the-art methods.

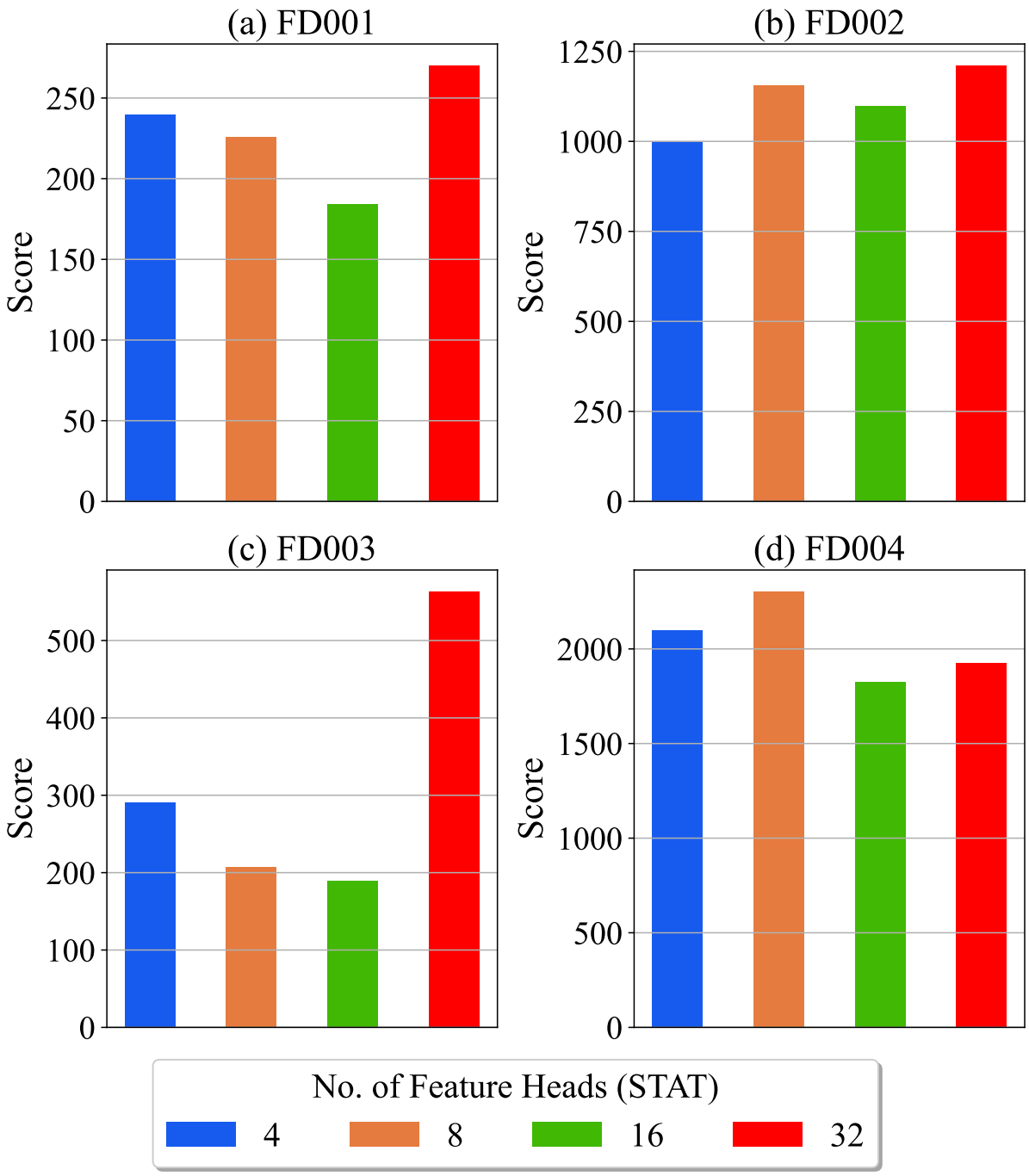


Figure 2.29: Effect of Feature Heads *(heads$_f$)* in proposed STAT Architecture.

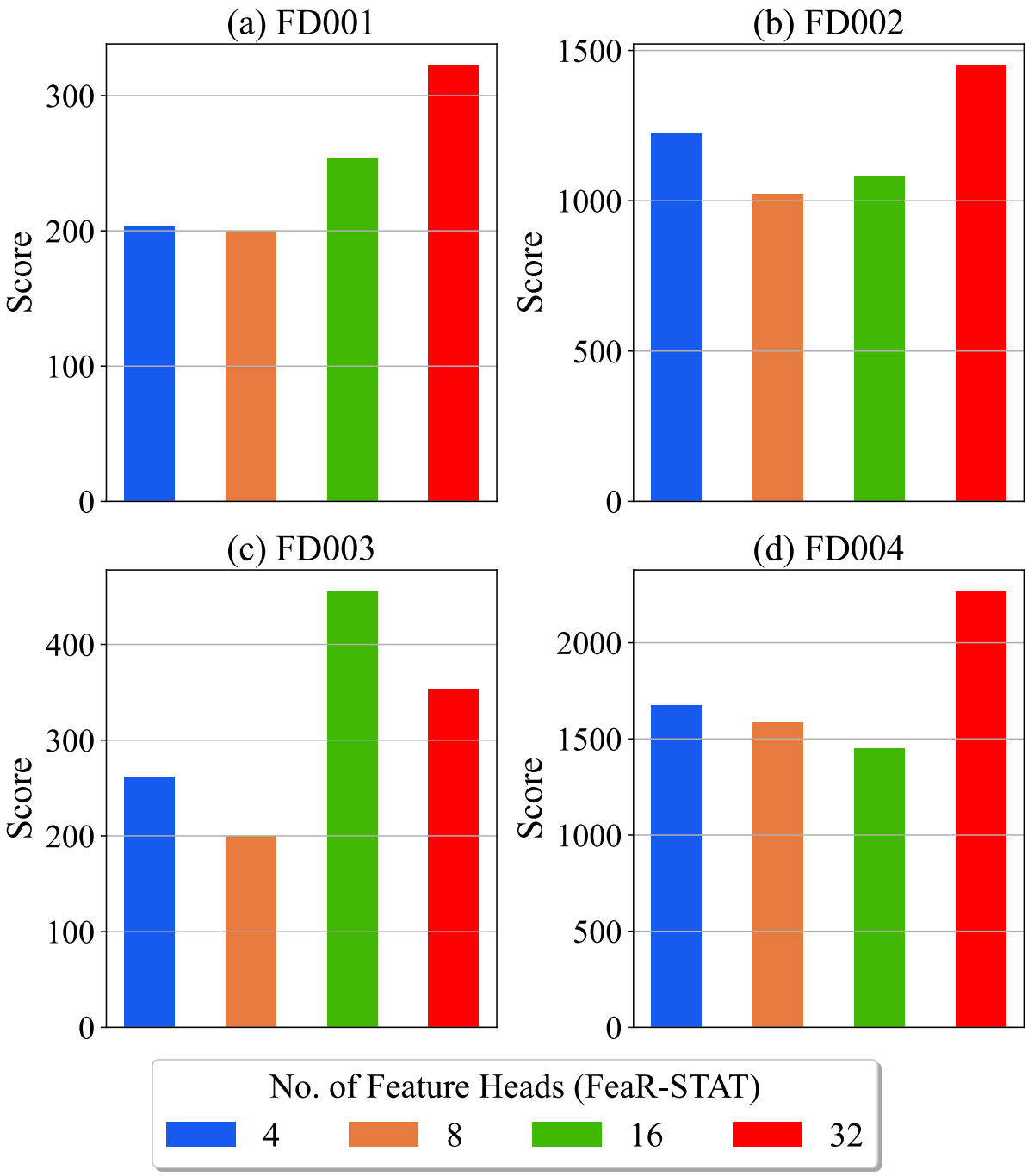


Figure 2.30: Effect of Feature Heads *(heads$_f$)* in proposed FeaR-STAT Architecture.

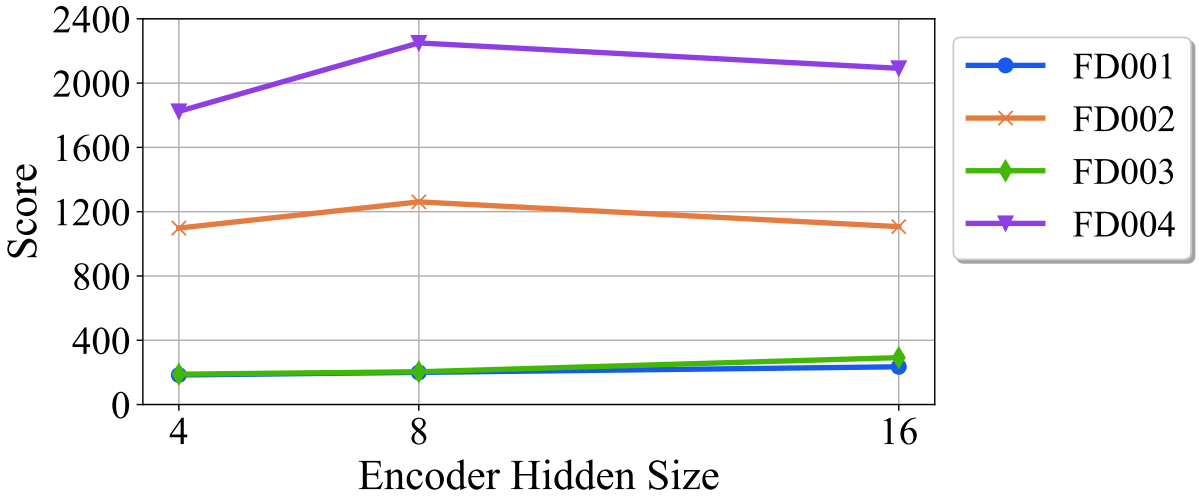


Figure 2.31: Effect of Encoder Hidden Size (d_e) in proposed STAT Architecture.

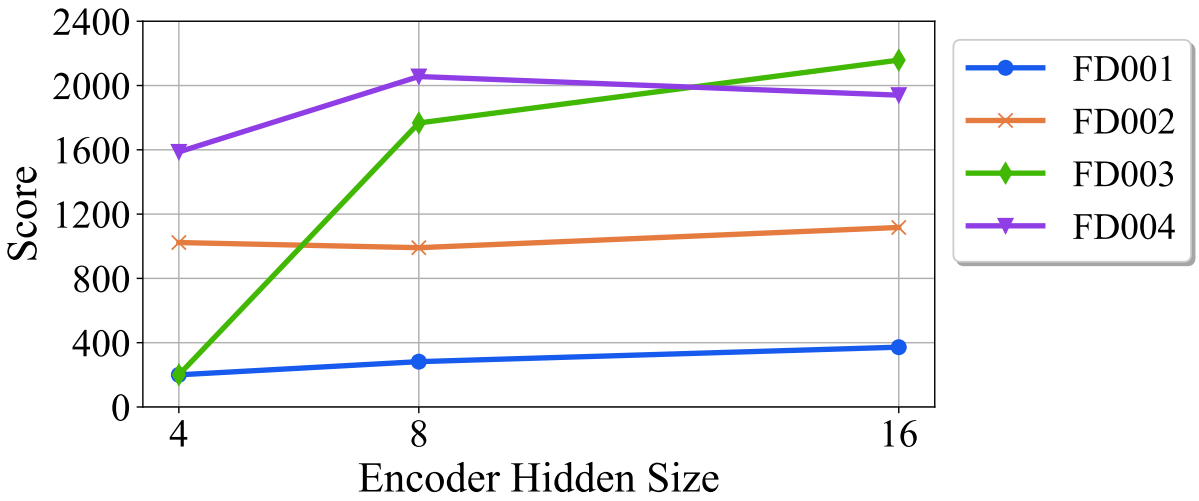


Figure 2.32: Effect of Encoder Hidden Size (d_e) in proposed FeaR-STAT Architecture.

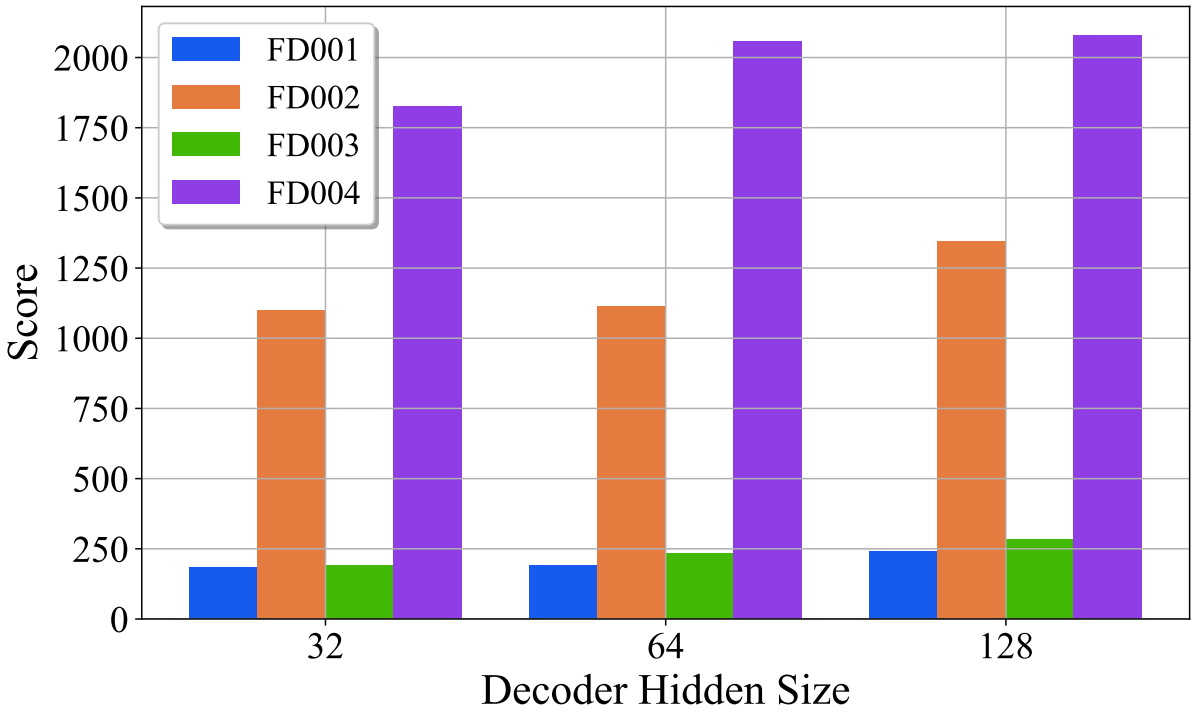


Figure 2.33: Effect of Decoder Hidden Size (d_d) in proposed STAT Architecture.

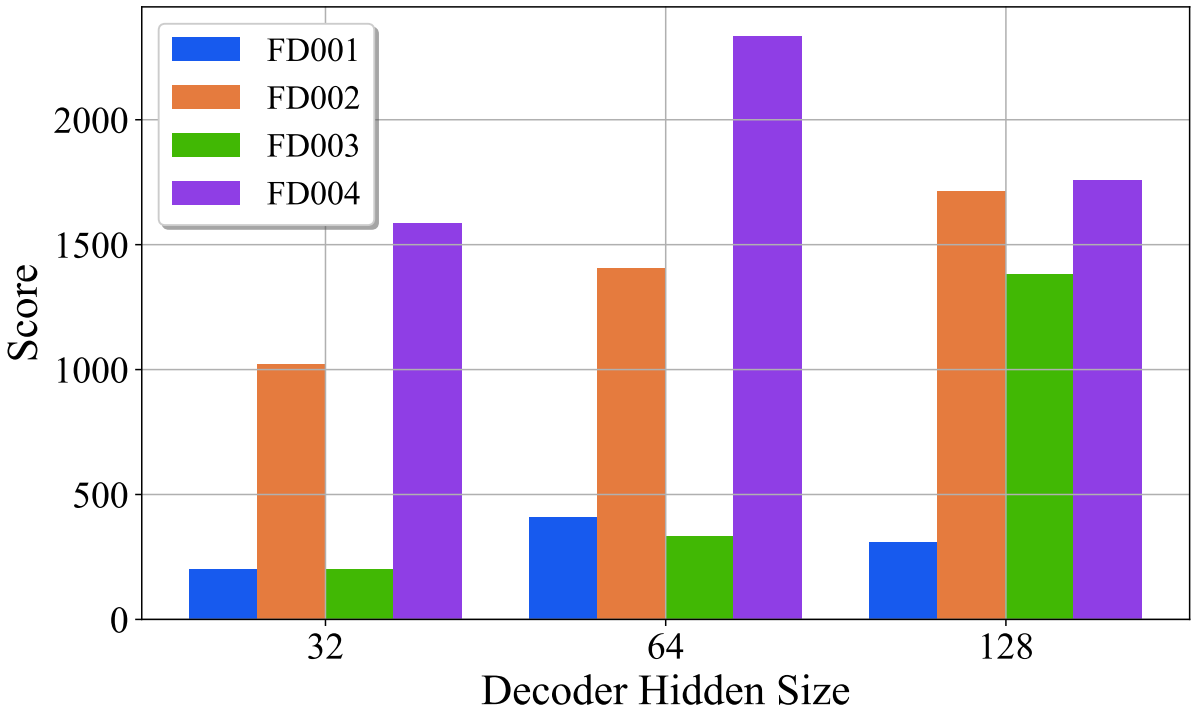


Figure 2.34: Effect of Decoder Hidden Size (d_d) in proposed FeaR-STAT Architecture.

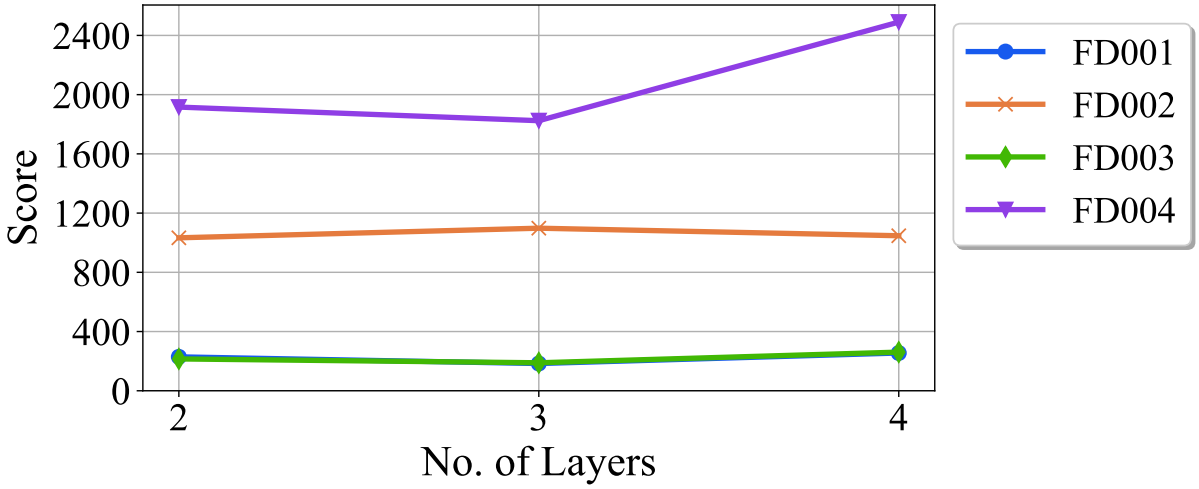


Figure 2.35: Effect of no. of Layers (L_{ED}) in proposed STAT Architecture.

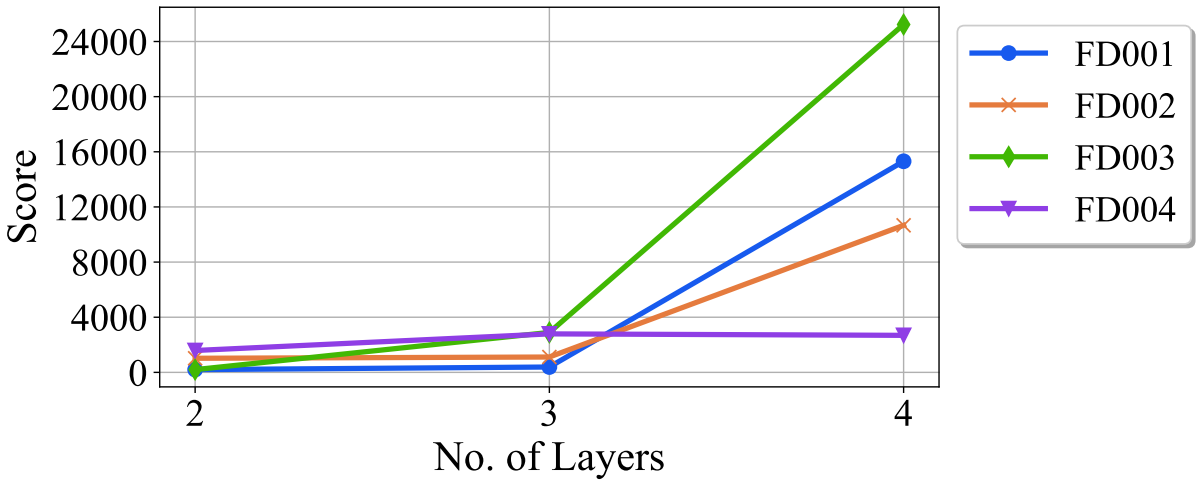


Figure 2.36: Effect of no. of Layers (L_{ED}) in proposed FeaR-STAT Architecture.

3. Conclusion and Future Work

The work in this dissertation is focused on the development and assessment of suitable DNN architectures and learning algorithms for prognostics and health management of industrial systems. In particular, several enhancements to the existing DNN architectures for multivariate time-series analysis have been proposed to improve prediction performance on multiple complex industrial benchmark settings. This chapter provides an outline of potential future efforts in addition to summarizing the entire research project, techniques, and conclusions.

3.1 Conclusion

Prognostics and health management of industrial systems is an essential part of product life-cycle management, and DNN architectures have emerged to play a critical role in this field. Standard DNN architectures such as Convolutional Neural Networks, Recurrent Neural Networks and Transformers have been used for multivariate time-series analysis for industrial systems. I investigated three significant directions for Prognostics and health management of industrial systems with DNN architectures: fault classification, anomaly detection and remaining useful lifetime estimation. The novel DNN architectures and learning algorithms are presented and evaluated for all three directions. The findings are as follows.

First, it is shown that using a bidirectional LSTM architecture implicitly improves the prediction performance for incipient fault classification, as the proposed architecture provides a viewpoint change for time-series analysis. The proposed sliding window data operation makes storing long-term data in LSTM cells easier, making it possible to take into account even longer time series patterns. Consequently, increasing the sequence length increased the generalisation capability in the incipient fault case detection. The proposed data preprocessing and efficient training techniques allow all the proposed RNN models to handle raw multivariate time-series signals. It is illustrated that the proposed techniques yield better results than previous findings on the benchmark TE fault detection dataset. The initial multi-class classification approach also provides promising results with the bidirectional LSTM architecture.

Second, a new framework for anomaly detection with unsupervised and semi-supervised training based on CNN auto-encoders is proposed, which utilises an auxiliary loss function to enhance the hidden representation of the CNN models. The auxiliary loss function is calculated only for a subset of the hidden variables and is based on the K-means clustering loss. This extra constraint on the subsection of the CNN auto-encoders latent variables forces the model to learn a representation which is amenable for the anomaly detection task. The proposed training algorithm includes the splitting of the latent variables based on a Top-k approach wherein the latent variables with the maximum euclidean

distance between the cluster centres are identified as the highest contributing variables for anomaly detection. Furthermore, the suggested approach may be used in any neural network model due to its modular design. The proposed approach also addresses the lack of labelled data for anomaly detection in a real-world setting. The experimental results on the benchmark TE dataset show that the CNN models with the auxiliary loss outperform the vanilla CNN models on all the anomaly categories. Furthermore, analysing the type of input fed to the semi-supervised classifier also underlines that the clustering loss is helpful in the overall prediction performance of the CNN model.

Third, a new approach for CNN is presented wherein a generalized dilation layer is designed for adaptive sampling in case of multivariate time series data analysis. The generalized dilation extends the rigid standard dilation approach to be flexible across the input receptive field resulting in the selection of the most relevant time-steps and feature variables for the RUL prediction task. Unlike standard dilation in CNN, the generalized dilation layer is trainable for each convolution kernel. Since training the generalized dilation is a discrete optimisation problem, two new training methodologies were developed to make it compatible with the gradient-based learning of CNN. The first approach uses the barrier function, which calculates varying penalties for failing to comply with the constraints of discrete optimisation. The second solution applies the hard constraints using a "Top-K" sampling strategy wherein the constraints are fulfilled by utilising the maximum K values in the convolution weight matrix. Both of these training approaches were tested on benchmark RUL data sets to understand the impact of each of the components individually. Both the training procedures lead to outstanding performance on RUL prediction tasks, with the Top-K training approach performing slightly better. Overall, both the proposed methodologies perform better than the other Deep Learning RUL estimation approaches, underlining the universal applicability of the methods.

Fourth, two new architectures, the STAT and FeaR-STAT models, are proposed based on the transformer neural network architecture. The architectures are designed to better focus on the parameter-sharing phenomena to model the inter-dependency in the temporal and feature domains. Specifically, the Shared Temporal Attention block and the Split-Temporal Multi-Head Attention focus on modelling the raw input signal such that the information at any given time can attend to its past and future information as well. Furthermore, the encoder's suggested shared temporal attention layer achieves the objective of focusing on the temporal patterns of signal degradation in each unique sensor signal before establishing a shared correlation across the feature range. The STAT and FeaR-STAT architectures employ the proposed shared temporal attention mechanism in the encoder and the Split-Feature Multi-Head Attention in the decoder. The architectural modifications result in two high-performance models with fewer trainable parameters than standard transformer models to avoid scalability issues in practical applications. The C-MAPSS Turbofan Engine dataset is used to evaluate the proposed architectures. Both the transformer architectures proposed in this thesis reach state-of-the-art performance in predicting highly accurate RUL estimation from raw input sequences.

3.2 Future Work

This section provides suggestions for future work and enhancements to the proposed models and training procedures. The future research work can be summarised as follows. First, the explainability and interpretability of deep neural network models is a promising area for understanding their predictions. This is becoming more important in recent times since deep neural network models are being used more often in real production environments. Additionally, since health management models usually work in tandem with human supervisors on the shop floor, it is necessary that humans also understand the learned representations of these models. Some initial research works proposing such methods in the visual modelling tasks are [84, 85].

Second, pre-training deep neural network models with unlabelled data and then adapting the model to a specific task with labelled data has revolutionised their generalisation capabilities. These pre-training approaches have been used to learn high-level representations effectively in different application domains like fault detection [86]. These existing approaches can be extended to RUL estimation using multivariate time-series data. Furthermore, generative or energy-based pre-training approaches can be investigated for system health management.

Third, the generalized dilation technique proposed in this thesis can be extended to the entire input space and not just to the receptive field. In this new architecture, we first apply a dilated operation on the input image and a standard convolution operation to the dilated input. The resulting architecture essentially separates its operation into a pattern extraction step followed by a standard convolution operation on a reduced input data set.

Finally, since all the approaches proposed in this thesis are highly modular, they can be individually combined to create robust and reliable architectures for system health management. In conjunction with the Top-K clustering algorithm, the generalized dilation models can create a stable anomaly detection system. Similarly, the STAT and FeaR-STAT can also be used as feature extractors for the Top-K clustering algorithm. The concept of dilation can also be extended to the STAT and FeaR-STAT architectures, where only certain parts of the input are used for representing the interdependency in the input dataset.

Bibliography

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[3] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. van der Laak, B. van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical Image Analysis*, vol. 42, pp. 60–88, 2017.

[4] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *International Journal of Computer Vision*, vol. 128, no. 2, pp. 261–318, 2020.

[5] F. Lateef and Y. Ruichek, "Survey on semantic segmentation using deep learning techniques," *Neurocomputing*, vol. 338, pp. 321–348, 2019.

[6] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.

[7] L. Zhang, S. Wang, and B. Liu, "Deep learning for sentiment analysis: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 4, p. e1253, 2018.

[8] M. R. Costa-jussà, "From feature to paradigm: Deep learning in machine translation," *Journal of Artificial Intelligence Research*, vol. 61, pp. 947–974, 2018.

[9] Z. Abbasiantaeb and S. Momtazi, "Text–based question answering from information retrieval and deep neural network perspectives: A survey," *WIREs Data Mining and Knowledge Discovery*, vol. 11, no. 6, p. 429, 2021.

[10] M. Yousefi-Azar and L. Hamey, "Text summarization using unsupervised deep learning," *Expert Systems with Applications*, vol. 68, pp. 93–105, 2017.

[11] T. N. Sainath, R. J. Weiss, K. W. Wilson, B. Li, A. Narayanan, E. Variani, M. Bacchiani, I. Shafran, A. Senior, K. Chin, A. Misra, and C. Kim, "Multichannel signal processing with deep neural networks for automatic speech recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 5, pp. 965–979, 2017.

[12] M. Kim, "Collaborative deep learning for speech enhancement: A run-time model selection method using autoencoders," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 76–80, 2017.

[13] Haytham M. Fayek, Margaret Lech, and Lawrence Cavedon, "Evaluating deep learning architectures for speech emotion recognition," *Neural Networks*, vol. 92, pp. 60–68, 2017.

[14] S. Khan and T. Yairi, "A review on the application of deep learning in system health management," *Mechanical Systems and Signal Processing*, vol. 107, pp. 241–265, 2018.

[15] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989.

[16] L. Da Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Transactions on industrial informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[17] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. S. Iyengar, "A survey on deep learning: Algorithms, techniques, and applications," *ACM Computing Surveys*, vol. 51, no. 5, 2018.

[18] Y. Maki and K. A. Loparo, "A neural-network approach to fault detection and diagnosis in industrial processes," *IEEE Transactions on Control Systems Technology*, vol. 5, no. 6, pp. 529–541, 1997.

[19] V. Venkatasubramanian and K. Chan, "A neural network methodology for process fault diagnosis," *AIChE Journal*, vol. 35, no. 12, pp. 1993–2002, 1989.

[20] R. Vaidyanathan and V. Venkatasubramanian, "Representing and diagnosing dynamic process data using neural networks," *Engineering Applications of Artificial Intelligence*, vol. 5, no. 1, pp. 11–21, 1992.

[21] S. X. Ding, *Model-Based Fault Diagnosis Techniques*. London: Springer London, 2013.

[22] Z. Gao, C. Cecati, and S. X. Ding, "A survey of fault diagnosis and fault-tolerant techniques—part i: Fault diagnosis with model-based and signal-based approaches," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 6, pp. 3757–3767, 2015.

[23] Z. Gao, C. Cecati, and S. X. Ding, "A survey of fault diagnosis and fault-tolerant techniques—part ii: Fault diagnosis with knowledge-based and hybrid/active approaches," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 6, pp. 3768–3774, 2015.

[24] S. X. Ding, *Data-driven Design of Fault Diagnosis and Fault-tolerant Control Systems*. London: Springer London, 2014.

[25] S. Yin, S. X. Ding, A. Haghani, H. Hao, and P. Zhang, "A comparison study of basic data-driven fault diagnosis and process monitoring methods on the benchmark tennessee eastman process," *Journal of Process Control*, vol. 22, no. 9, pp. 1567–1581, 2012.

[26] F. Jia, Y. Lei, J. Lin, X. Zhou, and N. Lu, "Deep neural networks: A promising tool for fault characteristic mining and intelligent diagnosis of rotating machinery with massive data," *Mechanical Systems and Signal Processing*, vol. 72-73, pp. 303–315, 2016.

[27] P. Jiang, Z. Hu, J. Liu, S. Yu, and F. Wu, "Fault diagnosis based on chemical sensor data with an active deep neural network," *Sensors (Basel, Switzerland)*, vol. 16, no. 10, 2016.

[28] W. Sun, S. Shao, R. Zhao, R. Yan, X. Zhang, and X. Chen, "A sparse auto-encoder-based deep neural network approach for induction motor faults classification," *Measurement*, vol. 89, pp. 171–178, 2016.

[29] M. Xia, T. Li, T. Shu, J. Wan, C. W. de Silva, and Z. Wang, "A two-stage approach for the remaining useful life prediction of bearings using deep neural networks," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3703–3711, 2019.

[30] Y. Wang, H. Yang, X. Yuan, Y. A. Shardt, C. Yang, and W. Gui, "Deep learning for fault-relevant feature extraction and fault classification with stacked supervised auto-encoder," *Journal of Process Control*, vol. 92, pp. 79–89, 2020.

[31] J. Tao, Y. Liu, and D. Yang, "Bearing fault diagnosis based on deep belief network and multisensor information fusion," *Shock and Vibration*, vol. 2016, no. 7, pp. 1–9, 2016.

[32] Z. Zhang and J. Zhao, "A deep belief network based fault diagnosis model for complex chemical processes," *Computers & chemical engineering*, vol. 107, pp. 395–407, 2017.

[33] J. Deutsch and D. He, "Using deep learning-based approach to predict remaining useful life of rotating components," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 1, pp. 11–20, 2018.

[34] C. Zhang, P. Lim, A. K. Qin, and K. C. Tan, "Multiobjective deep belief networks ensemble for remaining useful life estimation in prognostics," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2306–2318, 2017.

[35] H. Liu, J. Zhou, Y. Zheng, W. Jiang, and Y. Zhang, "Fault diagnosis of rolling bearings with recurrent neural network-based autoencoders," *ISA transactions*, vol. 77, pp. 167–178, 2018.

[36] T. de Bruin, K. Verbert, and R. Babuška, "Railway track circuit fault diagnosis using recurrent neural networks," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 3, pp. 523–533, 2017.

[37] J. Lei, C. Liu, and D. Jiang, "Fault diagnosis of wind turbine based on long short-term memory networks," *Renewable Energy*, vol. 133, pp. 422–432, 2019.

[38] L. Guo, N. Li, F. Jia, Y. Lei, and J. Lin, "A recurrent neural network based health indicator for remaining useful life prediction of bearings," *Neurocomputing*, vol. 240, pp. 98–109, 2017.

[39] Y. Cheng, H. Zhu, J. Wu, and X. Shao, "Machine health monitoring using adaptive kernel spectral clustering and deep long short-term memory recurrent neural networks," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 987–997, 2019.

[40] S. Zhao, Y. Zhang, S. Wang, B. Zhou, and C. Cheng, "A recurrent neural network approach for remaining useful life prediction utilizing a novel trend features construction method," *Measurement*, vol. 146, pp. 279–288, 2019.

[41] X. Guo, L. Chen, and C. Shen, "Hierarchical adaptive deep convolution neural network and its application to bearing fault diagnosis," *Measurement*, vol. 93, pp. 490–502, 2016.

[42] L. Wen, X. Li, L. Gao, and Y. Zhang, "A new convolutional neural network-based data-driven fault diagnosis method," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 7, pp. 5990–5998, 2018.

[43] M. Xia, T. Li, L. Xu, L. Liu, and C. W. de Silva, "Fault diagnosis for rotating machinery using multiple sensors and convolutional neural networks," *IEEE/ASME Transactions on Mechatronics*, vol. 23, no. 1, pp. 101–110, 2018.

[44] T. Pan, J. Chen, Z. Zhou, C. Wang, and S. He, "A novel deep learning network via multiscale inner product with locally connected feature extraction for intelligent fault detection," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 9, pp. 5119–5128, 2019.

[45] R. Liu, G. Meng, B. Yang, C. Sun, and X. Chen, "Dislocated time series convolutional neural architecture: An intelligent fault diagnosis approach for electric machine," *IEEE Transactions on industrial informatics*, vol. 13, pp. 1310–1320, 2017.

[46] G. Jiang, H. He, J. Yan, and P. Xie, "Multiscale convolutional neural networks for fault diagnosis of wind turbine gearbox," *IEEE Transactions on Industrial Electronics*, vol. 66, no. 4, pp. 3196–3207, 2019.

[47] K. B. Lee, S. Cheon, and C. O. Kim, "A convolutional neural network for fault classification and diagnosis in semiconductor manufacturing processes," *IEEE Transactions on Semiconductor Manufacturing*, vol. 30, no. 2, pp. 135–142, 2017.

[48] H. Wu and J. Zhao, "Deep convolutional neural network model based chemical process fault diagnosis," *Computers & chemical engineering*, vol. 115, pp. 185–197, 2018.

[49] G. Sateesh Babu, P. Zhao, and X.-L. Li, "Deep convolutional neural network based regression approach for estimation of remaining useful life," in *Database Systems for Advanced Applications* (S. B. Navathe, W. Wu, S. Shekhar, X. Du, X. S. Wang, and H. Xiong, eds.), vol. 9642 of *Lecture Notes in Computer Science*, pp. 214–228, Cham: Springer International Publishing, 2016.

[50] X. Li, Q. Ding, and J.-Q. Sun, "Remaining useful life estimation in prognostics using deep convolution neural networks," *Reliability Engineering & System Safety*, vol. 172, pp. 1–11, 2018.

[51] X. Li, W. Zhang, and Q. Ding, "Deep learning-based remaining useful life estimation of bearings using multi-scale feature extraction," *Reliability Engineering & System Safety*, vol. 182, pp. 208–218, 2019.

[52] T. S. Kim and S. Y. Sohn, "Multitask learning for health condition identification and remaining useful life prediction: deep convolutional neural network approach," *Journal of Intelligent Manufacturing*, 2020.

[53] H. Liu, Z. Liu, W. Jia, X. Lin, and S. Zhang, "A novel transformer-based neural network model for tool wear estimation," *Measurement Science and Technology*, vol. 31, no. 6, p. 065106, 2020.

[54] Y. Mo, Q. Wu, X. Li, and B. Huang, "Remaining useful life estimation via transformer encoder enhanced by a gated convolutional unit," *Journal of Intelligent Manufacturing*, vol. 32, no. 7, pp. 1997–2006, 2021.

[55] W. Zhang, D. Yang, Y. Xu, X. Huang, J. Zhang, and M. Gidlund, "Deephealth: A self-attention based method for instant intelligent predictive maintenance in industrial internet of things," *IEEE Transactions on industrial informatics*, vol. 17, no. 8, pp. 5461–5473, 2021.

[56] R. J. Patton, J. Chen, and T. M. Siew, "Fault diagnosis in nonlinear dynamic systems via neural networks," in *1994 International Conference on Control - Control '94*, vol. 2, pp. 1346–1351 vol.2, 1994.

[57] R. Isermann and M. Münchhof, *Identification of Dynamic Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[58] S. X. Ding, *Advanced Methods for Fault Diagnosis and Fault-tolerant Control*. Springer Berlin, Heidelberg, 2020.

[59] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[60] 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, pp. 1–14, 2011.

[61] J. Qiu, B. B. Seth, S. Y. Liang, and C. Zhang, "Damage mechanics approach for bearing lifetime prognostics," *Mechanical Systems and Signal Processing*, vol. 16, no. 5, pp. 817–829, 2002.

[62] C. Cempel, H.G. Natke, and M. Tabaszewski, "A passive diagnostic experiment with ergodic properties," *Mechanical Systems and Signal Processing*, vol. 11, no. 1, pp. 107–117, 1997.

[63] S.-T. Tseng, J. Tang, and I.-H. Ku, "Determination of burn-in parameters and residual life for highly reliable products," *Naval Research Logistics (NRL)*, vol. 50, no. 1, pp. 1–14, 2003.

[64] J.M. van Noortwijk, "A survey of the application of gamma processes in maintenance," *Reliability Engineering & System Safety*, vol. 94, no. 1, pp. 2–21, 2009.

[65] J. Luo, M. Namburu, K. Pattipati, L. Qiao, M. Kawamoto, and S. Chigusa, "Model-based prognostic techniques [maintenance applications]," in *Proceedings AUTOTESTCON 2003. IEEE Systems Readiness Technology Conference*, pp. 330–340, 2003.

[66] T. Berghout and M. Benbouzid, "A systematic guide for predicting remaining useful life with machine learning," *Electronics*, vol. 11, no. 7, 2022.

[67] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, 2008.

[68] D. P. Kingma, M. Welling, *et al.*, "An introduction to variational autoencoders," *Foundations and Trends\textregistered in Machine Learning*, vol. 12, no. 4, pp. 307–392, 2019.

[69] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," *arXiv preprint arXiv:1511.05644*, 2015.

[70] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[71] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1724–1734, Association for Computational Linguistics, 2014.

[72] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[73] J. J. Downs and E. F. Vogel, "A plant-wide industrial process control problem," *Computers & chemical engineering*, vol. 17, no. 3, pp. 245–255, 1993.

[74] L. H. Chiang, E. L. Russell, and R. D. Braatz, *Fault Detection and Diagnosis in Industrial Systems*. Advanced Textbooks in Control and Signal Processing, London: Springer London, 2001.

[75] H. M and S. M.N, "A review on evaluation metrics for data classification evaluations," *International Journal of Data Mining & Knowledge Management Process*, vol. 5, no. 2, pp. 01–11, 2015.

[76] G. S. Chadha, J. Kim, A. Schwung, and S. X. Ding, "Permutation learning in convolutional neural networks for time-series analysis," in *Artificial Neural Networks and Machine Learning – ICANN 2020* (I. Farkaš, P. Masulli, and S. Wermter, eds.), vol. 12396 of *Lecture Notes in Computer Science*, pp. 220–231, Cham: Springer International Publishing, 2020.

[77] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008.

[78] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network denver, colorado, usa, november 27-30, 1989]," in *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]* (David S. Touretzky, ed.), pp. 396–404, Morgan Kaufmann, 1990.

[79] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[80] A. Saxena, K. Goebel, D. Simon, and N. Eklund, "Damage propagation modeling for aircraft engine run-to-failure simulation," in *Prognostics and Health Management, 2008, PHM 2008, International Conference on*, ([Piscataway, N.J.]), pp. 1–9, IEEE, 2008.

[81] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *ICLR*, 2016.

[82] P. Nectoux, R. Gouriveau, K. Medjaher, E. Ramasso, and et. al, "Pronostia: An experimental platform for bearings accelerated life test," in *IEEE International Conference on Prognostics and Health Management, Denver, CO, USA*, 2012.

[83] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[84] P. Schramowski, W. Stammer, S. Teso, A. Brugger, F. Herbert, X. Shao, H.-G. Luigs, A.-K. Mahlein, and K. Kersting, "Making deep neural networks right for the right scientific reasons by interacting with their explanations," *Nature Machine Intelligence*, vol. 2, no. 8, pp. 476–486, 2020.

[85] D. Bau, J.-Y. Zhu, H. Strobelt, A. Lapedriza, B. Zhou, and A. Torralba, "Understanding the role of individual units in a deep neural network," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 117, no. 48, pp. 30071–30078, 2020.

[86] J. Pöppelbaum, G. S. Chadha, and A. Schwung, "Contrastive learning based self-supervised time-series analysis," *Applied Soft Computing*, vol. 117, p. 108397, 2022.

Publications

I. Bidirectional Deep Recurrent Neural Networks for Process Fault Classification

Outline

Bibliographic Information

Gavneet Singh Chadha, Ambarish Panambilly, Andreas Schwung, Steven X. Ding (2020, November): Bidirectional deep recurrent neural networks for process fault classification. In ISA Transactions 106, pp. 330–342.
doi: 10.1016/j.isatra.2020.07.011.

Author's contribution

The author contributed in the conceptualization, methodology, in data curation, software development, validation, formal analysis, results investigation, writing the original draft, writing the review and editing and making visualisations.

Copyright Notice

I.1 Abstract

In this study, a new approach for time series based condition monitoring and fault diagnosis based on bidirectional recurrent neural networks is presented. The application of bidirectional recurrent neural networks essentially provide a viewpoint change on the fault diagnosis task, which allows to handle fault relations over longer time horizons helping in avoiding critical process breakdowns and increasing the overall productivity of the system. To further enhance the capability, we propose a novel procedure of data preprocessing and restructuring which enforces the generalization and a more efficient data utilisation and consequently yields more efficient network training, especially for sequential fault classification task. The proposed Bidirectional Long Short Term Memory network outperforms standard recurrent architectures including vanilla recurrent neural networks, Long Short Term Memories and Gated Recurrent Units. We apply the proposed approach to the Tennessee Eastman benchmark process to test the effectiveness of the mentioned deep architectures and provide a detailed comparative analysis. The experimental results for binary as well as multi-class classification show the superior average fault detection capability of the bidirectional Long Short Term Memory Networks compared to the other architectures and to results from other state-of-the-art architectures found in the literature.

I.2 Introduction

Smart and interconnected manufacturing processes are the backbone of an economy, which increases the demand for insightful anayltics into these complex systems. The requirement for manufacturing environments and machines to run in a 24/7 setting poses high challenges to the production management. Due to the required high workload, unintentional standstill of machines has to be avoided reliably while intentional standstill times shall be reduced as much as possible. Therefore, industries now require a system that does not only prevent the faults, but can also predict them. This real time system monitoring can be achieved by utilizing the large amount of data provided by sources such as sensors, actuators, manufacturing executing systems and enterprise resource planning systems. This easy availability of huge amount of data, along with the recent research agendas of "Industry 4.0" have caused new techniques to arise ranging from self optimization and self learning manufacturing systems to highly modular and flexible production systems.

Condition monitoring and fault detection systems play a crucially important role in reliable and efficient manufacturing plants operation. Fundamentally, condition monitoring and fault detection approaches can be distinguished into data- and model-based methods [1] where the latter can be derived either from physical principles [2] or human expert knowledge. However, deriving suitable models of the system or collecting process knowledge from operators, e.g. in form of rules [3], can be tedious if not infeasible, owing to the inherent complexity and interconnected nature of modern production systems. As modern manufacturing systems provide huge amount of process data, data-based methods become more and more applicable and competitive to extract meaningful patterns and perform detection or classification for different fault cases.

Consequently, data based methods using machine learning techniques have been frequently developed, see [4, 5] for recent overviews. Particularly, deep learning [6] has

been shown to provide superior performance compared to conventional machine learning methods. Typical deep network architectures and learning paradigms include deep sparse filtering [7], deep belief networks [8] and deep autoencoder [9] where the latter are mainly used in unsupervised or semisupervised monitoring tasks. However, all of these deep learning approaches are not specifically designed for time series data as required by condition monitoring and hence, do not scale well for mining patterns in longer time sequences. Recurrent Neural Networks (RNN) [10] like Long Short Term Memory (LSTM) [11] or Gated Recurrent Units (GRU) [12] are thus better suited for sequential data analytics, which feed in the data sequentially and in theory, can handle sequences of arbitrary length. However, in practice, the sequence length is shown to be limited mainly due to training issues. This poses challenges for condition monitoring tasks as relations within multidimensional data sequences indicating failures can range over long time horizons. Furthermore, fault detection data sets are normally sparse and unbalanced, i.e. faulty data is rare compared to normal operation data which renders direct application of standard machine learning algorithms difficult.

In this paper we tackle the above problem by introducing a novel system architecture for condition monitoring and fault diagnosis using Bidirectional Long Short Term Memory (B-LSTM) [13]. B-LSTM operate directly on the raw sensor data and can model the temporal dependencies based on forward and backward data representation to two separate LSTM networks allowing to cover longer time horizons. The advantage of the proposed method is fault detection capabilities for incipient faults, which require a modelling information over a longer time horizon. Moreover, in contrast to existing B-LSTM training, where the input is fed in sequentially, we apply a many-to-one processing which inputs the whole sequence at once. Together with a sliding window approach and suitable restructuring of the data sets this allows for higher efficient data usage then previous approaches. Moreover, we present a novel data-preprocessing technique, which avoids the dependence on a specific sequence length or period of a time series signal.

The approach is applied to the well known Tennessee Eastman (TE) Process previously used as a benchmark for various data based fault diagnosis algorithms. We provide a thorough comparison study of B-LSTM, vanilla LSTM, GRU and vanilla RNN. The results clearly show the improved performance of the approach and particularly show that B-LSTM obtained increased performance compared with the literature, especially with regard to faults which are hard to classify with other approaches.

The contributions of the paper can be summarized as follows:

- We present a new approach for data based condition monitoring and fault diagnosis based on the recently developed B-LSTM. Specifically, B-LSTM provide a view point change which allows for fault classification on longer data sequences. The system is based on raw data and uses the output of the network directly as the classification results without the need for any subsequent statistical test as otherwise suggested in the literature.

- We present a novel methodology of data preprocessing for training of recurrent neural networks for condition monitoring and fault diagnosis. Particularly, we propose a restructuring of the training data set which enables the recurrent networks to better generalize to the fault conditions.

- We provide a thorough comparison between different recurrent neural network architecture including vanilla-LSTM, vanilla GRU and vanilla RNN underlining the superior performance of B-LSTMs.

The paper is organized as follows. Sec. I.3 introduces related work. In Section I.4 we state the considered problem, followed by the considered neural network structures in Sec. I.5. Sec. I.6 entails the newly developed data presentation technique for training the RNN. In Sec. I.7 we provide a thorough comparison of the recurrent networks and compare the performance with existing methods from the literature. Section I.8 concludes the paper.

I.3 Related Work

We focus our literature review on data-based methods and refer to [2] for model based approaches. In the field of data based condition monitoring, various approaches exist as has been surveyed in [14, 15, 16]. Fundamentally, previous works can be categorized in statistical, shallow learning and deep learning methods.

Statistical methods include partial least squares, principle component analysis and independent component analysis. These methods have been widely used [17, 18] to find trends and change in the individual process variables for distinguishing faulty and non-faulty case. However, as these methods are defined for linear systems, sophisticated preprocessing and feature generation methods have to be defined, e.g. in form of kernel functions, to operate with nonlinear relations in the data set as is usually the case in condition monitoring. Particularly for time series data this requires detailed knowledge from an expert to successfully design such systems. Other data-driven learning methods include K-Nearest Neighbour [19], Support Vector Machines [20], and single hidden layer Feed Forward Neural Network (FFNN) [21] that model fault detection as a supervised or a unsupervised learning problem wherein a trained classifier or multiple classifiers try to classify normal operation from faulty operations. However, as these methods are inherently static, time series condition monitoring can only be applied by incorporating additional time series feature generation methods which again require for detailed process knowledge. In contrast, deep learning methods like Deep Stacking Networks [1], Stacked Sparse Deep Autoencoders [22] and Deep Belief Networks [8] inherently allow for the generation of relevant features by using multiple layers of non-linear transformations to break the complex pattern recognition problem into a series of simpler mathematical patterns that are incorporated for the final classification task. However, extraction of relevant information from raw production data sets with these architectures remains a challenge. Particularly the inherent time dependency of fault development and its propagation demands for architectures and training schemes accounting for this time dependency.

Recently, architectures from the deep learning field particularly accounting for the time series character of condition monitoring, namely Convolutional Neural Network (CNN) and RNN have been developed. In [23] a CNN is used to detect bearing faults using a low dimensional vibration data set. Dislocated convolutions are presented in [24] where windows of different strides are processed via a CNN architecture on a (one dimensional) data set to infer electric motor faults. CNN based transfer learning approaches have been proposed in [25] where task independent features are transferred to CNN for different

fault cases and verified on six transfer fault diagnosis experiments. However, as CNN are basically designed for image data, a straightforward transfer to the time series domain is not possible. As standard 2D-filters cannot favorably be used due to the missing spatial relations, 1D filters ranging over the complete input dimension have to be used. This results in big filter sizes for multidimensional input sizes which reduces the effectiveness of CNN. Hence, the proposed CNN are applied to one or low dimensional time series data sets only. Furthermore, the fixed and predefined time window size restricts the sequence length considerably and avoids mining patterns over longer time horizons. In addition, the local nature of convolutions requires for depth to also cover broader relations, resulting in overly complex networks.

Alternatively, some approaches for time series based data-driven condition monitoring using RNN and its variants have been developed which allow for modeling longer time dependencies. Stacked LSTM autoencoders for process planning have been presented in [26]. LSTM together with correlation analysis were proposed for a time-series forecasting problem for Industrial Internet of Things equipment in [27]. The results showed an improved performance in comparison to standard autoregressive methods such as auto-regressive integrated moving average models. LSTM have also been proposed for identifying faults in railway track circuits in [28] wherein the time taken for a train to pass over a section of track circuit is taken as the sequence length of the event. The time series signals from a wind turbine have also been used for fault diagnosis with LSTM in [29]. In [30], LSTM have been applied to fault diagnosis in the TE process. However, results have only been presented for the training data set which render the results doubtful, especially as LSTM tend to strongly overfit. The above approaches rely on vanilla LSTM while we employ B-LSTMs which changes the viewpoint of fault detection, allows for detecting longer time series relations and is shown to result in improved performance. Furthermore, in contrast to the previous approaches with sequential data operation, we employ a many-to-one training scheme which fed in the time series data sequence at once. Together with a sliding window operation, this allows for highly efficient data usage. Moreover, we present a novel data pre-processing procedure particularly suitable for fault detection purposes which combines faulty with non-faulty data sequences during training allowing to detect more subtle differences.

I.4 Problem Statement

In this section we state the considered problem of data-driven fault identification based on dynamic time series signals. Since there exist many different approaches for data based process monitoring, some clarifications as well as the considered assumptions have to be stated.

1. It is assumed that we have a data set of signals that are recorded over time from various sources such as sensors and actuators. These sources can generate either discrete or continuous values.

2. It is assumed that the data set is labelled with the corresponding fault type as well as the no fault.

3. Except for data normalization, which is standard in machine learning, no feature extraction techniques have been used. The raw data from the sensors is used and is presented in a novel way to the learning algorithm which shows considerable improvements in the generalization capabilities of the networks and hence improving the test performance.

4. No additional statistical test have been undertaken for evaluating the model performance, rather the neural network output directly indicates if and which type of fault has occurred

Considered in this regard, the problem can be stated as follows: Given labeled data samples $(\mathbf{x}_t, f_t^i)$ from a time series data set of a process denoting the faulty and normal operation conditions, where $t = 0, \dots T$ denotes the sequence number and i is the type of fault, f^0 indicating normal operation, a recurrent neural network $g(\cdot)$ is trained to minimize the negative log likelihood loss function

$$\min_{\theta} L_\theta(\mathbf{x}, \mathbf{f}) = -\sum_{t=0}^{T} f(t) log(g_\theta(\mathbf{x}(t)), \tag{I.1}$$

where θ are the trainable parameters of the neural network. The learning process to solve the above stated problem by means of various RNN structures and suitable learning algorithms is presented in the next section.

I.5 Recurrent Neural Networks Architectures

We consider different neural network architectures for the application to condition monitoring and fault diagnosis. Standard FFNN consider each data point or observation independently and therefore cannot model temporal dependencies. FFNN would require an exponentially higher number of parameters to model temporal dependencies if the sequence length is taken as an additional input dimension. Such a high number of parameters would be almost impossible to train if the sequence length increases. Therefore, we postulate that a) an incipient fault cannot be detected by one instance of the data set, rather a certain time window of the input data set is required and b) FFNN are not suitable to learn the temporal dependency in a data set efficiently.

RNN on the other hand with their inherent structure to retain the state of a process over a longer time period, can solve these types of problems. This is precisely the reason why RNN and its variants have found massive success in the various fields of Natural Language Processing and Natural Language Understanding [31] since they are able to model the sequential information which is in the form of word embedding. Consequently, we model the problem of fault detection and identification from the data set consisting of raw sensor data instead of word embedding. As stated above, especially to detect difficult fault classes, we postulate that network architectures which are able to store the sequential information of the process over a certain window are needed. With these objectives of modelling sequential information in the backdrop, we introduce the various recurrent neural network architectures, namely vanilla RNN, vanilla LSTM, vanilla GRU and B-LSTM.

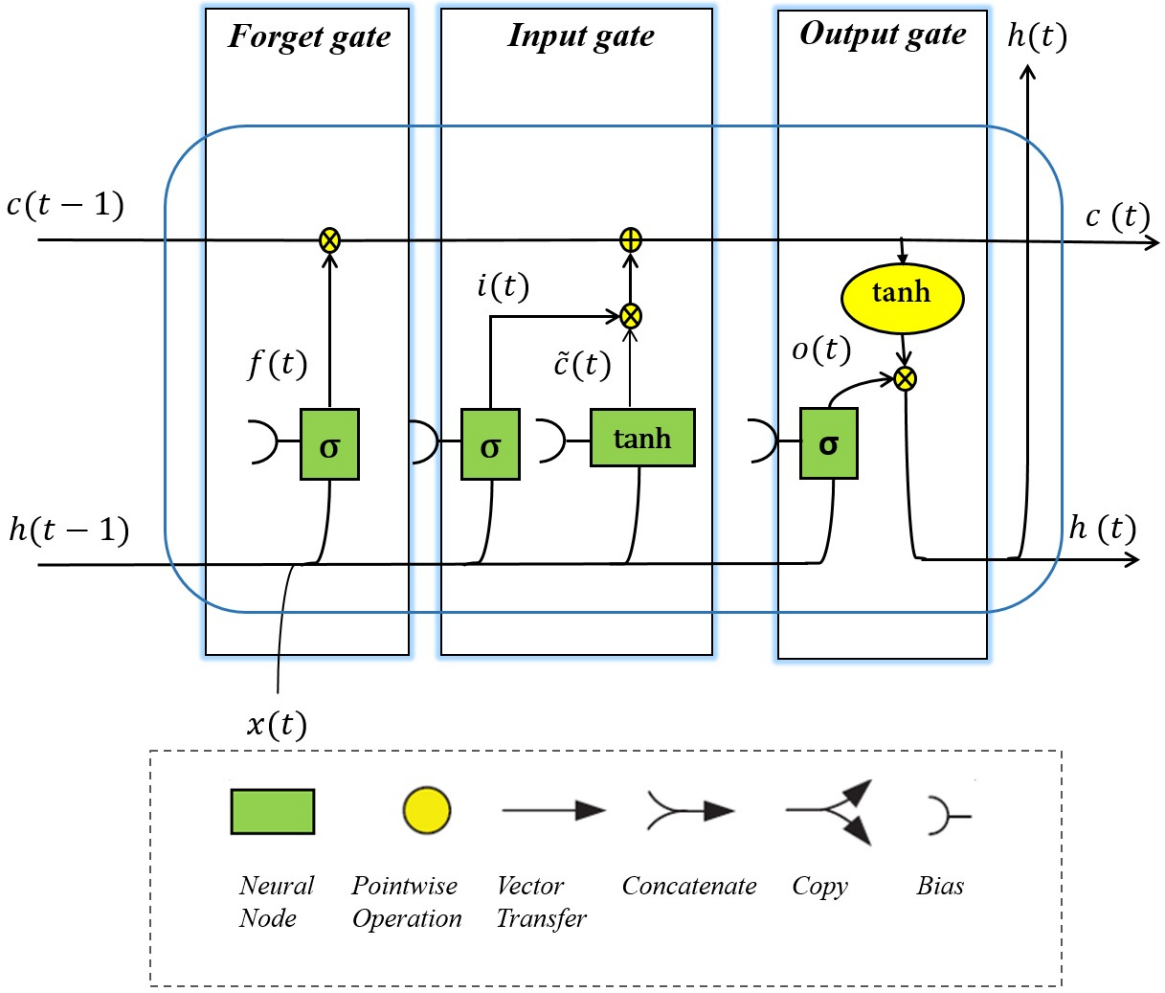


Figure I.1: Structure of a LSTM Cell.

I.5.1 Vanilla Recurrent Neural Networks

Each node of the RNN layer represents one time-step, receiving the input from the current time step x_t as well as the hidden state value of the hidden layer in the previous time step h_{t-1}. Consequently, given the input sequence $X = [x_1, x_2, \ldots, x_T]$, the sequence of hidden states h_t in matrix form yields

$$h_t = \psi(A_t) = \psi(W_x x_t + W_h h_{t-1} + b), \tag{I.2}$$

where $W_x \in \mathbb{R}^{d_h \times d_x}$ denotes the weight matrix from the input to hidden states, $W_h \in \mathbb{R}^{d_h \times d_h}$ denotes the hidden state to hidden state weight matrix, $b \in \mathbb{R}^{d_x}$ denote the input biases and $h_{t-1} \in \mathbb{R}^{d_h}$ denote the hidden state at time step $t-1$ or previous time step. It is to be noted here that unlike the FFNN, all the weight and biases are shared across the time steps which not only allows for keeping the amount of parameters under control but also for sharing knowledge across the time steps. Usually sigmoidal functions like the hyberbolic tangent are used as the activation function $\psi(\cdot)$.

A multi-layer RNN network is obtained by stacking a certain number of layers represented by Eq. (I.2) where the input data is subsequently passed through the layers. Due

to the loops implemented by the hidden state recurrence, the architectures theoretically allows information to adaptively persist for longer periods of time. However, the training of RNN using backpropagation through time is notoriously difficult due to the problem of vanishing or exploding gradients [32]. While exploding gradients can be avoided by applying hard constraint to the norm of the gradients, vanishing gradients remain a serious problem. This effect can only be solved by changes in the architecture as provided by LSTM or GRU.

I.5.2 Long-Short Term Memories

Originally, LSTM are proposed in [11] and are a special variant of RNN. LSTM have had great success recently in various domains of sequential modelling including but not limited to speech recognition and machine translation. The overall network is obtained similarly to the vanilla RNN by stacking multiple LSTM layers. However, the structure of the neural units varies considerably as is illustrated in Fig. I.1 showing just one unrolled LSTM cell, representing one time-step. The LSTM cell includes the block input x_t, three gates, namely the forget f_t, the input i_t and output gate o_t, a cell state c_t, a hidden state h_t and the output activation function ψ. The function of these sigmoidal gates is to determine how much information should the cell carry forward to the next time step. Specifically, the gates output a number between 0 and 1 which determines the amount of information to be retained and modified. For e.g., the forget gate determines the amount of incoming information should the cell keep and how much it should forget. The input gate determines how much new information is to be accumulated in the cell state. The output gate decides the output cell state c_t and the hidden state h_t of the LSTM cell. These outputs are fed to the next cell.

The vector equations for a vanilla LSTM layer in the forward pass can be written as summarized from the original work [11]

$$\tilde{c_t} = \gamma(W_z x_t + R_z h_{t-1} + b_z), \tag{I.3}$$
$$i_t = \sigma(W_i x_t + R_i h_{t-1} + b_i), \tag{I.4}$$
$$f_t = \sigma(W_f x_t + R_f h_{t-1} + b_f), \tag{I.5}$$
$$c_t = \tilde{c_t} \odot i_t + c_{t-1} \odot f_t, \tag{I.6}$$
$$o_t = \sigma(W_o x_t + R_o h_{t-1} + b_o), \tag{I.7}$$
$$h_t = \psi(c_t) \odot o_t, \tag{I.8}$$

where σ, γ and ψ are point-wise activation functions. In the vanilla LSTM, logistic sigmoid functions $\sigma(x) = 1/(1+e^{-x})$ are used for the gate activations while the hyperbolic tangent $\gamma(x) = \psi(x) = \tanh(x)$ is used as input and output activations. The matrices $W_z, W_i, W_f, W_o \in \mathbb{R}^{N\times M}$, $R_z, R_i, R_f, R_o \in \mathbb{R}^{N\times N}$ and vectors $b_z, b_i, b_f, b_o \in \mathbb{R}^N$ are the input, recurrent and bias weights, respectively where N denotes the size of the hidden layer per LSTM cell and M is the number of inputs or feature size. The initial states are given by $h_0 \in \mathbb{R}^N$ and $c_0 \in \mathbb{R}^N$.

In contrast to the vanilla RNN, an additional memory variable known as cell state c_t is introduced that can be considered as an information superhighway where the information flows through continuously in an uninterrupted fashion across many time steps due to only linear interactions with the other LSTM activations. Additionally, the sigmoid gates are

incorporated which control the information flow i.e. determine the amount of information to be retained from the previous time step or modified from input at the current time step. The output hidden cell state h_t, which is passed onto the next cell in the time-step and maybe a deeper LSTM cell, is a filtered version of the calculated cell state c_t. Since the LSTM cell has two distinct states as output and allowing the uninterrupted information flow, it mitigates to a certain extent the vanishing gradient problem. Therefore, the LSTM neural networks can learn long-term dependencies by modelling information over a long sequences.

I.5.3 Gated Recurrent Units

The GRU are introduced in [12] and provide a more simplified unit compared to the LSTM. A schematic is shown in Fig. I.2 pointing to the reset and update control gates instead of three gates in the LSTM cell and the absence of the cell state. The reset and update gate have a similar function to the forget gate an input gate in the LSTM cell but the difference is in how the output of these gates are used inside the GRU cell. Specifically, the reset and upgate gate are simultaneously used for the hidden output from the GRU cell. As a result, a GRU cell has less training parameters than an LSTM which makes the training faster. GRU were especially developed to capture dependencies of different time scales in machine translation tasks. The vector equations for a vanilla GRU layer in the forward pass can be written as summarized from the original work [12]

$$
\begin{aligned}
z_t &= \sigma(W_z x_t + R_z h_{t-1} + b_z), \\
r_t &= \sigma(W_r x_t + R_r h_{t-1} + b_r), \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \psi(W_h x_t + R_h(r_t \odot h_{t-1}) + b_h),
\end{aligned}
$$

where $x_t, z_t, r_t, h_t \in \mathbb{R}^N$ are input, update gate, reset gate and output vector, respectively and $W_z, W_r, W_h \in \mathbb{R}^{N \times M}$, $R_z, R_r, R_h \in \mathbb{R}^{N \times N}$ and $b_z, b_r, b_h \in \mathbb{R}^N$ are weight matrices and bias vectors.

Essentially, a GRU cell incorporates the two gates i.e. the forget and input gates of the LSTM cell into a solitary update gate. This gate then independently decide the amount of information to be passed from a preceding hidden states to the next hidden state. The reset gate r_t allows to drop information irrelevant for the future. The output hidden state h_t is calculated by the value from the reset gate is element-wise multiplied with the previous hidden state and added with a scaled current input. Consequently, GRU simplifies LSTM by eliminating certain parameters but still retains most of its fundamental properties.

I.5.4 Bidirectional RNN and LSTM

The first introduction of bidirectional RNN dates back to 1997 [33] wherein the idea is to connect two recurrent networks trained in opposite directions i.e. they are trained with the input sequence read once from the left and once from the right, eventually feeding into the same output layer. With this architecture, the network has for each data point full knowledge about the neighbouring points before and after it and hence, obtains more information in contrast to unidirectional RNN. Particularly, future input information can be accessed for the actual data sample which is beneficial if the actual classification

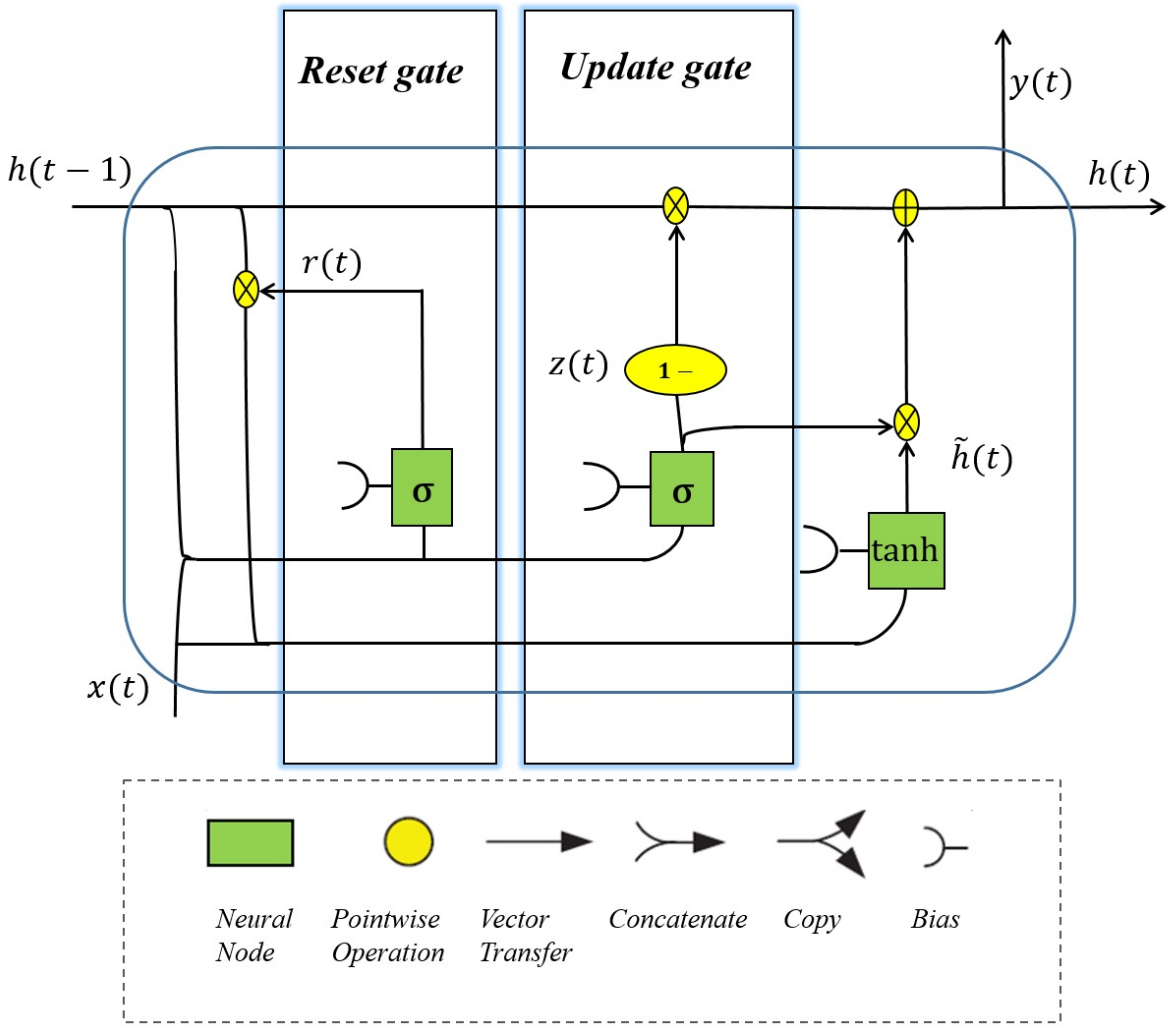


Figure I.2: Structure of a GRU cell

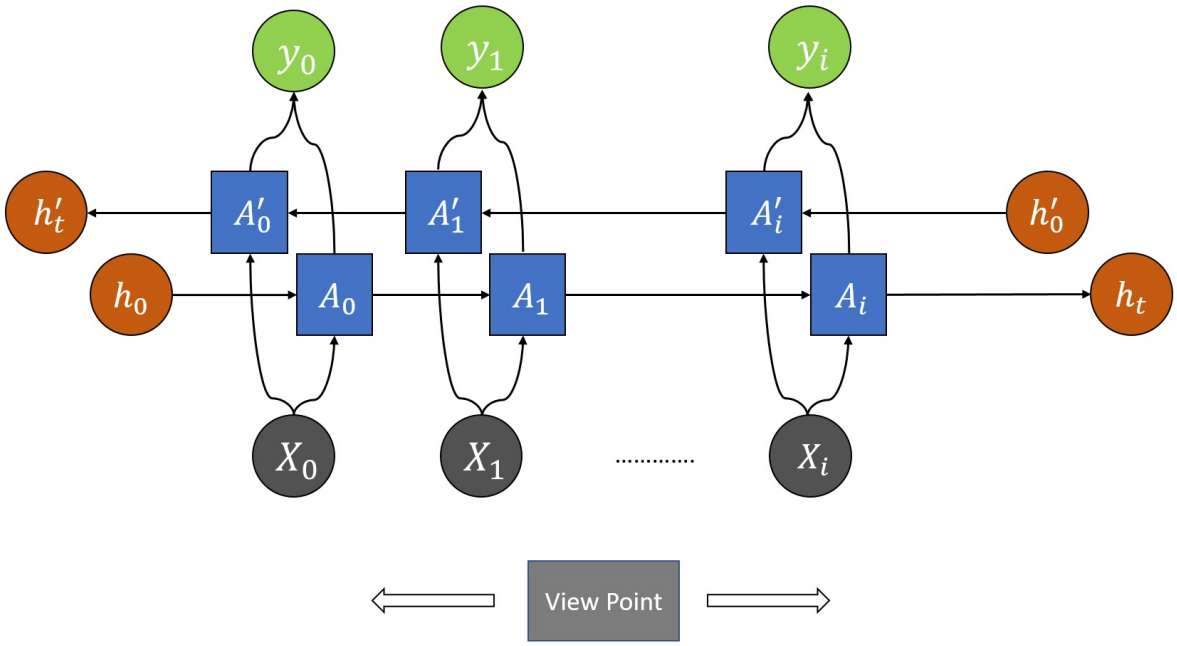


Figure I.3: Structure and Viewpoint of Bidirectional RNN network.

decision depends on the context, i.e. on both previous and future inputs. Consequently, as is illustrated in Fig. I.3, each node A_t and A'_t represents one cell of the bidirectional RNN layer and the sequence of hidden states h_t and h'_t for both the RNN in matrix form yields

$$h_t = \psi(A_t) = \psi(W_x x_t + W_h h_{t-1} + b) \tag{I.9}$$

$$h'_t = \psi(A_t) = \psi(W'_x x_t + W'_h h'_{t-1} + b'), \tag{I.10}$$

where $h_0 = h'_t$ and $h_t = h'_0$. The concept of bidirectionality was extended to LSTM units in [13] with the supplement of the two cell states along with the two hidden states. In practice, one may decide to combine the two hidden states and cell states from the two RNN or LSTM layers for eg. sum, average, merge or concatenate. Here we concatenate the two hidden and cell states, i.e. double the size of the hidden state and cell state. In the case of fault classification, a unidirectional model can only observe information about the system from one side of the sequence and try to predict fault case. A bidirectional model has the advantage of processing information from both sides of the sequence simultaneously which is shown in Eqn. (I.10).

Consequently, the main difference between LSTM and B-LSTM lies in the viewpoint of the network on the data sequence, on which it has to make a decision as illustrated in Fig. I.3. In vanilla LSTM the viewpoint for analysis of the multivariate time series lies at the end of the considered sequence, i.e. looking back on the past values. On the contrary, the viewpoint for B-LSTM is set in the middle of the considered sequence from where the complete sequence can be considered. This potentially allows for "overviewing" longer sequences and hence, more information can be stored by the memory element and subsequently taken into account for the analysis. We will show later on, that the sequence length indeed has an impact on the performance such that B-LSTM indeed provides some advantages in terms of performance compared to unidirectional RNN architectures.

I.6 Data Preprocessing for Deep Recurrent Architectures and Efficient Training

In this section, we propose a novel data preprocessing and restructuring process as well as a many-to-one training architecture for B-LSTM which considerably improves the efficiency of training.

I.6.1 Data Preprocessing and Restructuring

One of the challenges for training RNN and specifically LSTM is to handle the length of the raw time series signal (also referred to as sequence length) which is fed into the network. Theoretically, one could feed an entire dataset as a sequence length, however that would increase the model complexity exponentially and could also lead to overfitting as the model has to remember over the complete length of the dataset. As a solution, we propose a variable sequence length strategy which helps in creation of a better discriminative representation of the training data. The variable sequence length strategy is illustrated in Fig. I.4 where dataset from two classes i.e. No Fault (top) and Fault 1 (bottom) is shown. The complete feature space is denoted by variables 1 to 52 and the values of the variables are scaled between 0 and 1 for brevity. The proposed sliding window approach samples sequences from the two datasets and arranges them consecutively. This two step procedure is explained in detail below:

1. **Sequence Generation:** Sequences of length T (a hyperparameter) from both the class datasets (No fault and Faulty) are extracted. In the Fig. I.4, two sample sequences for each class dataset are denoted by sequence s and sequence $s+1$ in the No fault case and the sequence s' and sequence $s'+1$ in the Fault 1 case. For simplicity, the stride parameter is kept constant at 1 i.e. two consecutive sequences in a class dataset differ by 1 time-step. This step is performed iteratively over the complete sample space such that each class dataset is divided into multiple sequences. This step can be seen as a form of maximum information usage for the appropriate trend analysis, as there are samples from the training set that are used multiple times.

2. **Dataset Restructuring:** The sequences generated from each class is subsequently alternatively stacked as shown by the black, red and blue square boxes in Fig. I.4. Each of these square boxes denote a separate sequence which is generated and concatenated together. This step is performed to enforce the sequential information flow, which is extremely important to any type of RNN model. Since the goal is to predict a discrete class at the end of each sequence, the training dataset is restructured in such a way that the model is able to attain as much information about the difference in each of these classes. Specifically, the model should be able to differentiate among the different classes more often during the sequential fed in of the training data. Another advantage of this procedure is that when the input sequence length to the RNN model is longer than the individual sequences generated from each class dataset, then the model will be able to see both the class information in one sequence. Therefore, we introduce a new data-partitioning hyperparameter,

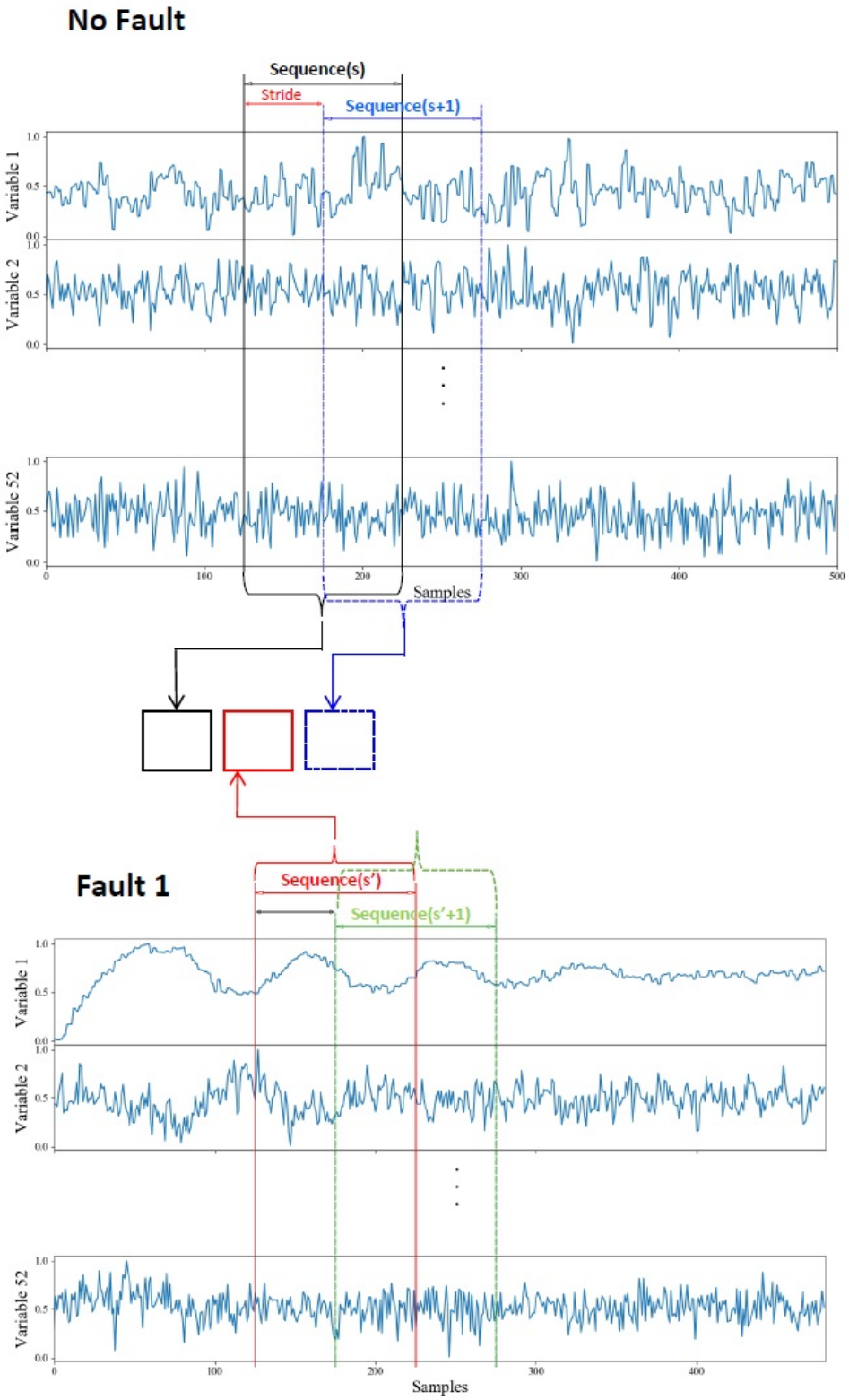


Figure I.4: Sliding Window Approach for Data Representation

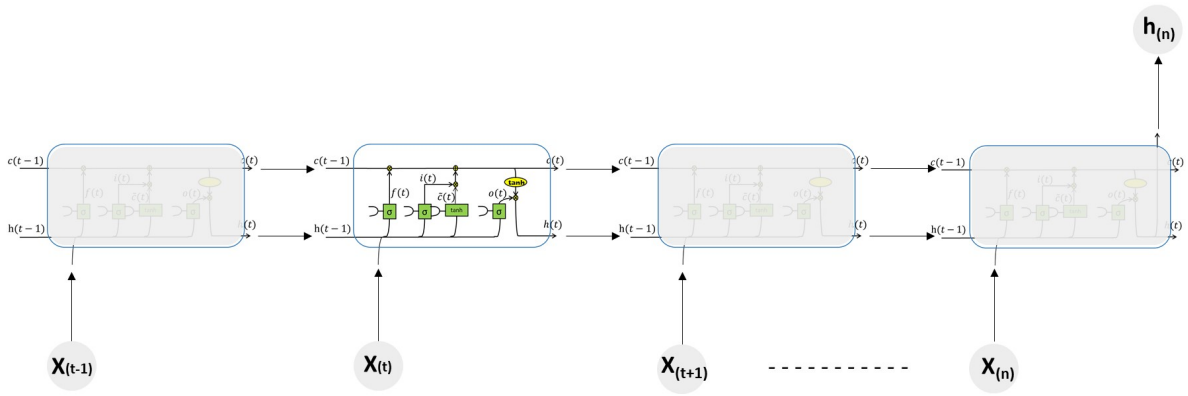


Figure I.5: A many to one representation of a LSTM layer.

β which signifies the minimum number of sequences of each individual class dataset in the input sequence length to the RNN model.

The benefit of these approaches is twofold: First, feeding the data in parallel and with a longer sequence length T facilitates the information storage by the LSTM cells over long time horizons. Second, due to the sliding window operation, each data point is shown numerous times to the network which effectively implements a form of data augmentation.

I.6.2 Training of a Many to One Recurrent Architecture

Since the objective is to have a sequence classifier, we propose a many-to-one recurrent architecture as illustrated in Fig. I.5, with the raw sensor data as input sequence and the output from the layer being just the hidden state of the last cell. This hidden state of the last cell contains the information over the complete sequence because all of the previous hidden states are used to calculate and update it. Since in this study, we consider a single LSTM layer, the hidden state of the last LSTM cell is fed as input to the Softmax activation function [34], creating a probability distribution over the labelled classes. Formally, the softmax function is given by

$$softmax(z_i) = \frac{\exp(z_i)}{\sum_k \exp(z_k)}, \tag{I.11}$$

such that it exponentiates and normalizes z_i such that the probability prediction from the neural network is as close as possible to the target label y_i. This softmax output and the target label are fed in Eq. (I.1) for the loss and the subsequent loss gradient calculation with respect to every weight parameter from the back-propagation [10] algorithm. After the gradients are calculated, we use the Adam [35] optimizer for updating the weight parameters. We remark that from a fault detection viewpoint, a sequence prediction as proposed by the many-to-one architecture is naturally representing trends in the data set which are often important patterns and hence, increases detection performance.

The overall training procedure is described in Algorithm 1 wherein a bank of recurrent architectures are trained for fault classification. All the networks are trained individually with specific training dataset corresponding to the respective fault cases. The train validation split was set at 20% for hyperparameter selection based on the performance on

the validation set. Thereafter, all the trained networks are tested on their respective test dataset for evaluating their generalization performance.

Algorithm 2 Training Procedure of a Recurrent Architecture for Fault Classification

1: Normalize the complete training and testing data with zero mean and unit standard deviation.
2: Randomly divide 20 % of the training set into validation set for hyperparameter tuning.
3: Preprocess and partition the training data as described in Sec. I.6.1.
4: Train a bank of RNN classifiers with the backpropagation algorithm for each of the fault cases with a set of hyper-parameters tuned based on the performance on the validation set.
5: Evaluate the generalization ability of the classifiers on the test set with the F_1 score.

I.6.3 Evaluation Metric

An appropriate choice of an evaluation metric is essential for determining the performance of a predictive model on a given classification task, especially when the given test set is imbalanced. The right evaluation metric should be chosen for an unbiased and neutral estimate of the model's predictions. Otherwise the model will always favour the dominant class. Therefore, the F_1 measure [36] was chosen in this study, since it considers an even balance for the false positive and false negative classified samples. In this way, we combine the Fault Detection Rate (FDR) and False Alarm Rate (FAR) evaluation metrics, which have been widely used in previous studies [17], into a single evaluation metric for binary fault classification case. An additional advantage of the F_1 measure is that in contrast to theFDR and FAR measures used in [37, 38]. Formally, Precision and Recall are defined as

$$I_{Precision} = \frac{TP}{TP + FP}, \tag{I.12}$$

$$I_{Recall} = \frac{TP}{TP + FN}. \tag{I.13}$$

such that True Positives (TP) denotes the accurately classified faulty samples, False Positives (FP) denotes the missclassified faulty samples and False Negatives (FN) denotes the missclassified non-faulty operating condition samples. F_1 score is then derived as the harmonic mean of Precision and Recall to give a much more balanced measure in case of class imbalance.

$$F_1 = \frac{2I_{Precision}I_{Recall}}{I_{Precision} + I_{Recall}} = \frac{2TP}{2TP + FP + FN}. \tag{I.14}$$

Evidently, if either of Precision or Recall is extremely small, the F1 Score will remain closer to the smaller value giving a much better overall estimate of the model's performance.

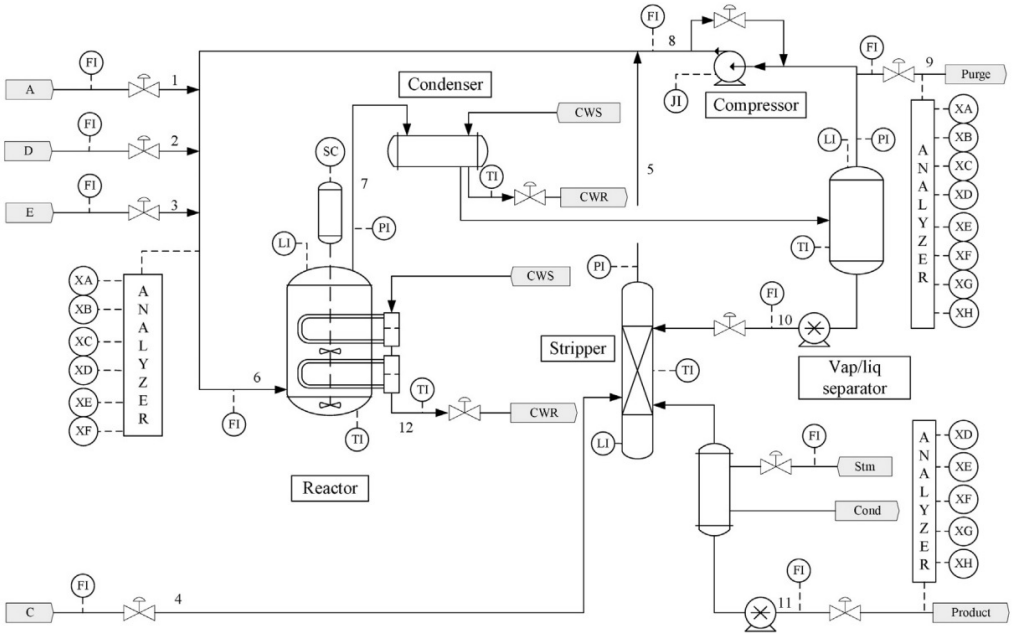


Figure I.6: Piping and instrumentation diagrams of the considered Tennessee Eastman Process [43]

I.7 Experimental Results and Comparison Study

We provide a thorough comparison of the proposed recurrent network structures on condition monitoring and fault diagnosis tasks. To this end, we employ the well known benchmark TE process to evaluate their performance. All the experiments were carried out in a Python 3.6 environment in that the NumPy [39] and Pandas [40] packages were used for data-preprocessing, Keras [41] were used for model building and Scikit-learn [42] was used for evaluation scores.

I.7.1 Tennessee Eastman Process

The TE process has initially been proposed in [43]. According to the process schematic illustrated in Fig.I.6 [43] the TE process includes a reactor unit, condenser, compressor, separator, and stripper. More specifically, four reactants (A, C, D, E), a byproduct (F) and an inert element (B) are transformed into two chemicals (G, H). The product in the form of steam from the reactor is cooled down by the condenser, subsequently fed to the vapour-liquid separator and recycled to the reactor feed using a compressor. The remaining condensed Steam 10 is pumped to a stripper where the remaining reactants from stream 10 are stripped by Stream 4. The products G and H exit from the bottom of the stripper and are used for a downstream process. The physical dynamics of the process owing to the irreversible and exothermic chemical reactions are difficult to derive, hence providing a typical example for applying and evaluating data-driven techniques for fault detection as e.g. shown in [20, 44, 18].

The TE process outputs 41 measured and 12 manipulated variables under closed loop condition. To evaluate classification models, 22 training and 22 testing data sets corresponding to the normal operation as well as 21 pre-programmed process faults proposed

Table I.1: Description of Process Faults in TE

Faults	Description	Type
1	A / C feed ratio (B composition constant)	Step Change
2	B composition (A / C feed ratio constant)	Step Change
3	D Feed temperature	Step Change
4	Reactor cooling water inlet temperature	Step Change
5	Condenser cooling water inlet temperature	Step Change
6	A Feed loss	Step Change
7	C header pressure loss	Step Change
8	A, B, C feed composition	Random Variation
9	D feed temperature	Random Variation
10	C feed temperature	Random Variation
11	Reactor cooling water inlet temperature	Random Variation
12	Condenser cooling water inlet temperature	Random Variation
13	Reaction kinetics	Slow drift
14	Reactor cooling water valve	Sticking
15	Condenser cooling water valve	Sticking
16 - 20	Unknown	Unknown
21	The valve fixed at steady state position	Constant position

in [45] and shown in Table I.1. The faulty training sets consist of 480 samples. The training and testing data is collected with a sampling interval of 3 minutes. The test set includes 960 samples representing 48 hours of plant operation, where the fault is injected after 8 hours corresponding to 160 samples. Therefore, we evaluate the last 800 samples for the testing of each fault cases.

I.7.2 Fault Classification Analysis

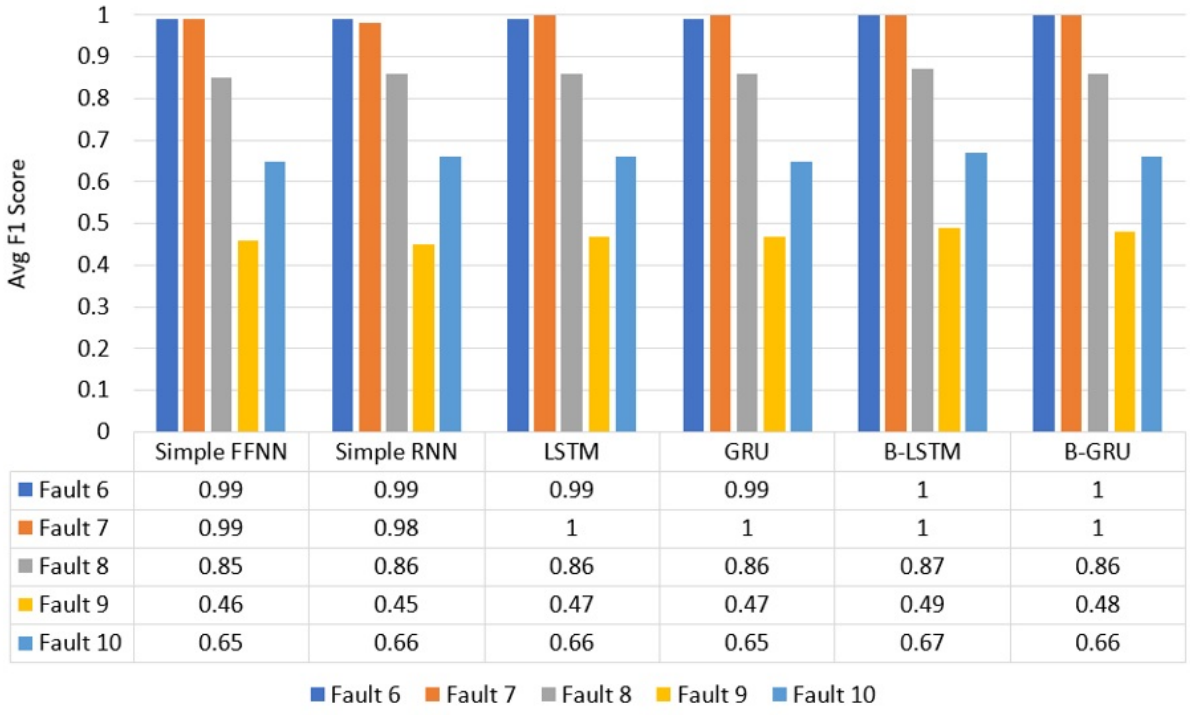
We start with a performance comparison of the different architectures on the binary fault classification case on TE process. To this end, we first categorize the 21 fault cases based on the order of difficulty to detect. In fact, some faults are considerably more difficult to detect than others due to the varying effects of the faults on the process variables which deviate substantially or trivially from normal operation. This phenomenon has also been observed in previous studies as e.g. in [17]. Since in the binary classification case with a nearly balanced test set, we set the baseline of the F_1 score to be 0.6. The faults are categorized based on this baseline. The fault cases are categorized as follows:

- Easy – Average F1 Score > 0.9
- Medium – Average F1 Score > 0.6 but <0.9
- Hard – Average F1 Score < 0.6

The architecture of the RNN were chosen to have similar properties and have been described in Table I.2. The number of layers and the hidden layer size were chosen after extensive tuning of the hyperparameters and the respective performance on the

Table I.2: Architectural Detail of the Baseline Recurrent Neural Networks used for Fault Categorization

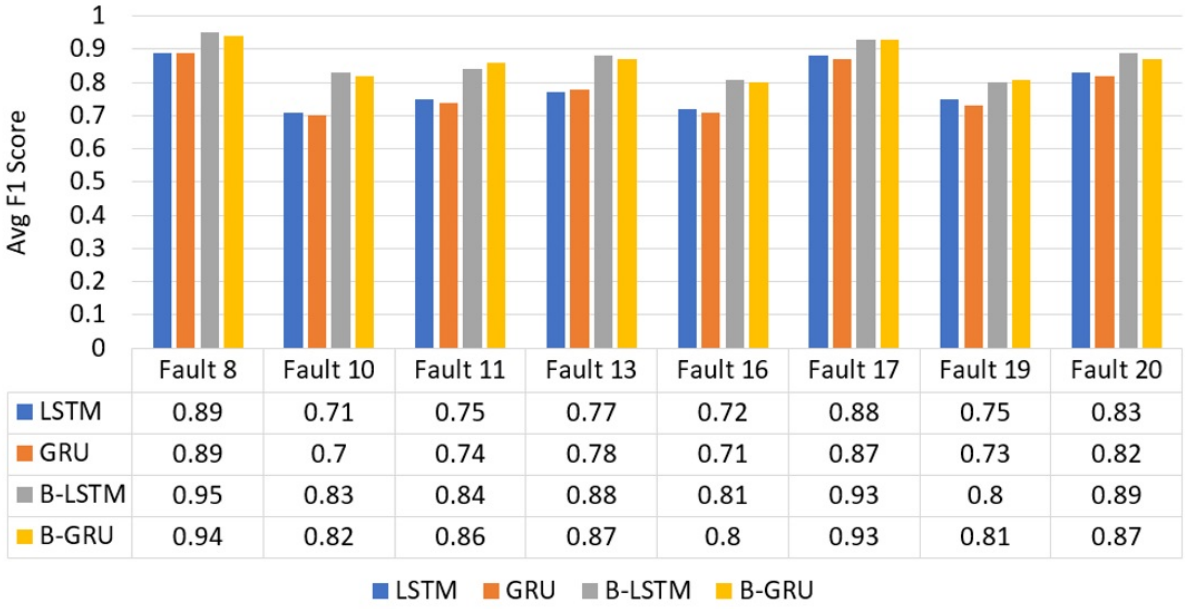
Type	Hidden Size	Window Size
FFNN	64	-
Simple RNN	64	10
LSTM	64	10
GRU	64	10
B-LSTM	64	10
Bi-GRU	64	10



	Simple FFNN	Simple RNN	LSTM	GRU	B-LSTM	B-GRU
Fault 6	0.99	0.99	0.99	0.99	1	1
Fault 7	0.99	0.98	1	1	1	1
Fault 8	0.85	0.86	0.86	0.86	0.87	0.86
Fault 9	0.46	0.45	0.47	0.47	0.49	0.48
Fault 10	0.65	0.66	0.66	0.65	0.67	0.66

Figure I.7: Comparison of a Feedforward Network with different RNN architectures

validation sets. Through the fault categorization results over all the baseline models and no data restructuring the fault cases $\{1, 2, 4-7, 12, 14, 18\}$ were categorized as easy detectable fault cases, the fault cases $\{8, 10, 11, 13, 16, 17, 19, 20\}$ as medium and the fault cases $\{3, 9, 15, 21\}$ as hard detectable. An important thing to be noted here is that the performance of a simple Feedforward Neural Network was comparable to the other RNN architectures on most of the fault cases even though some of the RNN architectures are more complex. An example is shown in Fig. I.7 where performance on fault cases 6 - 10 is being compared where a significant performance improvement is not observable in all the fault cases with the maximum improvement being 0.01 in the F_1 score. This can be explained by the relatively smaller window size and no data restructuring during the training of the RNN architectures. Furthermore, most of the easy categorized fault cases had an average F_1 score of 0.99 with the B-LSTM. Therefore, in the next subsection we focus on improving the prediction capability of the model on the hard and medium



	Fault 8	Fault 10	Fault 11	Fault 13	Fault 16	Fault 17	Fault 19	Fault 20
LSTM	0.89	0.71	0.75	0.77	0.72	0.88	0.75	0.83
GRU	0.89	0.7	0.74	0.78	0.71	0.87	0.73	0.82
B-LSTM	0.95	0.83	0.84	0.88	0.81	0.93	0.8	0.89
B-GRU	0.94	0.82	0.86	0.87	0.8	0.93	0.81	0.87

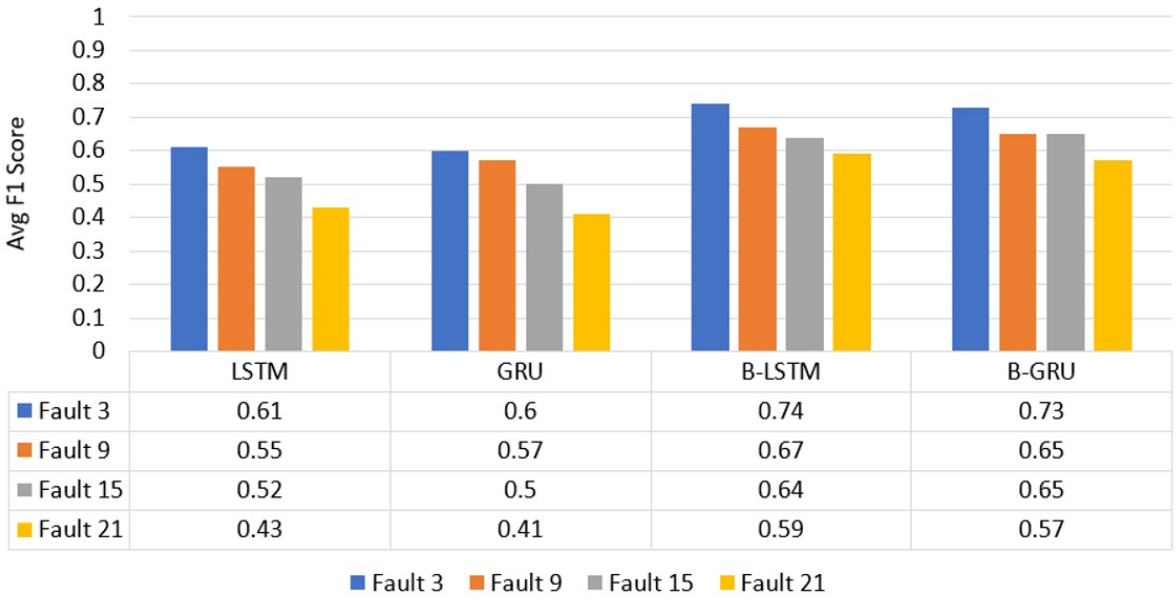
Figure I.8: Comparison of all RNN architectures for Medium Categorized Faults

categorized faults only. These easy category faults have also been found in literature to have very high FDR [17].

Results on hard and medium categorized faults

This section reports the comparison of the generalization capabilities of the RNN architectures after data partitioning and a longer sequence length. In order to detect the incipient fault cases, we formulate longer training sequences with data partitioning to provide the model the opportunity to understand the difference between the examples of the two classes. We report the improvement in model performance for incipient fault cases on both these accounts with the RNN architectures. After the experimental results, it was observed that the B-LSTM architecture has a consistently better generalization capability than the other architectures as illustrated in Fig. I.8 and Fig. I.9. Additionally, the other bidirectional architecture, Bidirection Gated Recurrent Unit (B-GRU) performs better than the other architectures, emphasizing our hypothesis from section I.5.4 that bidirectional architectures are better suited for longer sequences. The only change to the baseline RNN architectures here is the bigger sequence length, $T = 200$ with the partitioning parameter $\beta = 2$. Just to put the scale of improvement into perspective, the significant increase in the average F_1 score before and after data restructuring (partitioning) for the B-LSTM model was from 0.63 to 0.81 on the fault case 10. This performance improvement was noticeable with all the other RNN architectures exemplifying the applicability of the developed approach.

To further analyse the capability of the methods, Fig. I.10 and Fig. I.11 display side by side the significant increase in the fault detection capability of the B-LSTM model for incipient fault cases. Both the charts show the effectiveness of the proposed data-restructuring strategy proposed in Section I.6.1. There has been a significant improvement in the Average F_1 score in all the fault cases tested on TE dataset. For the hard categorized



	LSTM	GRU	B-LSTM	B-GRU
Fault 3	0.61	0.6	0.74	0.73
Fault 9	0.55	0.57	0.67	0.65
Fault 15	0.52	0.5	0.64	0.65
Fault 21	0.43	0.41	0.59	0.57

Figure I.9: Comparison between the RNN architectures for Hard Categorized Faults

fault cases, the average F_1 score of the best performing model B-LSTM on fault case 3 improved from 0.59 to 0.73.

Another important phenomenon is the effect of sequence length on the model's performance. By having a larger amount of data in the time dimension and the partition parameter $\beta \geq 2$, the model is able to discriminate among the different classes in the input sequence, yielding far better results than the models with shorter sequence length. This is illustrated in Fig. I.12 where the B-LSTM models with time-step 200, 100 and 50 with $\beta = 2$ on hard fault cases where the bigger sequence length yield a better performance in all the hard fault cases tested. A larger sequence length also means a more complex model. Therefore, it is important to note that this behaviour of the increase in performance with increase in sequence length is only relevant for the hard and medium categorized fault cases. On the easy category fault cases this leads to over-fitting at the start of the training process and thereby under-par performance. Therefore, determining an optimal sequence length is an iterative task for deciding the complexity of the model for the specific fault case.

I.7.3 Comparison with literature

In this section, we provide a comparison with the existing literature on the benchmark TE process namely [17, 46, 47, 48]. Unfortunately, the other related literature in [44, 20] does not report results on the incipient fault cases specifically fault cases 3,9,15 and 21. As mentioned above in subsection I.6.3, there are different evaluation metrics used in these previous studies except in [48]. For brevity, we chose the FDR% which ranges from $[0, 100]$ from the previous studies as the comparison statistic. To keep the comparison among the different evaluation metrics uniform, the F_1 score which actually ranges from $[0, 1]$ is multiplied by 100 to have the same order with FDR%. We also chose the best performing model from the respective studies which are Independent Component Analysis, Dynamic

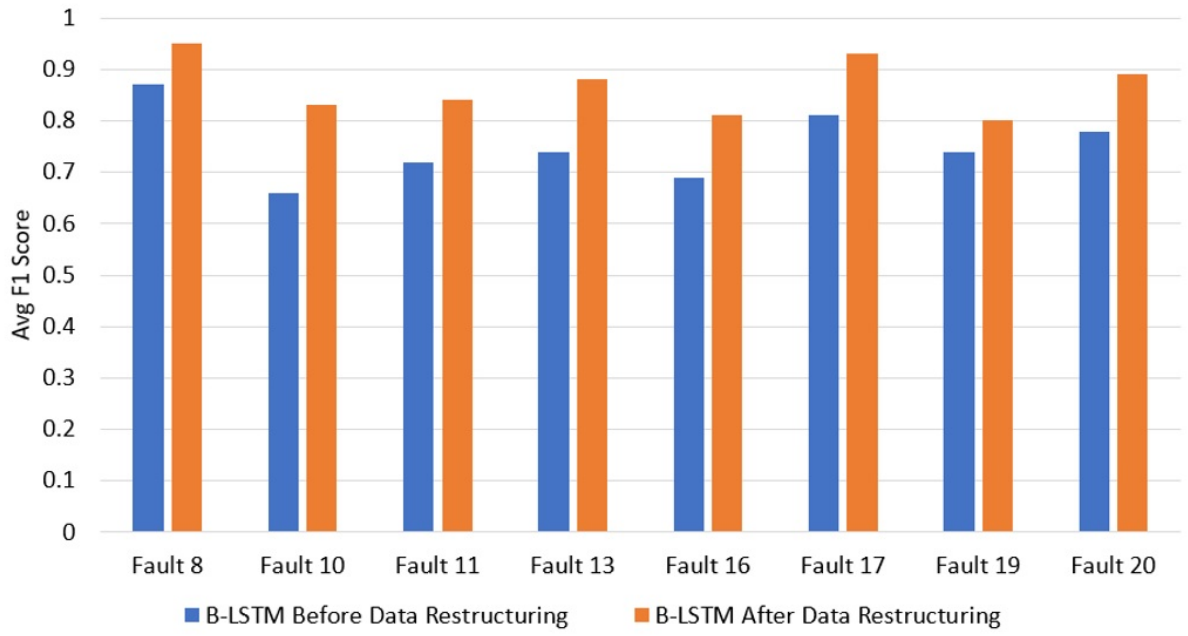


Figure I.10: B-LSTM Comparison before and after data partitioning for Medium Faults

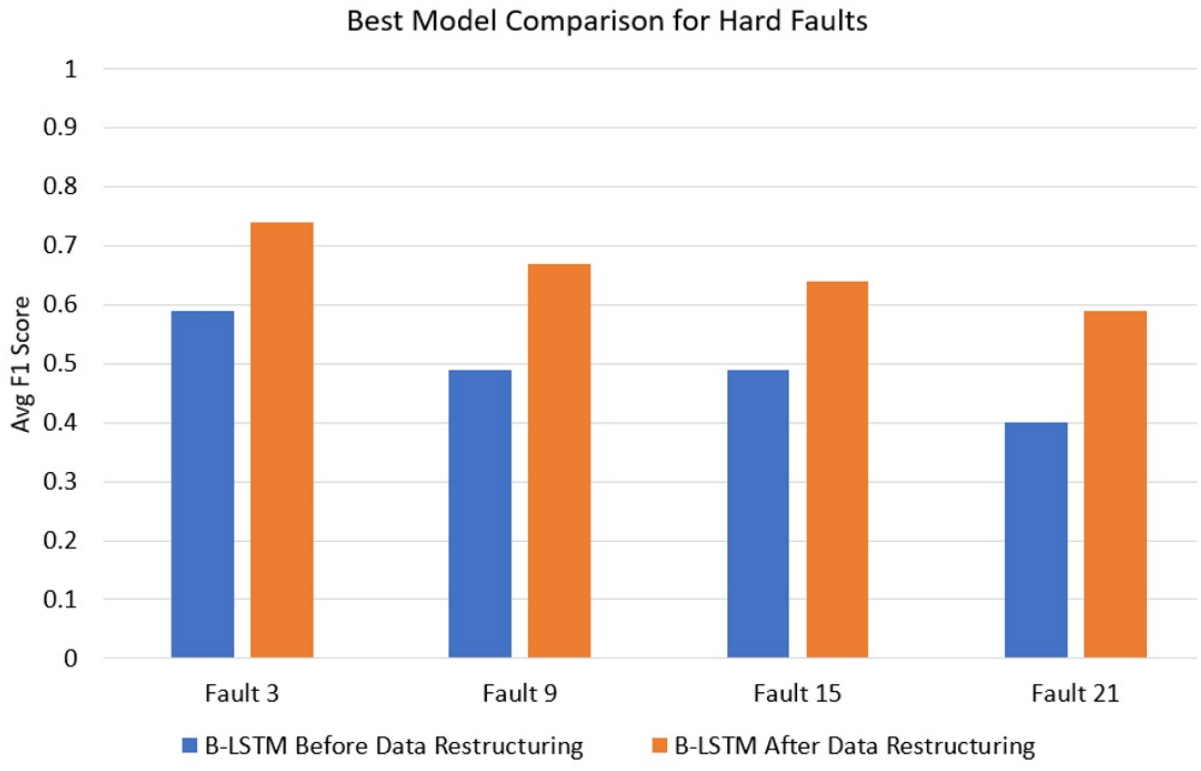


Figure I.11: B-LSTM Comparison before and after data partitioning for Hard Faults

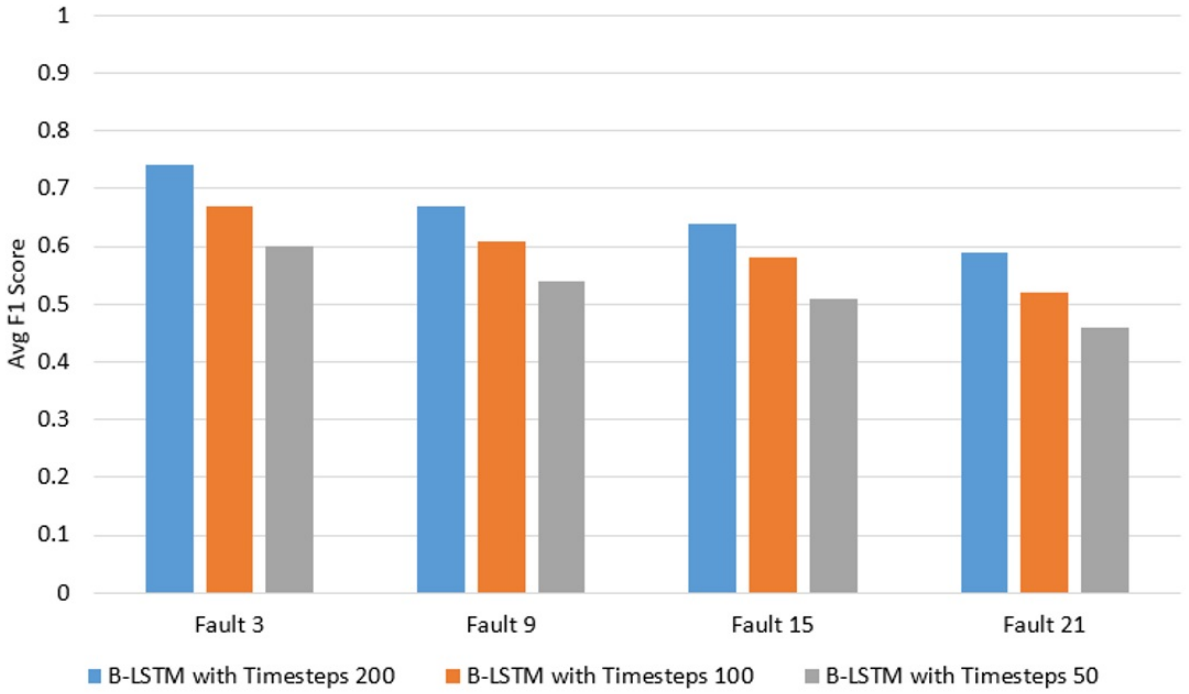


Figure I.12: Effect of Sequence length on the Model's performance

Table I.3: Performance Comparison with Existing Literature

Fault Case	B-LSTM	[17]	[46]	[47]	[48]
1	*100*	*100*	99.6	*100*	91.39
2	*100*	98.25	98.5	99	87.96
3	*73*	4.5	2.1	1.4	50.59
4	*100*	*100*	99.8	*100*	99.73
5	*100*	*100*	99.9	*100*	90.35
6	*100*	*100*	99.9	*100*	91.5
7	*100*	*100*	99.9	*100*	91.55
8	95	98.25	*98.5*	98.4	82.95
9	*67*	4.75	2	0.7	49.53
10	83	89.25	*95.6*	90.1	70.05
11	84	78.88	*96.5*	80.5	60.16
12	98	99.88	99.8	*100*	86.66
13	88	95.25	95.8	*96*	46.92
14	99	*100*	99.8	*100*	88.68
15	*64*	7.75	38.5	9.7	43.54
16	81	92.38	*97.6*	91.6	66.84
17	93	96.88	*97.6*	97.6	77.11
18	*98*	90.5	90.5	90.8	82.74
19	80	92.88	97.1	*98.1*	70.87
20	89	*91.38*	90.8	91.3	72.88
21	59	56.38	53.9	*65.8*	31.26
Overall	*88.14*	80.82	83.51	81.48	73.01

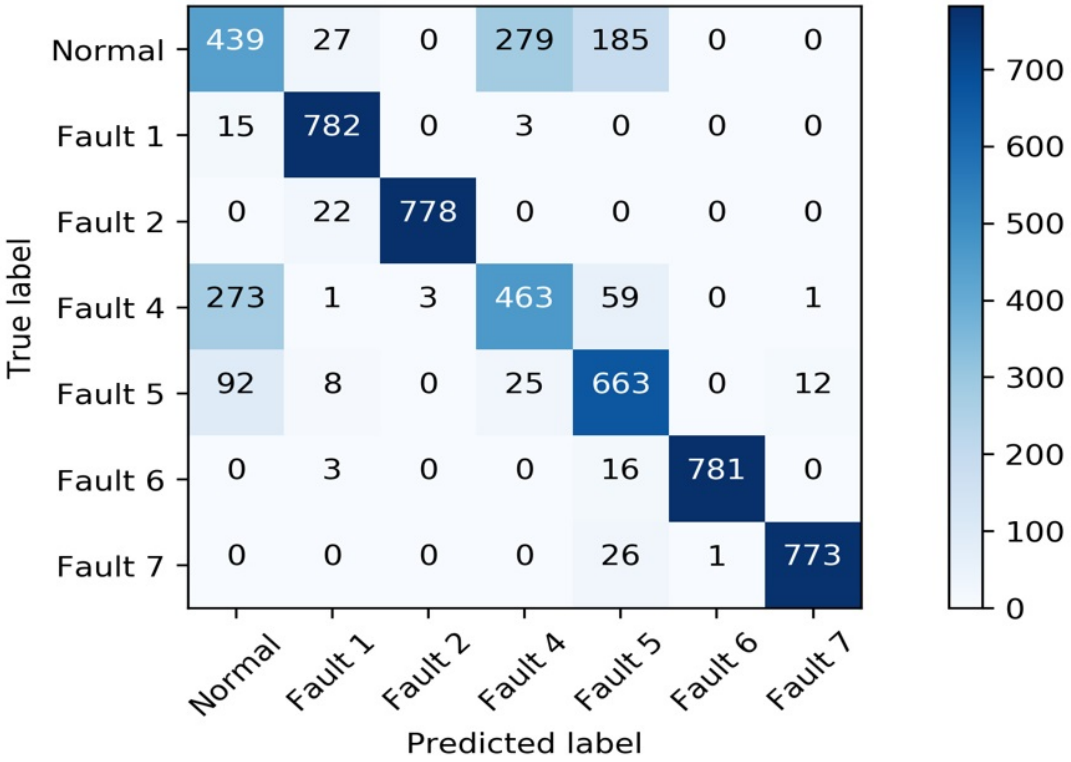


Figure I.13: Multi Class Fault Classification Confusion Matrix

Principal Component Analysis with decorrelated residuals and canonical variate analysis and CNN from [17, 46, 47, 48] respectively. The result comparison is given in Table I.3 in which we highlight the best performing algorithm in blue. The comparison shows a considerable increase in the performance on five different fault cases compared to existing approaches. The performance improvement is particularly noticeable on the hard faults 3,9 and 15 which are unpredictable before. This strong performance gain of the proposed B-LSTM model emphasizes its applicability to industrial processes. Furthermore, the overall performance of the B-LSTM shows an increase of around 4.6% compared to the second best performing approach which underlines the general applicability of the B-LSTM.

I.7.4 Results for Multi-class Fault Identification

The results of the best performing B-LSTM model on multi class fault classification is described here wherein 6 easy categorized fault cases and the normal operating condition is chosen. Multi-class classification on the hard categorized faults is part of the future research works. The architecture of the model is same as the binary classification, with the only difference being the number of neurons in the final fully connected layer chosen to be 7. A sequence length of $T = 175$ and the data restructuring parameter $\beta = 1$ is set after hyperparameter tuning. Observing the results from the confusion matrix in Fig. I.13, it is clear that the model is not able to completely differentiate among the fault case 4 and 5 with the normal operating condition although in the binary case the B-LSTM model had an F_1 score of 1. It can be concluded that for the multi-class case more hyper-parameter

tuning, testing deeper models and efficient training procedures need to be defined.

I.8 Conclusions

In this paper we propose a novel recurrent neural network architecture for time-series based condition monitoring and fault detection on the industrial benchmark TE process. The framework is based on the recently introduced B-LSTM which allow for incorporating more knowledge in a training sequence about the neighbouring points compared to its unidirectional counterparts where this feature is missing. In contrast to existing approaches using sequential data processing, we use a many to one prediction architecture which allows to process an input sequence at once. Using the many to one architecture with a sliding window data operation eases the long term data storage in the LSTM cells and hence allows for considering even longer time series patterns. Furthermore, a new data restructuring technique is presented to attenuate the sequence dependency among classes. The novel approach provides superior results compared to the existing results on the TE fault detection benchmark. We conduct various experiments including comparisons to vanilla LSTM and vanilla GRU and the impact of the sequence length on the fault detection capabilities with an emphasis on hard to detect faults.

Future work will additionally consider multi-class fault classification for all the fault cases. For the architecture development, we will further experiment with attention mechanism and how to effectively apply them in condition monitoring. Furthermore, alternative architectures like convolutional neural networks will be checked with respect to their performance and compared to the proposed B-LSTM.

Bibliography

[1] G. S. Chadha and A. Schwung, "Comparison of deep neural network architectures for fault detection in tennessee eastman process," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, 2017.

[2] Z. Gao, C. Cecati, and S. X. Ding, "A survey of fault diagnosis and fault-tolerant techniques—part i: Fault diagnosis with model-based and signal-based approaches," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 6, pp. 3757–3767, 2015.

[3] R. Milne, C. Nicol, L. Travé-Massuyès, and J. Quevedo, "Tiger: knowledge based gas turbine condition monitoring," *Ai Communications*, vol. 9, no. 3, pp. 92–108, 1996.

[4] Y. Lei, B. Yang, X. Jiang, F. Jia, N. Li, and A. K. Nandi, "Applications of machine learning to machine fault diagnosis: A review and roadmap," *Mechanical Systems and Signal Processing*, vol. 138, p. 106587, 2020.

[5] S. Khan and T. Yairi, "A review on the application of deep learning in system health management," *Mechanical Systems and Signal Processing*, vol. 107, pp. 241–265, 2018.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[7] Y. Lei, F. Jia, J. Lin, S. Xing, and S. X. Ding, "An intelligent fault diagnosis method using unsupervised feature learning towards mechanical big data," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 5, pp. 3137–3147, 2016.

[8] Z. Zhang and J. Zhao, "A deep belief network based fault diagnosis model for complex chemical processes," *Computers & chemical engineering*, vol. 107, pp. 395–407, 2017.

[9] W. Sun, S. Shao, R. Zhao, R. Yan, X. Zhang, and X. Chen, "A sparse auto-encoder-based deep neural network approach for induction motor faults classification," *Measurement*, vol. 89, pp. 171–178, 2016.

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[11] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[12] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1724–1734, Association for Computational Linguistics, 2014.

[13] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural Networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[14] V. Venkatasubramanian, R. Rengaswamy, S. N. Kavuri, and K. Yin, "A review of process fault detection and diagnosis: Part iii: Process history based methods," *Computers & chemical engineering*, vol. 27, no. 3, pp. 327–346, 2003.

[15] S. Yin, S. X. Ding, X. Xie, and H. Luo, "A review on basic data-driven approaches for industrial process monitoring," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 11, pp. 6418–6428, 2014.

[16] Z. Gao, C. Cecati, and S. X. Ding, "A survey of fault diagnosis and fault-tolerant techniques—part ii: Fault diagnosis with knowledge-based and hybrid/active approaches," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 6, pp. 3768–3774, 2015.

[17] S. Yin, S. X. Ding, A. Haghani, H. Hao, and P. Zhang, "A comparison study of basic data-driven fault diagnosis and process monitoring methods on the benchmark tennessee eastman process," *Journal of Process Control*, vol. 22, no. 9, pp. 1567–1581, 2012.

[18] M. Kano, K. Nagao, S. Hasebe, I. Hashimoto, H. Ohno, R. Strauss, and B. R. Bakshi, "Comparison of multivariate statistical process monitoring methods with applications to the eastman challenge problem," *Computers & chemical engineering*, vol. 26, no. 2, pp. 161–174, 2002.

[19] Q. P. He and J. Wang, "Fault detection using the k-nearest neighbor rule for semiconductor manufacturing processes," *IEEE Transactions on Semiconductor Manufacturing*, vol. 20, no. 4, pp. 345–354, 2007.

[20] S. Yin, X. Gao, H. R. Karimi, and X. Zhu, "Study on support vector machine-based fault detection in tennessee eastman process," *Abstract and Applied Analysis*, vol. 2014, no. 8, pp. 1–8, 2014.

[21] Y. Maki and K. A. Loparo, "A neural-network approach to fault detection and diagnosis in industrial processes," *IEEE Transactions on Control Systems Technology*, vol. 5, no. 6, pp. 529–541, 1997.

[22] Y. Qi, C. Shen, D. Wang, J. Shi, X. Jiang, and Z. Zhu, "Stacked sparse autoencoder-based deep network for fault diagnosis of rotating machinery," *IEEE Access*, vol. 5, pp. 15066–15079, 2017.

[23] M. Xia, T. Li, L. Xu, L. Liu, and C. W. de Silva, "Fault diagnosis for rotating machinery using multiple sensors and convolutional neural networks," *IEEE/ASME Transactions on Mechatronics*, vol. 23, no. 1, pp. 101–110, 2018.

[24] R. Liu, G. Meng, B. Yang, C. Sun, and X. Chen, "Dislocated time series convolutional neural architecture: An intelligent fault diagnosis approach for electric machine," *IEEE Transactions on industrial informatics*, vol. 13, pp. 1310–1320, 2017.

[25] L. Guo, Y. Lei, S. Xing, T. Yan, and N. Li, "Deep convolutional transfer learning network: A new method for intelligent fault diagnosis of machines with unlabeled data," *IEEE Transactions on Industrial Electronics*, vol. 66, no. 9, pp. 7316–7325, 2019.

[26] N. Mehdiyev, J. Lahann, A. Emrich, D. Enke, P. Fettke, and P. Loos, "Time series classification using deep learning for process planning: A case from the process industry," *Procedia Computer Science*, vol. 114, pp. 242–249, 2017.

[27] W. Zhang, W. Guo, X. Liu, Y. Liu, J. Zhou, B. Li, Q. Lu, and S. Yang, "Lstm-based analysis of industrial iot equipment," *IEEE Access*, vol. 6, pp. 23551–23560, 2018.

[28] T. de Bruin, K. Verbert, and R. Babuška, "Railway track circuit fault diagnosis using recurrent neural networks," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 3, pp. 523–533, 2017.

[29] J. Lei, C. Liu, and D. Jiang, "Fault diagnosis of wind turbine based on long short-term memory networks," *Renewable Energy*, vol. 133, pp. 422–432, 2019.

[30] H. Zhao, S. Sun, and B. Jin, "Sequential fault diagnosis based on lstm neural network," *IEEE Access*, vol. 6, pp. 12929–12939, 2018.

[31] W. de Mulder, S. Bethard, and M.-F. Moens, "A survey on the application of recurrent neural networks to statistical language modeling," *Computer Speech & Language*, vol. 30, no. 1, pp. 61–98, 2015.

[32] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *2013 International conference on machine learning*, pp. 1310–1318, 2013.

[33] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[34] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[35] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv e-prints*, p. arXiv:1412.6980, 2014.

[36] H. M and S. M.N, "A review on evaluation metrics for data classification evaluations," *International Journal of Data Mining & Knowledge Management Process*, vol. 5, no. 2, pp. 01–11, 2015.

[37] F. Lv, C. Wen, Z. Bao, and M. Liu, "Fault diagnosis based on deep learning," in *2016 American Control Conference (ACC)*, pp. 6851–6856, 2016.

[38] D. Xie and L. Bai, "A hierarchical deep neural network for fault diagnosis on tennessee-eastman process," in *2015 IEEE 14th International Conference on Machine Learning and Applications*, (Los Alamitos, California), pp. 745–748, Conference Publishing Services, IEEE Computer Society, 2015.

[39] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: a structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[40] W. McKinney, *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, Inc, 2012.

[41] F. Chollet *et al.*, "Keras," 2015.

[42] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[43] J. J. Downs and E. F. Vogel, "A plant-wide industrial process control problem," *Computers & chemical engineering*, vol. 17, no. 3, pp. 245–255, 1993.

[44] A. Kulkarni, V. K. Jayaraman, and B. D. Kulkarni, "Knowledge incorporated support vector machines to detect faults in tennessee eastman process," *Computers & chemical engineering*, vol. 29, no. 10, pp. 2128–2133, 2005.

[45] L. H. Chiang, E. L. Russell, and R. D. Braatz, *Fault Detection and Diagnosis in Industrial Systems*. Advanced Textbooks in Control and Signal Processing, London: Springer London, 2001.

[46] T. J. Rato and M. S. Reis, "Fault detection in the tennessee eastman benchmark process using dynamic principal components analysis based on decorrelated residuals (dpca-dr)," *Chemometrics and Intelligent Laboratory Systems*, vol. 125, pp. 101–108, 2013.

[47] E. L. Russell, L. H. Chiang, and R. D. Braatz, "Fault detection in industrial processes using canonical variate analysis and dynamic principal component analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 51, no. 1, pp. 81–93, 2000.

[48] G. S. Chadha, M. Krishnamoorthy, and A. Schwung, "Time series based fault detection in industrial processes using convolutional neural networks," in *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1, pp. 173–178, 2019.

II. Deep Convolutional Clustering-Based Time Series Anomaly Detection

Outline

Bibliographic Information

Gavneet Singh Chadha, Intekhab Islam, Andreas Schwung, Steven X. Ding (2021, August): Deep Convolutional Clustering-Based Time Series Anomaly Detection. In Sensors 21 (16):5488 doi: 10.3390/s21165488.

Author's contribution

The author contributed in the conceptualization, methodology, software development, validation, formal analysis, results investigation, writing the original draft, writing the review and editing and making visualisations.

Copyright Notice

II.1 Abstract

This paper presents a novel approach for anomaly detection in industrial processes. The system solely relies on unlabeled data and employs a 1D-convolutional neural network-based deep autoencoder architecture. As a core novelty, we split the autoencoder latent space in discriminative and reconstructive latent features and introduce an auxiliary loss based on k-means clustering for the discriminatory latent variables. We employ a Top-K clustering objective for separating the latent space, selecting the most discriminative features from the latent space. We use the approach to the benchmark Tennessee Eastman data set to prove its applicability. We provide different ablation studies and analyze the method concerning various downstream tasks, including anomaly detection, binary and multi-class classification. The obtained results show the potential of the approach to improve downstream tasks compared to standard autoencoder architectures.

II.2 Introduction

Sophisticated and interconnected modern manufacturing systems require transparent and insightful analytics. Consequently, intelligent condition monitoring of such processes is necessary to analyze changes in the process parameters and determine anomalies that hurt the reliability of the overall system. This unreliability can also lead to substantial financial consequences. However, modern production systems constitute complex interconnected behaviour, which renders the derivation of models through the first principle very difficult [1]. Hence, data-driven methods are an appealing alternative, particularly as a huge amount of data ranging from field level devices like sensors and actuators to manufacturing execution systems and enterprise resource planning systems are available through the Industrial Internet of Things [2].

However, a significant part of data-driven methods, namely supervised machine learning relies on the availability of labelled data from all of the possible operating conditions of the system. This availability of labelled data for industrial processes is infeasible due to various reasons. First, the faulty or abnormal operation often results in shutdowns or instantaneous repair actions, such that sufficient data instances are lacking. Second, data set labelling has to be done manually, which is usually not accomplished in industrial practice. Third, data sets for inconceivable fault cases are impossible to gather. In such cases, unsupervised or semi-supervised learning based data-driven techniques is the only alternative as they can suitably characterize the fault-free state of the system, which can subsequently be used to assess abnormal or faulty conditions.

Unsupervised or semi-supervised methods have been aggressively used in the area of novelty or anomaly detection. As surveyed in [3], the methods for anomaly detection can be categorized into Probabilistic Models, Distance-based Models, Reconstruction Models, One-Class Classification Models and Information-theoretic Models. The methods for anomaly detection can be further categorized into shallow and deep learning methods as surveyed in [4]. Recently, deep neural networks (DNNs) have shown a great capability to extract meaningful patterns from raw data with multiple levels of abstraction, providing state of the art results in various application fields like image recognition, object detection, speech recognition and natural language processing [5]. For unsupervised learning, approaches based on the Autoencoder (AE) framework [6] and Generative Adversarial

Networks (GANs) [7] have proven helpful for anomaly detection. GANs are trained by employing a minimax game where a discriminator is trained to distinguish between real and fake data generated by a generator network. However, the training objective resulting in a saddle point convergence renders GANs notoriously hard to train. The AE framework encodes the multivariate sensor signal into a latent variable space by means of a DNN from which a decoder network reconstructs the input. AE architectures can be distinguished based on the form of input data corruption, and latent variable sampling they possess, namely Denoising AE [8], Variational AE (VAE) [9] and Adversarial AE [10]. In all approaches, the latent variable space constitutes an abstract representation of the input signals, which can infer between normal and abnormal conditions. However, as the training objective of the AE is the reconstruction loss between input and output, the discriminative power of latent variables to distinguish between operation modes is not enforced, which can result in poor performance in anomaly detection.

This paper tackles this problem and proposes a novel approach for anomaly detection in industrial processes based on a clustering-loss augmented convolutional autoencoder (CAE). We use a 1-dimensional CAE as the backbone architecture for the multivariate time-series task. In contrast to existing approaches, we split the latent space of the CAE into two sets, namely discriminative and reconstructive latent variables, and add an auxiliary loss for the discriminative latent variables. The loss is defined in terms of the well known K-means [11, 12] clustering loss, where the auxiliary loss from the K-means algorithm during training is sampled only for the Top-K latent variables based on the greatest cluster centre distance achieved in clustering space. The reconstruction and the auxiliary loss are propagated through the discriminative latent variables, allowing for more discriminative hidden representation. We provide thorough experiments with unsupervised and semi-supervised approaches on the Tennessee Eastman [13] benchmark data set for anomaly detection. The results underline the applicability of the approach resulting in state-of-the-art performance.

The contributions of the paper can be summarized as follows:

- We present a novel unsupervised learning approach based on 1-dimensional convolutional neural networks and deep autoencoder structure where we define an auxiliary loss to increase the expressiveness of the latent representation.
- The proposed Top-K Deep Convolutional Clustering algorithm (Top-K DCCA) is novel in that the encoder parameters are divided into clustering and reconstruction subsets with the help of the Top-K operator. After this division, the encoder parameters from the clustering part are updated with an auxiliary clustering loss.
- We experiment with pure unsupervised and semi-supervised learning evaluation of the proposed method and report remarkable improvement on the Tennessee Eastman benchmark data set for anomaly detection. The results show the superior performance of the approach compared to the state-of-the-art.

The paper is organized as follows. In Sec. II.3 the related work is presented. In Sec. II.4 we state the considered problem, followed by the theoretical background on Clustering and Convolutional AE in Sec. II.5. Sec. II.6 presents the proposed approach for Convolutional Clustering-based unsupervised anomaly detection. In Sec. II.7 we provide results and comparisons on the well-known Tennessee Eastman benchmark dataset. Sec. II.8 concludes the paper.

II.3 Related Work

We discuss related work on data-based condition monitoring and anomaly detection in multivariate time series data and unsupervised and self-supervised learning approaches. Anomaly detection has been researched in various application fields and datasets. Some examples of datasets include the ISCX dataset [14] dataset for network intrusion detection, credit card fraud detection from the Mellon Bank Fraud Detection Feasibility Study [15] and health deterioration detection from the Oxford Cancer Hospital dataset [16]. The Tennessee Eastman dataset is chosen because the focus of this study is data-based condition monitoring for industrial processes.

Data Based Condition Monitoring and Anomaly Detection: Condition monitoring and anomaly detection have a long history in various application domains. Anomaly detection for a production process can be seen as a sub-category in the condition monitoring field. In general, we can distinguish the existing works for condition monitoring in shallow learning and deep learning approaches. The various shallow learning approaches have been surveyed in [17, 18, 19]. Some examples of shallow methods for anomaly detection with unsupervised learning include Kernel Density Estimation [20], Principal Component Analysis [21], k nearest neighbours [22] and One-Class Support Vector Machines [23]. However, most of the mentioned shallow approaches are static, such that they cannot be efficiently used for time-series anomaly detection tasks. Additionally, extraction of relevant features from multivariate raw data is still a challenge with shallow methods. Deep domain knowledge of the process is required for choosing suitable techniques for feature extraction in the shallow approaches.

Different approaches from the deep learning field appear as high performing and efficient algorithms for condition monitoring and time series analysis. The deep learning architectures use multiple layers of non-linear transformations to extract high-level features from raw data, which provide relevant information for the respective task. The various deep learning approaches for condition monitoring have been surveyed in [24, 25, 26, 27]. Most deep learning approaches consider supervised learning problems where faulty operation modes or process anomalies are labelled. However, the assumption of labelled data sets in industrial applications is too restrictive in various applications, including condition monitoring, remaining useful lifetime estimation and tool wear detection, to name a few. Hence, recent approaches also consider unsupervised deep learning approaches for anomaly detection and condition monitoring. Notably, deep AE and GANs have shown to be of particular use for such applications.

Deep Semi-supervised and Unsupervised learning: Deep learning models for anomaly detection have been used in various domains such as Intrusion Detection, Fraud detection, Malware detection, Medical detection [28]. We will highlight some specific examples from GANs and AE here. Anomaly detection for imaging markers relevant for disease progression with unsupervised learning based GANs has been reported in [29]. A semi-supervised learning based GAN has been presented for anomaly detection in multiple image datasets in [30]. Recently, there have been studies that use GAN for unsupervised fault diagnosis in rolling bearing [31] and semi-supervised fault diagnosis in planetary gearbox in [32]. Although recent improvements have been made in the GAN architecture, GANs are still known to have unstable training progress [33].

Deep AE, on the other hand, started the deep learning era in [34] and have been widely

tested in various domains of anomaly detection such as brain scans [35], outlier detection in videos [36] and multiple public datasets from the UCI machine learning repository [37]. Recently, an automatic thermography defects detection using a spatial and temporal segmentation model has been proposed in [38]. A sparse mixture of Gaussian decomposition algorithm for inductive thermography has been proposed in [39]. Although deep AE for anomaly detection can be used in a supervised setting [40], we will focus on the methods for unsupervised and semi-supervised settings in production processes. An unsupervised learning based, memory augmented AE architecture has been proposed in [41] to better identify anomalies from normal data. A deep support vector data description method inspired by kernel-based one-class classification method for anomaly detection has been proposed in [42]. Stacked Sparse AE in a semi-supervised setting has been proposed in [43] for fault diagnosis in rotating machinery such as gearboxes. A similar semi-supervised learning approach for induction motor fault detection has been proposed in [44]. Unsupervised learning-based wind turbine monitoring with deep AE has been proposed in [45, 46]. Unsupervised learning based spatiotemporal feature extraction methodology using Restricted Boltzmann Machines for fault detection has been proposed in [47]. Unsupervised Process monitoring with the variant AE has been presented in [48]. A comparison of deep AE, deep Denoising AE and VAE for semi-supervised anomaly detection approach in the TE process has been proposed in [49]. However, all of the previous methods are static approaches, which do not consider the dynamic nature of time-series data.

For time-series based anomaly detection, a Long Short Term Memory (LSTM) based encoder-decoder architecture has been proposed in [50]. Convolutional AE (CAE) was first presented in [51] for a higher level of feature extraction in images. CAE has, after that, been used for anomaly detection in images [52] and videos [53]. The Attention augmented Convolutional LSTM model has been proposed in [54] for anomaly detection in multivariate time series data. However, none of these approaches enhances the discriminative ability of the latent representation of the CAE model.

Deep Clustering: Some approaches in the literature join the use of feature extraction and clustering together to have better discriminative features. [55] proposed a joint clustering and reconstruction approach for image and text data. The main idea is to connect a clustering module at the bottleneck layer of an AE and optimize the parameters of the AE and the cluster centres jointly. A similar approach with CAE and clustering has been proposed in [56] for image data. Deep clustering has been also used for learning the weights of a convolutional network by using the cluster assignments as supervision [57]. Apart from K-means, an approach with KL-divergence minimization has been proposed in [58].

Our approach differentiates from the previous methods in two ways. First, we propose a Top-Kclustering approach where the latent space is divided into clustering friendly and reconstruction friendly spaces. Therefore, the latent features for reconstruction only get a gradient from the reconstruction error. However, the clustering features receive the update gradient from reconstruction and the clustering errors. Secondly, we apply the proposed approach on a multivariate time-series dataset from an industrial benchmark for anomaly detection. Therefore, the application field is very different from the usual image datasets.

II.4 Problem Statement

The main challenge for anomaly detection is to distinguish anomalous behaviour from data set noise. We conjecture that an incipient anomaly cannot be detected by one instance of the data set; instead, a specific time window of the input data set is required. Therefore, we concentrate on the analysis of multivariate time series data, i.e., we consider a sequence $\{x_1, x_2, \ldots, x_T\}$ where $x_i \in \mathbb{R}^m$ as input for the anomaly detection task, with m denoting the number of variables and T the length of the time-series signal. Further, we consider a hybrid, reconstruction-clustering based unsupervised learning methodology for anomaly detection, i.e., we assume that the evaluated data set is unlabeled. No indication is available whether the sequence exhibits normal or abnormal behaviour. Note, however, that for semi-supervised evaluation of the proposed approach, we use the learned AE for anomaly detection; labelled data is partly required and assumed to be known.

Then we can state the considered problem as follows: The purpose of the approach is to train the CAE structure $f_\theta(\mathbf{x})$, in such a way that the learned latent representation z, is able to best discriminate between normal z_{no} and anomalous behavior z_{ano}, i.e., $|z_{no} - z_{ano}| \rightarrow \max$. Particularly, we aim to find an optimal separation between normal and anomalous data using unlabelled data only.

We present the solution that combines a deep CAE architecture with a latent representation clustering algorithm to find better discriminative latent representations.

II.5 Theoretical Background

II.5.1 K-Means Clustering

Clustering is one of the most profound and fundamental tasks in the field of unsupervised learning. However, various sets of factors make clustering notoriously complex. Some of these factors include [59]

- amount of noise in the data which can occur during data acquisition,
- use of data pre-processing techniques such as any form of dimensionality reduction
- the clustering criterion and optimization algorithm is chosen
- the initialization of the cluster centres

These factors can affect the outcome of the clustering algorithm and can produce trivial solutions.

We keep the focus of our study to the K-means [11] algorithm. K-means, like most other data clustering algorithms, partitions the data into a pre-specified number of clusters. Clustering algorithms achieve this by minimizing a well-defined cost function involving the data and the assignment of the centres for each data instance. K-means belongs to the hard type, where each data point belongs to only one partition.

Formally, the task of clustering is to group N data samples into K clusters given a set of data samples $\{x_i\}_{i=1,\ldots,N}$ where $x_i \in \mathbb{R}^{\mathbb{M}}$. The *K-Means clustering* algorithm achieves

this goal by the optimization of the following cost function:

$$\underset{M \in \mathbb{R}^{\mathbb{M} \times \mathbb{K}}, \{s_i\} \in \mathbb{R}^{\mathbb{K}}}{\text{minimize}} \sum_{i=1}^{N} ||x_i - M s_i||_2^2 \tag{II.1}$$

$$s.t.\, s_{j,i} \in \{0, 1\},\, 1^T s_i = 1 \,\forall i, j$$

where s_i is the assignment vector of the ith data instance which consists of only one non-zero element, $s_{j,i}$ stands for the jth element of s_i, and the kth column of $\boldsymbol{M}$ stands for the centroid of the kth cluster.

The efficiency of the *K-Means* algorithm is the most when the data samples are evenly scattered around their centroids in their feature space. The data sets which possess this characteristic are called *K-Means* friendly data sets. However, this phenomenon rarely holds up in real-world data sets, because most of the real-world data sets are very high dimensional. Adding to that, most of the real-world data sets contain unwanted noise in the data. All these factors hinder the possibility of a data set being *K-Means* friendly [55].

To avoid these issues, usually, some form of dimensionality reduction or non-linear representation technique is used on the data set before applying *K-Means.* The *K-Means* algorithm applied to this non-linear representation usually yields better results [60]. The several available dimensionality reductions or non-linear representation techniques use *Deep Neural Networks* to learn better features from the data set. These methods are widely used for data pre-processing before applying *K-Means*, or other clustering algorithms [61].

II.5.2 1-D CNN Autoencoder

The proposed encoder-decoder network architecture for the Top-K DCCA is shown in Fig. II.1, in which the encoder consists of 3 convolution layers, and the decoder comprises 3 deconvolution layers. Addtionally, there is a clustering module on the bottleneck representation of the encoder. The autoencoder applies a stack of 1-dimensional convolutional layers at both encoder and deconvolution layers at the decoder. The encoder transforms the multivariate time series data set to a latent representation thereby extracting relevant features of the data set. The decoder subsequently reconstructs the original data set from the general low dimensional latent representation. Since the decoder reconstructs the input based on the encoded representation of the bottleneck layer, i.e. Conv 3 layer, the activation maps from the Conv 3 layer can be considered as an encoded representation for a batch of the input dataset. Therefore, it is clear that the encoded representation has a verifiable relationship to the input features since the decoder recreates the input features from the activation maps in the encoded representation. The input size of each of the layers follows the naming convention as $(Batch - Size \times Number - of - Input - Channels \times Sequence - Length)$.

On top of the latent representation, we employ a clustering module to make the latent representation more discriminative, allowing us better to capture the differences between normal and anomalous behaviour. As shown, we only employ the clustering on a subset of latent representations chosen based on different criteria to be discussed below. The rationale behind that architectural choice is to find a trade-off between consistent latent representations resulting in good reconstruction accuracy while making a subset of latent

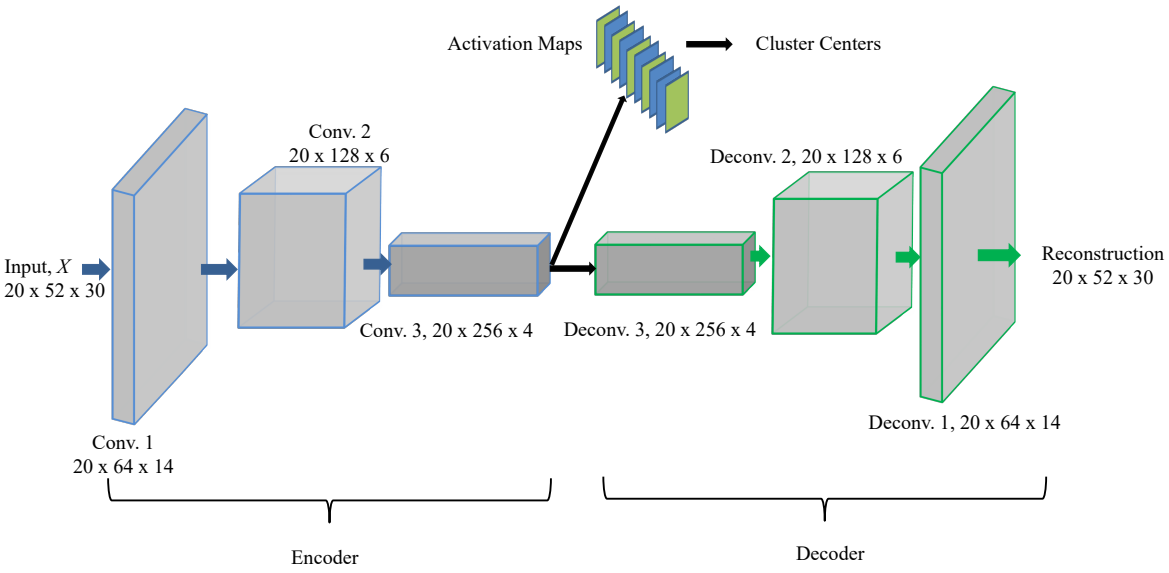


Figure II.1: Proposed architecture of the clustering augmented deep autoencoder for anomaly detection.

representation more discriminative, which suits downstream processing. In the following, we discuss the architectural modules in detail.

The combination of autoencoder structures with CNNs is a standard approach for deep unsupervised learning in various image and video processing tasks [53]. Here, at the encoder and decoder, convolutional and deconvolutional layers are employed to extract essential information within the latent representation. We use a similar approach to the time series analysis as proposed in [62], where the sensor channel and time dimensions make up the input to the network. As mentioned in the study, applying the standard 2-dimensional kernel is not appropriate as a meaningful relation between sensor channels is missing, resulting in poor performance. The 1D convolution operation is performed over a part of the complete input space, which is referred to as the receptive field. We denote the receptive field of size $n_r \times m$, which strides over the input $T \times m$ sequences, accounting for each of the variables. The pth convolution 1D kernel in the first layer can be denoted with a 2-dimensional tensor $K^{(p)} = [k_{i,j}^{(p)}] \in \mathbb{R}^{n_r \times m}$. The indices i, j denote the dimension along the time and variable axis, respectively. The outputs or feature maps extracted from the convolution operation with 1 convolution kernel is a 1-dimensional tensor $H = [h_i]$. Usually, multiple convolution kernels are used in each convolution layer leading to multiple feature maps, which subsequently make the feature maps a 2-dimensional tensor $H = [h_{i,p}]$. Each convolution kernel is responsible for extracting different features from the input data. Formally, the convolution 1D operation can be

summarized as follows:

$$h_{i,p} = (x * k)_i = \sum_{g=1}^{n_r} \sum_{f=1}^{m} x_{i+g-1,f} \cdot k_{g,f}^{p}$$
$$\forall i \in \{1, \ldots, T - n_r + 1\}$$
$$\forall p \in \{1, \ldots, d_{q+1}\}, \tag{II.2}$$

where $h_{i,p}$ denotes the output of the $(i)^{th}$ receptive field and the pth convolution kernel, $x_{i+g-1,f}$ are the elements in the receptive field of the input variable, $k_{g,f}$ is the convolution kernel and d_{q+1} denotes the number of convolution kernels in the given layer.

The deconvolution, sometimes called the transposed convolution operation, performs the inverse operation as the convolution operation, such that it up-samples the individual feature maps into the original input. The weights of the convolution and deconvolution filters can be tied, but we keep them untied in this study.

As we cope with time series of variable length where the time dimension is significant, we employ a sliding window approach for the time dimension. As such, we define a window of size of $m_w \times n$ with $T >> m_w > m$, which is analyzed within one processing step of the deep autoencoder. Then the time series is strided in the time dimension by a stride of s_w to define a new window to be processed in the next step. This approach has some advantages compared to processing directly on the complete input sequence. Notably, an individual data point $\{x_i\}$ is processed more than once in different settings, increasing the robustness of the resulting convolution kernels.

II.6 Convolution Clustering Based Unsupervised Learning for Anomaly Detection

In this section, we propose the training strategy for the unsupervised learning approach for the Top-K DCCA approach.

II.6.1 Top-K DCCA

We augment the previously defined CAE architecture by a novel Top-Kclustering objective defined on a subset of the latent space as illustrated in Fig. II.1. Particularly, we split the latent space into two subsets of latent variables $\mathcal{Z}_c \subseteq \mathbb{R}^{n_c}$ and $\mathcal{Z}_r \in \mathbb{R}^{n_{rec}}$ which we term clustering and reconstruction friendly latent variables in the following. The rationale behind the split of the latent space is to better weigh-off between reconstruction accuracy and discriminative clustering accuracy. Hence, we force consistent representation of the input data by the reconstruction space and the discriminative power of the clustering features to improve performance on downstream tasks.

As such, the clustering related latent variables are passed through an arbitrary clustering algorithm. We employ the well-known K-means algorithm for clustering in this work due to its simplicity. However, we emphasize that various other clustering approaches can be combined with our framework. The k-means algorithm is subsequently used on the

latent representation $\mathcal{Z}$, leading to the optimization of the following cost function:

$$\min_{M_j \in \mathbb{R}^{n_c \times k}, s_i \in \{0,1\}^K} \sum_{i=1}^{N} ||z_j^i - M_j s_i|| \tag{II.3}$$

$$\text{s.t.} \quad \mathbf{1}^T s_i = 1 \; \forall i, \tag{II.4}$$

$$\forall z_j \in \mathcal{Z}, \tag{II.5}$$

where the column vector $m_{k,j}$ of M denotes the kth cluster center in the n_c-dimensional space and s_i is the cluster assignment of the ith data points latent representation.

A crucial part of the system setup is the split of the latent space. A straightforward approach would be to separate cluster and reconstruction friendly latent variables before training. However, this appears to be restrictive when used together with the CAE, particularly during training. Hence, instead of defining the split at the start of training, we augment the K-means clustering by a Top-K sampling method that uses the top-n_c latent variables in terms of their discriminative performance. The splitting criterion is the euclidean distance between the 2 cluster centers present in each of the latent variables. The Top-K operation of latent variables ranking returns indices of the K latent variables where the distance between the cluster centers is maximum. The discriminative performance is measured based on the euclidean distance between the cluster centres in the latent space. According to the authors, the maximum distance between the cluster signifies that the latent variable has more discriminative performance since it can efficiently identify the 2 different operating conditions. Specifically, if we assume an anomaly detection task with two clusters with centres $m_{no,j}$ and $m_{ano,j}$ indicating normal and anomalous operation, respectively, we employ the following euclidean distance measure

$$\max_{j \in \mathcal{Z}_c} d(m_{no,j}, m_{ano,j}) = \max_{j \in \mathcal{Z}_c} ||m_{no,j} - m_{ano,j}||^2, \tag{II.6}$$

to identify the Top-n_c latent variables forming the set $\mathcal{Z}_c$.

It is important to note that the clustering loss is employed independently on the latent variables in the set $\mathcal{Z}_c$. However, during training, we fed back the loss of the top-n_c latent variables only. Therefore, during training, the latent variables switch among the clustering subset and reconstruction subset, based on the euclidean distance of their respective cluster centers. This ensures that a subset of latent space is discriminative by forcing the model to learn a hidden representation in which certain cluster centers are as far away as possible based on the criterion from Eq. (II.6). During the testing phase, the trained division latent space into the 2 subsets is kept constant.

The split percentage of the latent variables defined by n_c, n_r is a hyperparameter that has to be determined a priori. It has to trade-off between reconstruction and discrimination capability of the latent variable space. In practice, we found a 50/50 split between working well in all the experiments.

II.6.2 End-to-end training of the clustering augmented AE

This section introduces the end-to-end training for the clustering augmented deep autoencoder. Particularly, we discuss the interaction between the loss propagation of the clustering and the reconstruction module of the autoencoder. The parameters of the CNN

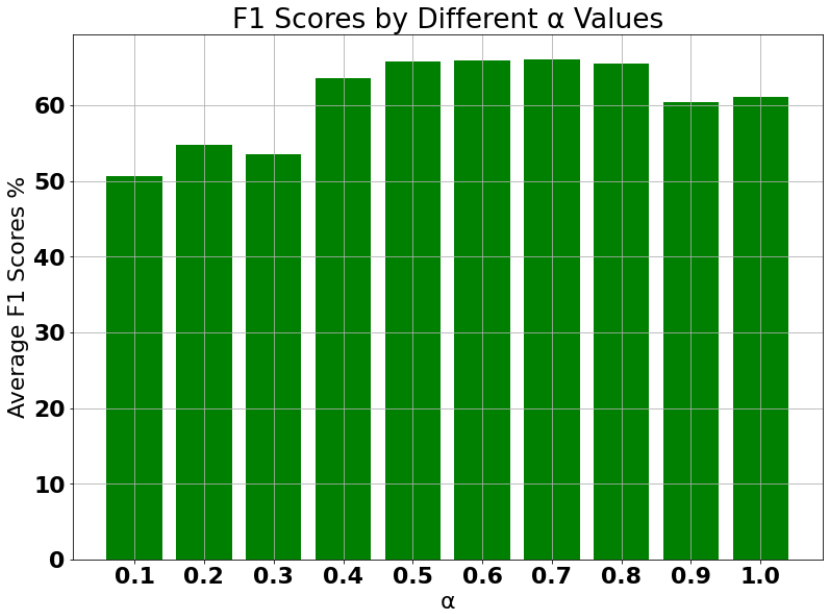


Figure II.2: Average F_1 score of the model on all the fault cases based on different values of α.

of both encoder f_θ and decoder g_ψ are trained by the reconstruction loss between input and reconstructed output, i.e.,

$$L_{\text{AE}}(\theta, \psi) = \sum_{i=1}^{N_B} ||x^i - g_\psi(f_\theta(x^i))||_2^2, \tag{II.7}$$

where N_B is the minibatch size. Additionally, we feed back the clustering loss through the clustering friendly latent variables

$$L_{j,\text{CL}}(\theta) = \sum_{i=1}^{N_B} ||z_j^i - M_j s^i||_2^2 = \sum_{i=1}^{N_B} ||f_{j,\theta}(x^i) - M_j s^i||_2^2, \tag{II.8}$$

$$z_j \in \mathcal{Z}_c, \tag{II.9}$$

which subsequently affect the encoder parameters only.

The total loss for training the CAE is

$$L = \alpha \sum_{j=1}^{z_j} L_{j,\text{CL}}(\theta) + (1 - \alpha) L_{\text{AE}}(\theta, \psi) \tag{II.10}$$

where the value of α ranges between 0.6 to 1, and it acts as a weighing factor between the two loss functions. This range of optimal value of α was empirically found based on the average F_1 score that was achieved on all the fault cases.

It is considered an additional hyperparameter of the network and has to be tuned while training it. Since $\alpha \leq 1$ keeps the overall loss distribution towards the reconstruction and clustering losses balanced.

It is theoretically possible to chose a a different independent parameter β, with the condition that $\alpha + \beta = 1$. But to keep the number of hyperparameters in check, this setting of just one hyperparameter α has been chosen.

The gradient of the above equation with respect to the network parameters can be computed from the equation below:

$$\nabla \chi \, L = (1 \, - \, \alpha) \, \frac{\partial L_{\text{AE}}}{\partial \chi} \, + \, \alpha \sum_{j=1}^{z_j} \frac{\partial L_{j,\text{CL}}}{\partial \theta} \tag{II.11}$$

$$\nabla \chi \, L = (1 \, - \, \alpha) \, \sum_{i=1}^{N_B} 2(x^i - g_\psi(f_\theta(x^i))[g_\psi(f_\theta(x^i))]' \, + \, \alpha \sum_{i=1}^{N_B} \sum_{j=1}^{z_j} 2(f_{j,\theta}(x^i) - M_j s^i) f_{j,\theta}(x^i)' \tag{II.12}$$

where $\chi = (\theta, \psi)$ is the collection of encoder and decoder parameters and the partial gradients are calculated by back-propagation [63]. Subsequently, the network parameters are updated with gradient descent as

$$\chi \leftarrow \chi \, - \, \beta \nabla \chi L \tag{II.13}$$

where β is the learning rate.

During the initial stages of training, termed as pre-training, the value of α is set to 0. This ensures that the network learns from only the reconstruction loss. Since no clustering loss is imposed on the network, the network tries to reconstruct the input solely based on the non-clustering loss. For the clustering augmented training stage, a fixed value of α is set. The network is trained on both loss functions. This method ensures that the reconstruction of the input is taken into account and helps to avoid trivial solutions. Also, we define a *Cluster Update Interval* C, which denotes the interval in which the cluster centres of the latent feature representation are updated to have robust hidden representation.

The algorithm of the Top-K DCCA is represented in Algorithm 3, where a model is trained for N epochs.

Algorithm 3 Top-K Deep Convolutional Clustering Algorithm

procedure INITIALIZATION(Perform N epochs over the data)
    P = Number of pre-training epochs
    C = Cluster update interval
    for epoch = 1 to P + 1 **do**
        Reconstruct the data, extract latent representation $f_\theta(x^i)$
        Compute gradients $\nabla \chi L^i$ with $\alpha = 0$ by Eq. (II.11)
        Update network parameters χ by Eq. (II.13)
        if epoch = P + 1 **then**
            Perform *K-Means* optimising the Eq.(II.3)
            Return centers $m_{no,j}$ and $m_{ano,j}$ and center assignments $M_j s_i$
            Rank latent representation layer channels by Eq. (II.6)
            Return Top K ranked channels
    for epoch = P + 1 to N **do**
        Reconstruct the data, extract latent representation $f_\theta(x^i)$
        Compute gradients $\nabla \chi L^i$ with $\alpha = 0$ by Eq. (II.11)
        Update top K ranked channel parameters by Eq. (II.13)
        Zero the gradients
        Compute gradients $\nabla \chi L^i$ with $\alpha = 0$ by Eq. (II.11)
        Update rest of the channel parameters by Eq. (II.13)
        if epoch % C = 0 **then**
            Perform *K-Means* by optimising the Eq.(II.3)
            Return centers $m_{no,j}$ and $m_{ano,j}$ and center assignments $M_j s_i$
            Rank latent representation layer channels by Eq. (II.6)
            Return Top K ranked channels

II.7 Experimental Results

II.7.1 Tennesse Eastman Benchmark

The TE process was originally created by Downs and Vogel as a process control challenge problem in [13]. The generated dataset from the TE Process consists of 22 continuous process measurements, 19 component analysis measurements, and 12 manipulated variables. The dataset consists of 21 pre-programmed faults, among which 16 are known fault cases, and 5 fault cases are unknown. Both the training and testing datasets include a total of 52 observed variables. The training dataset consists of 22 different simulation runs, and simulation 0 is fault-free. In our case, this simulation is considered as our normal data sample. Simulations 1 to 21 were generated for 21 fault cases, and in our case, all of these 21 simulations are considered anomalous data samples. Similarly, the testing data set contains 22 different simulations, the first one being the normal case, and the rest are simulations for different fault cases. Table II.1 represents the Tennessee Eastman Process fault cases. Since the TE process dataset contains collected time-series sensor data, the data is prepared as time series sequences as discussed in [1] before the training.

Fault Cases	Description	Type
1	A/C ratio, B composition constant (Stream 4)	Step
2	B composition, A/C ratio constant (Stream 4)	Step
3	D feed temperature (Stream 2)	Step
4	Reactor cooling water supply temperature	Step
5	Condenser cooling water supply temperature	Step
6	A feed loss (Stream 1)	Step
7	C header pressure loss (Stream 4)	Step
8	A, B, C feed composition (Stream 4)	Random
9	D feed temperature (Stream 2)	Random
10	C feed temperature (Stream 4)	Random
11	Reactor cooling water supply temperature	Random
12	Condenser cooling water supply temperature	Random
13	Reaction Kinetics	Slow drift
14	Reactor cooling water valve	Sticking
15	Condenser cooling water valve	Sticking
16	Unknown	-
17	Unknown	-
18	Unknown	-
19	Unknown	-
20	Unknown	-
21	A, B, C feed valve (Stream 4)	Constant position

Table II.1: Tennessee Eastman Process Fault Cases

Subgroup	Normal Case	Fault Cases
Easy	0	1,2,4,5,6,7,12,14,18
Medium	0	8,10,11,13,16,17,19,20
Hard	0	3,9,15,21

Table II.2: Fault Groups in TE Process [1].

II.7.2 Training Setup

The length of each sequence is decided prior to the training, and both the data with and without faults are arranged into time-series sequences. This kind of arrangement has proved to help the model in increasing the performance since a time-series gives more context about the situation than a single measurement. We select a sequence length of 30 for our experiments as this length gives a good overall performance.

To define the anomaly detection setting, we follow previous works [1] by dividing the fault classes into subgroups based on how challenging the faults are to detect. Accordingly, we divide the 21 faults into three subgroups: easy, medium, and hard-to-detect faults. The three fault subgroups considered are as shown in Table II.2. The data from the literature have been adapted accordingly for comparison.

For evaluation of the anomaly detection task, we concentrate on measures related to the numbers of correctly and incorrectly classified data points. Specifically, we use the standard notions of true positives (TP) and true negatives (TN) to denote the number of examples predicted correctly as a positive and negative class, respectively and false positives (FP) and false negatives (FN) as the number of examples predicted incorrectly as a positive and negative class, respectively. Based on the values, we use the F_1 score as the performance measure. The F_1 score is chosen as the evaluation metric because if the number of examples in one of the classes is higher than the other, then even random guessing can result in high prediction accuracy. Therefore, we use the F_1 score, which is a geometric mean of precision P and recall R, is considered in the case of the TE process given as

$$F_1 = 2\frac{P \cdot R}{P + R}, \tag{II.14}$$

where

$$P = \frac{TP}{TP + FP}, \tag{II.15}$$

$$R = \frac{TP}{TP + FN}. \tag{II.16}$$

We apply the proposed learning methodology to the TE benchmark data set and provide a thorough ablation study. The comparison study is enlisted as follows.

- We start by comparing the fault detection capabilities for completely unsupervised learning techniques in which the proposed methodology is compared to the standard k-means augmented CNN approach.

- We then evaluate the fault detection capabilities with semi-supervised learning techniques, in which the proposed methodology is pre-trained with unlabelled data and finally, a fully connected layer is fine-tuned with labelled data. This technique is compared with and without K-means clustering, with and without Top-K K-means clustering.

In this section, we defined the training setup for the anomaly detection task on the TE process. Based on this setup, experimental results and ablation studies were performed to evaluate the prediction performance of the proposed methodology.

II.7.3 Unsupervised Learning Results

This section presents the results obtained by applying the proposed approach Top-K DCCA in a purely unsupervised learning setting. This means that no labels from the fault information have been used for training the models. The results obtained from the proposed approach are compared with the baseline architecture, hereafter referred to as the Vanilla architecture, and a standard DCCA approach. The Vanilla architecture is a 3 convolution layer architecture, whereas the Top-K DCCA model is tested with a 2 and 3 layer convolution layer architecture. The architecture description for the Vanilla, DCCA and the Top-K DCCA architecture is as follows:

- Three convolution layers with the LeakyReLU [64] activation function
- A kernel size of 3 in all convolution layers
- The number of convolution channels doubling with each layer, starting with 64 channels.
- The number of clustering channels is set to 128 in the bottleneck layer.
- A batch-size of 20 with $\alpha = 0.6$ and $\beta = 0.001$ is used.
- All the models are trained for 100 epochs with the stochastic gradient descent (SGD) optimizer with an $L2$ penalty of 0.02.

Anomaly detection in the Vanilla architecture is obtained by performing K-means clustering once after the training process, whereas in the other two architectures, K-means clustering is part of the training process.

To evaluate the prediction performance of the proposed architecture, a 2 and 3 layer Top-K DCCA architecture is compared to the Vanilla model for the anomaly detection task in the TE process. The prediction performance in terms of F_1 score for the best performing architectures is shown in Fig. II.3. It is clear from Fig. II.3 that the proposed architecture performs drastically better than the baseline model on all the fault categories in the 2 layer and the 3 layer configuration. The 3 layer configuration performs slightly better than the 2 layer one in all the cases. Therefore, for the subsequent analysis, we keep the 3 layer configuration.

To better visualize the discriminative capability in the latent representation, the t-SNE [65] plots of some of the clustering friendly activation maps are shown in Fig. II.4. In all of these t-SNE visualizations of the activation maps, the model has learned through

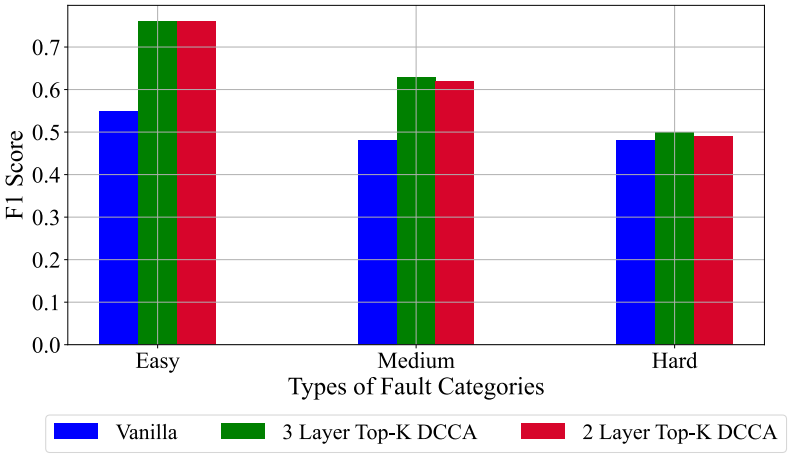


Figure II.3: F_1 score obtained by the Vanilla and the Top-K DCCA approach with different layers for anomaly detection task in an unsupervised learning setup

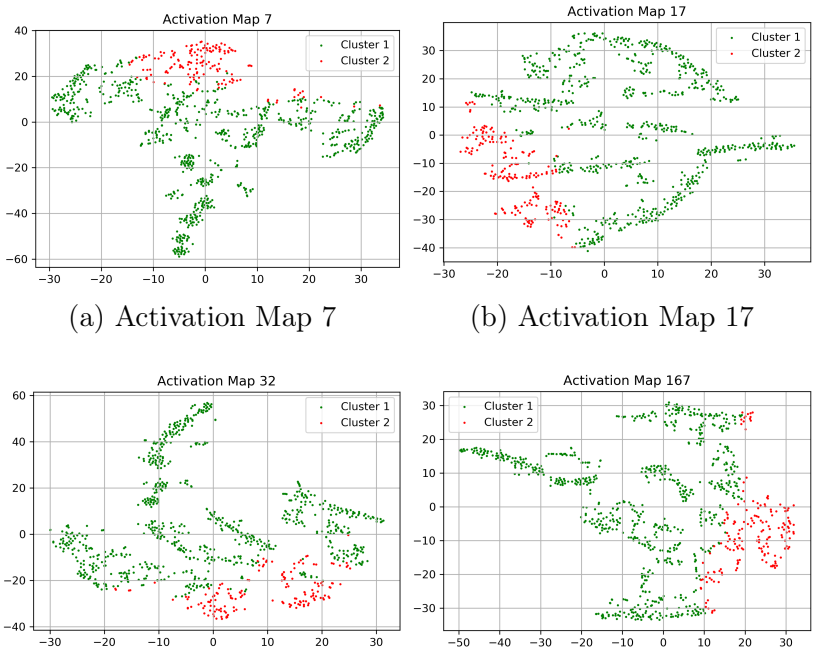
the training process that there are two distinct regions, i.e., normal and anomalous regions. The Fig. II.4 show two clusters because the Tennessee Eastman process dataset consists of either normal operation or faulty operation. That is why we limit the number of clusters to just two. The boundaries of the two distinct regions can be clearly seen, which demonstrates that the clustering operation has helped create these decision boundaries. The t-SNE visualizations show the distinct separation for most of the test samples. Some of the data samples from the two operating conditions are close to each other, signifying the hard to detect anomaly samples.

The unsupervised training results and the corresponding t-SNE plots prove the applicability of the proposed methodology to effectively identify anomalies in a dynamic and high-dimensional time-series process. A 3 layer unsupervised learning based Top-K DCCA approach performs the best under the considered experimental settings.

II.7.4 Semi-supervised Learning Results

In this section, we present the results from the semi-supervised training setup where the encoder of the Top-K DCCA architecture is pre-trained with unlabelled data as per Algorithm 3, with two fully connected layers with 300 and 2 hidden units being trained in a supervised manner with labelled data. The overall proposed architecture for semi-supervised learning is shown in Fig. II.5. The convolutional encoder is pre-trained using unlabelled data and the fully connected layers are fine-tuned using labelled data. During the fine-tuning stage, the weights and biases of the convolutional encoder are frozen.

The average F_1 score obtained by the Vanilla, DCCA and Top-K DCCA approach on the different fault categories is shown in Fig. II.6. It is clear from Fig. II.6 that the proposed Top-K DCCA approach outperforms the other two models in the Easy and Hard fault categories drastically. The standard DCCA only marginally performs better in the medium category; however, the proposed methodology works better than the Vanilla model in all three fault categories. To better estimate the anomaly detection performance of the model, confusion matrices for a sample of fault cases from the Easy, Medium and Hard fault groups have been illustrated in Fig. II.7. The confusion matrix from all the



(a) Activation Map 7 (b) Activation Map 17

(c) Activation Map 32 (d) Activation Map 167

Figure II.4: t-SNE Visualization of a sample of the activation maps with Top-K DCCA Approach on Tennessee Eastman Data

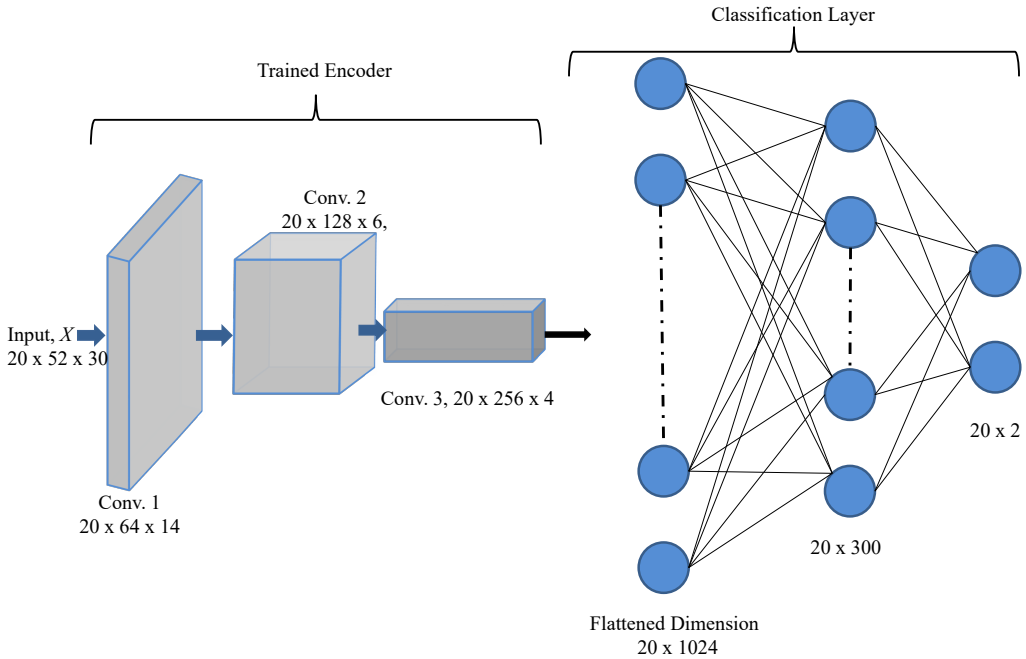


Figure II.5: Proposed architecture for semi-supervised deep autoencoder for anomaly detection.

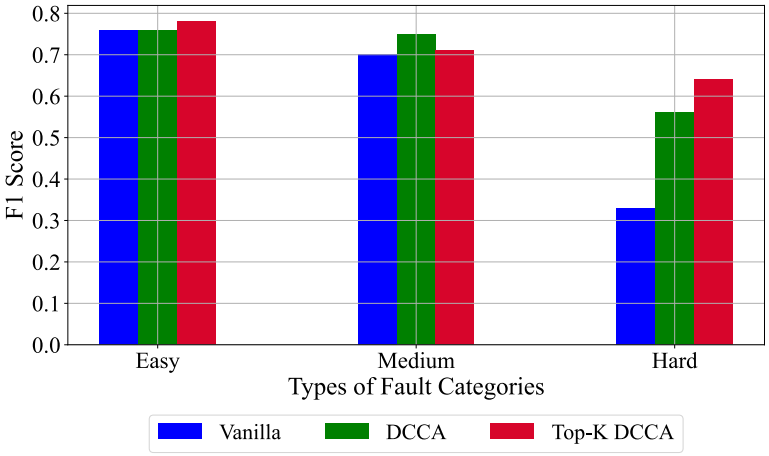


Figure II.6: F_1 score obtained by the Vanilla, DCCA and the Top-K DCCA approach for the anomaly detection task in a semi-supervised learning setup

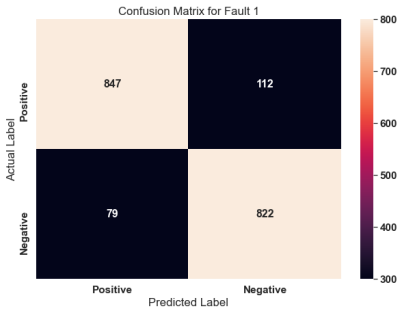
fault cases has not been added for the brevity of results. The confusion matrix for fault 1 and fault 2 shows that the model can distinguish the normal and faulty cases in most cases. However, the model has difficulty distinguishing some medium and hard fault cases from the normal case. This can be observed from the low performance on fault cases 3,9, and 10. It must be noted here that semi-supervised learning results are comparatively better than the unsupervised learning results since labelled data is used to train the final hidden layers.

II.7.5 Classification variants Results

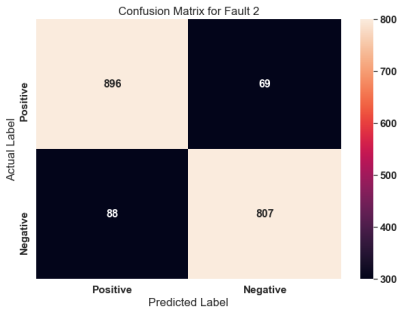
In this section, we present the results for the different classification variants that are possible with the proposed Top-K DCCA approach based on the semi-supervised learning approach. The classification variants include feeding only the clustering channels $\mathcal{Z}_c$ as input, reconstruction channels $\mathcal{Z}_r$ as input or both the sets together to the two fully connected layers. The architecture for the classification remains the same as in Fig. II.5. These different classification variants are done to observe how much each of the latent variables sets help in the final anomaly detection task. The average F_1 scores obtained by the three classification variants on the different fault categories is shown in Fig. II.8. It is clear from Fig. II.8 that the clustering set of latent variables $\mathcal{Z}_c$ as input performs consistently better than the reconstruction set $\mathcal{Z}_r$ as an input across all the different fault categories. This result emphasizes the importance of the Top-K clustering channels in the anomaly detection task. It must be noted, however, that using both the sets as input to the fully connected layers also drastically helps in improving the performance in the case of Medium and Hard fault cases.

II.7.6 Comparison with Literature

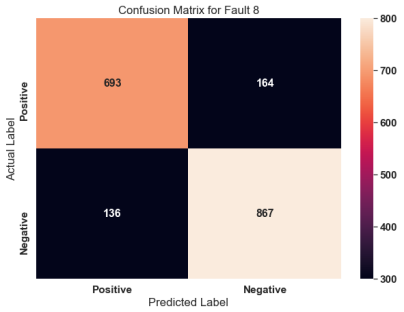
In this section, we provide a comparison of the anomaly detection performance of the proposed Top-K DCCA model with other existing approaches. We emphasize the performance of the hard to detect fault cases since having a good performing model on these



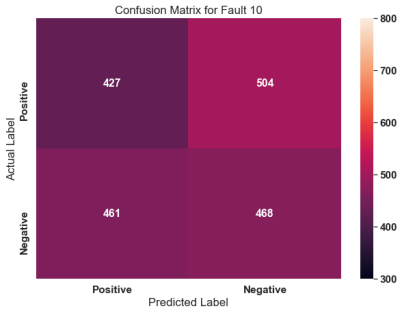
(a) Confusion Matrix for Fault 1



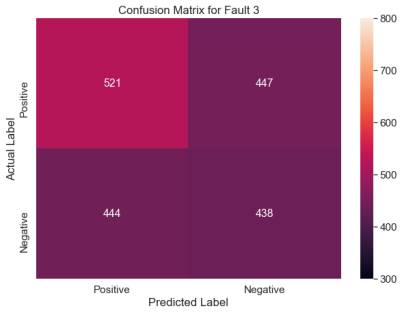
(b) Confusion Matrix for Fault 2



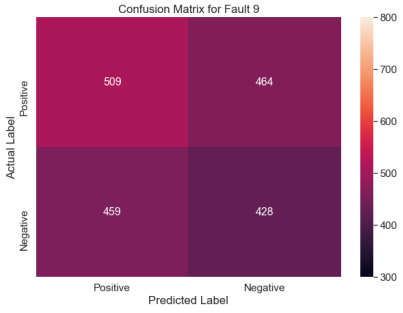
(c) Confusion Matrix for Fault 8



(d) Confusion Matrix for Fault 10



(e) Confusion Matrix for Fault 3



(f) Confusion Matrix for Fault 9

Figure II.7: Confusion matrix for a sample of fault cases from the Easy, Medium and Hard fault groups. The positive class represents normal case and the negative class represents the respective fault case.

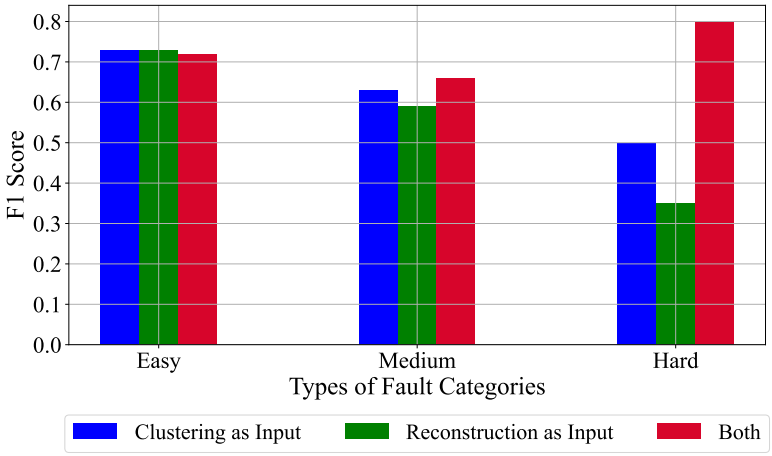


Figure II.8: F_1 score obtained by the different classification variants for the anomaly detection task in a semi-supervised learning setup

Table II.3: Comparison of the achieved F_1 scores for the hard to detect fault cases with existing approaches

Fault Case	**Top-K DCCA**	[21]	[66]	[67]	DAE [49]	Denoising DAE [49]
(3)	53.82	4.5	2.1	1.4	16.66	16.67
(9)	52.31	4.75	2	0.7	16.87	16.97
(15)	43.98	7.75	38.5	9.7	17.08	17.08
(21)	50.05	56.38	53.9	65.8	44.37	45
Overall	*50.04*	18.34	24.12	19.4	23.74	23.93

cases is a challenging task. Since most of the previous works use a percentage based evaluation metric, the F_1 score is multiplied by 100 to keep the comparison uniform. For the comparison, we selected the previous studies [21, 66, 67] and chose the best performing models Independent Component Analysis, Dynamic Principal Component Analysis with decorrelated residuals and canonical variate analysis, respectively. Furthermore, to compare the model's with other deep learning models, the Deep Autoencoder (DAE) and Denoising DAE have been selected from the previous work in [49]. The Table II.3 gives the comparison between the best performing unsupervised learning-based anomaly detection approaches with their achieved F_1 scores or fault detection rates as used in literature. The data from the literature have been adapted accordingly for comparison. The proposed Top-K DCCA model outperforms the existing literature methods in three out of the four fault cases and has a drastically better overall performance. In comparison to the other neural network approaches using fully connected layers, the proposed Top-K DCCA approach outperforms these methods on all hard to detect fault cases. The exceptional performance gain underlines the anomaly detection capability of the proposed model, especially in the case of incipient anomaly cases.

II.8 Conclusions

We presented a novel approach for unsupervised training of time series data sets with a particular focus on anomaly detection. The approach combines a deep 1D-CNN-based autoencoder with a clustering loss on a subset of the latent variable space, which increases the discriminative power within the latent variable space without sacrificing too much reconstruction performance on the data set. We make the approach end-to-end trainable by backpropagating both the clustering and the reconstruction objective through the network. We test the approach on the Tennessee Eastman benchmark data set with very encouraging results. In the unsupervised learning setting, a 3 layer proposed model drastically outperforms other deep Autoencoder networks and also shallow learning techniques proposed in the literature. The ablation studies in the semi-supervised learning setting show the superior performance of the model using the input from the clustering feature subset as compared to the reconstruction feature subset. This shows the discriminative power of the learnt features in the latent space.

In the future, authors would apply the proposed approach to other time-series datasets like Electric devices, Ford A and Ford B [68] to corroborate and confirm our findings.

Bibliography

[1] G. S. Chadha, A. Panambilly, A. Schwung, and S. X. Ding, "Bidirectional deep recurrent neural networks for process fault classification," *ISA transactions*, 2020.

[2] L. Da Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Transactions on industrial informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[3] M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko, "A review of novelty detection," *Signal Processing*, vol. 99, pp. 215–249, 2014.

[4] L. Ruff, J. R. Kauffmann, R. A. Vandermeulen, G. Montavon, W. Samek, M. Kloft, T. G. Dietterich, and K.-R. Müller, "A unifying review of deep and shallow anomaly detection," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 756–795, 2021.

[5] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[6] G. E. Hinton, "Connectionist learning procedures," *Artificial Intelligence*, vol. 40, no. 1-3, pp. 185–234, 1989.

[7] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, vol. 27, pp. 2672–2680, Curran Associates, Inc, 2014.

[8] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th International Conference on Machine learning*, pp. 1096–1103, 2008.

[9] D. P. Kingma, M. Welling, *et al.*, "An introduction to variational autoencoders," *Foundations and Trends in Machine Learning*, vol. 12, no. 4, pp. 307–392, 2019.

[10] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," *arXiv preprint arXiv:1511.05644*, 2015.

[11] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, pp. 281–297, 1967.

[12] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010.

[13] J. J. Downs and E. F. Vogel, "A plant-wide industrial process control problem," *Computers & chemical engineering*, vol. 17, no. 3, pp. 245–255, 1993.

[14] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, vol. 31, no. 3, pp. 357–374, 2012.

[15] Ghosh and Reilly, "Credit card fraud detection with a neural-network," in *1994 Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences*, vol. 3, pp. 621–630, 1994.

[16] L. Clifton, D. A. Clifton, P. J. Watkinson, and L. Tarassenko, "Identification of patient deterioration in vital-sign data using one-class support vector machines," in *2011 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pp. 125–131, 2011.

[17] V. Venkatasubramanian, R. Rengaswamy, S. N. Kavuri, and K. Yin, "A review of process fault detection and diagnosis: Part iii: Process history based methods," *Computers & chemical engineering*, vol. 27, no. 3, pp. 327–346, 2003.

[18] S. Yin, S. X. Ding, X. Xie, and H. Luo, "A review on basic data-driven approaches for industrial process monitoring," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 11, pp. 6418–6428, 2014.

[19] Z. Gao, C. Cecati, and S. X. Ding, "A survey of fault diagnosis and fault-tolerant techniques—part ii: Fault diagnosis with knowledge-based and hybrid/active approaches," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 6, pp. 3768–3774, 2015.

[20] A. Giantomassi, F. Ferracuti, S. Iarlori, G. Ippoliti, and S. Longhi, "Electric motor fault detection and diagnosis by kernel density estimation and kullback–leibler divergence based on stator current measurements," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 3, pp. 1770–1780, 2015.

[21] S. Yin, S. X. Ding, A. Haghani, H. Hao, and P. Zhang, "A comparison study of basic data-driven fault diagnosis and process monitoring methods on the benchmark tennessee eastman process," *Journal of Process Control*, vol. 22, no. 9, pp. 1567–1581, 2012.

[22] Q. P. He and J. Wang, "Fault detection using the k-nearest neighbor rule for semiconductor manufacturing processes," *IEEE Transactions on Semiconductor Manufacturing*, vol. 20, no. 4, pp. 345–354, 2007.

[23] S. Mahadevan and S. L. Shah, "Fault detection and diagnosis in process data using one-class support vector machines," *Journal of Process Control*, vol. 19, no. 10, pp. 1627–1639, 2009.

[24] S. Khan and T. Yairi, "A review on the application of deep learning in system health management," *Mechanical Systems and Signal Processing*, vol. 107, pp. 241–265, 2018.

[25] R. Zhao, R. Yan, Z. Chen, K. Mao, P. Wang, and R. X. Gao, "Deep learning and its applications to machine health monitoring," *Mechanical Systems and Signal Processing*, vol. 115, pp. 213–237, 2019.

[26] 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.

[27] Y. Lei, B. Yang, X. Jiang, F. Jia, N. Li, and A. K. Nandi, "Applications of machine learning to machine fault diagnosis: A review and roadmap," *Mechanical Systems and Signal Processing*, vol. 138, p. 106587, 2020.

[28] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," *arXiv preprint arXiv:1901.03407*, 2019.

[29] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, "Unsupervised anomaly detection with generative adversarial networks to guide marker discovery," in *Information processing in medical imaging* (M. Niethammer, ed.), vol. 10265 of *LNCS sublibrary: SL6 - Image processing, computer vision, pattern recognition, and graphics*, pp. 146–157, Cham: Springer, 2017.

[30] S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon, "Ganomaly: Semi-supervised anomaly detection via adversarial training," in *Computer vision – ACCV 2018* (C. V. Jawahar, H. Li, G. Mori, and K. Schindler, eds.), vol. 11363 of *LNCS sublibrary. SL 6, Image processing, computer vision, pattern recognition, and graphics*, pp. 622–637, Cham: Springer, 2019.

[31] H. Liu, J. Zhou, Y. Xu, Y. Zheng, X. Peng, and W. Jiang, "Unsupervised fault diagnosis of rolling bearings using a deep neural network based on generative adversarial networks," *Neurocomputing*, vol. 315, pp. 412–424, 2018.

[32] Z. Wang, J. Wang, and Y. Wang, "An intelligent diagnosis scheme based on generative adversarial learning deep neural networks and its application to planetary gearbox fault pattern recognition," *Neurocomputing*, vol. 310, pp. 213–222, 2018.

[33] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," *arXiv preprint arXiv:1701.04862*, 2017.

[34] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science (New York, N.Y.)*, vol. 313, no. 5786, pp. 504–507, 2006.

[35] N. Pawlowski, M. C. H. Lee, M. Rajchl, S. McDonagh, E. Ferrante, K. Kamnitsas, S. Cooke, S. Stevenson, A. Khetani, T. Newman, *et al.*, "Unsupervised lesion detection in brain ct using bayesian convolutional autoencoders," *International Conference on Medical Imaging with Deep Learning (MIDL)*, 2018.

[36] H. Yang, B. Wang, S. Lin, D. Wipf, M. Guo, and B. Guo, "Unsupervised extraction of video highlights via robust recurrent auto-encoders," in *Proceedings of the IEEE international conference on computer vision*, pp. 4633–4641, 2015.

[37] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen, "Deep autoencoding gaussian mixture model for unsupervised anomaly detection," in *International Conference on Learning Representations*, 2018.

[38] B. Hu, B. Gao, W. L. Woo, L. Ruan, J. Jin, Y. Yang, and Y. Yu, "A lightweight spatial and temporal multi-feature fusion network for defect detection," *IEEE Transactions on Image Processing*, vol. 30, pp. 472–486, 2021.

[39] J. Ahmed, B. Gao, and W. L. Woo, "Sparse low-rank tensor decomposition for metal defect detection using thermographic imaging diagnostics," *IEEE Transactions on industrial informatics*, vol. 17, no. 3, pp. 1810–1820, 2021.

[40] Y. Wang, H. Yang, X. Yuan, Y. A. Shardt, C. Yang, and W. Gui, "Deep learning for fault-relevant feature extraction and fault classification with stacked supervised auto-encoder," *Journal of Process Control*, vol. 92, pp. 79–89, 2020.

[41] D. Gong, L. Liu, V. Le, B. Saha, M. R. Mansour, S. Venkatesh, and A. van den Hengel, "Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 1705–1714, IEEE, 10/27/2019 - 11/2/2019.

[42] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *International conference on machine learning*, pp. 4393–4402, 2018.

[43] Y. Qi, C. Shen, D. Wang, J. Shi, X. Jiang, and Z. Zhu, "Stacked sparse autoencoder-based deep network for fault diagnosis of rotating machinery," *IEEE Access*, vol. 5, pp. 15066–15079, 2017.

[44] W. Sun, S. Shao, R. Zhao, R. Yan, X. Zhang, and X. Chen, "A sparse auto-encoder-based deep neural network approach for induction motor faults classification," *Measurement*, vol. 89, pp. 171–178, 2016.

[45] L. Wang, Z. Zhang, J. Xu, and R. Liu, "Wind turbine blade breakage monitoring with deep autoencoders," *IEEE Transactions on Smart Grid*, vol. 9, no. 4, pp. 2824–2833, 2018.

[46] G. Jiang, P. Xie, H. He, and J. Yan, "Wind turbine fault detection using a denoising autoencoder with temporal information," *IEEE/ASME Transactions on Mechatronics*, vol. 23, no. 1, pp. 89–100, 2018.

[47] C. Liu, S. Ghosal, Z. Jiang, and S. Sarkar, "An unsupervised spatiotemporal graphical modeling approach to anomaly detection in distributed cps," in *Proceedings of the 7th International Conference on Cyber-Physical Systems*, ICCPS '16, IEEE Press, 2016.

[48] W. Yan, P. Guo, L. gong, and Z. Li, "Nonlinear and robust statistical process monitoring based on variant autoencoders," *Chemometrics and Intelligent Laboratory Systems*, vol. 158, pp. 31–40, 2016.

[49] G. S. Chadha, A. Rabbani, and A. Schwung, "Comparison of semi-supervised deep neural networks for anomaly detection in industrial processes," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, vol. 1, pp. 214–219, 2019.

[50] C. Kim, J. Lee, R. Kim, Y. Park, and J. Kang, "Deepnap: Deep neural anomaly pre-detection in a semiconductor fab," *Information Sciences*, vol. 457-458, pp. 1–11, 2018.

[51] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," in *International conference on artificial neural networks*, pp. 52–59, 2011.

[52] H. T. M. Tran and D. Hogg, "Anomaly detection using a convolutional winner-take-all autoencoder," in *Proceedings of the British Machine Vision Conference 2017*, 2017.

[53] M. Ribeiro, A. E. Lazzaretti, and H. S. Lopes, "A study of deep convolutional auto-encoders for anomaly detection in videos," *Pattern Recognition Letters*, vol. 105, pp. 13–22, 2018.

[54] C. Zhang, D. Song, Y. Chen, X. Feng, C. Lumezanu, W. Cheng, J. Ni, B. Zong, H. Chen, and N. V. Chawla, "A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 1409–1416, 2019.

[55] B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong, "Towards k-means-friendly spaces: Simultaneous deep learning and clustering," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3861–3870, 2017.

[56] X. Guo, X. Liu, E. Zhu, and J. Yin, "Deep clustering with convolutional autoencoders," in *International conference on neural information processing*, pp. 373–382, 2017.

[57] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, "Deep clustering for unsupervised learning of visual features," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[58] K. Ghasedi Dizaji, A. Herandi, C. Deng, W. Cai, and H. Huang, "Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[59] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh, "Clustering with bregman divergences," *Journal of Machine Learning Research*, vol. 6, no. Oct, pp. 1705–1749, 2005.

[60] W. Xu, X. Liu, and Y. Gong, "Document clustering based on non-negative matrix factorization," in *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, SIGIR '03, (New York, NY, USA), pp. 267–273, Association for Computing Machinery, 2003.

[61] J. Bruna and S. Mallat, "Invariant scattering convolution networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1872–1886, 2013.

[62] G. S. Chadha, U. Panara, A. Schwung, and S. X. Ding, "Generalized dilation convolutional neural networks for remaining useful lifetime estimation," *Neurocomputing*, vol. 452, pp. 182–199, 2021.

[63] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[64] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, p. 3, 2013.

[65] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008.

[66] T. J. Rato and M. S. Reis, "Fault detection in the tennessee eastman benchmark process using dynamic principal components analysis based on decorrelated residuals (dpca-dr)," *Chemometrics and Intelligent Laboratory Systems*, vol. 125, pp. 101–108, 2013.

[67] E. L. Russell, L. H. Chiang, and R. D. Braatz, "Fault detection in industrial processes using canonical variate analysis and dynamic principal component analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 51, no. 1, pp. 81–93, 2000.

[68] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, vol. 31, no. 3, pp. 606–660, 2017.

III. Generalized dilation convolutional neural networks for remaining useful lifetime estimation

Outline

Bibliographic Information

Gavneet Singh Chadha, Utkarsh Panara, Andreas Schwung, Steven X. Ding (2021, September): Generalized dilation convolutional neural networks for remaining useful lifetime estimation. In Neurocomputing 452, pp. 182–199.
doi: 10.1016/j.neucom.2021.04.109.

Author's contribution

The author contributed in the conceptualization, methodology, validation, formal analysis, results investigation, writing the original draft, writing the review and editing and making visualisations.

Copyright Notice

III.1 Abstract

In this paper, we present a novel approach for multivariate time series data analysis with special emphasis on industrial sensor data sets. The approach applies deep convolutional neural networks as a base architecture, incorporating a generalization of the dilated convolution operation on the receptive fields. The dilation operation allows for the aggregation of distributed information in the input space compared to standard convolution operation. The proposed dilation methodology allows for a trainable selection and ignorance of individual sensor features, based on their relevance to the prediction task. Furthermore, arbitrary patterns in the input feature space, including in the temporal dimension of the multivariate time series data can be extracted. In contrast to the standard dilation methodology, the proposed generalized dilation technique is end-to-end differentiable and hence can be trained with off the shelf gradient descent optimizers. Two methodologies have been proposed for the resulting constrained optimization problem namely, the Barrier Function and Top-K sampling approach. We apply the dilated convolutional neural networks to remaining useful lifetime (RUL) estimation problems where degradation recognition over a longer time horizon is crucial for precise estimation. We test the approach on two challenging benchmark datasets, namely the PRONOSTIA Bearing Dataset and the C-MAPSS Aircraft Engine Dataset for RUL prediction. The experimental results obtained for RUL estimation show the superior prediction capability of the proposed generalized dilation methodologies and constitute a new state of the art compared to previous results in literature.

III.2 Introduction

The requirement for current manufacturing environments and machinery to have high availability, poses a great challenge to the production management. Due to the required high workload, unintentional standstill of machines has to be avoided reliably, while the frequency and duration of intentional standstill times should be reduced as much as possible. A common approach to avoid unintentional standstill is the incorporation of recurring maintenance actions in the production schedule. This prevents machines from severe breakdowns, however resulting in potential downtime of the machine due to unnecessary maintenance actions. A plant-wide fault diagnosis and prognosis has therefore been emphasized as crucial in industrial cyber-physical systems [1]. A solution is provided by using condition monitoring and predictive maintenance approaches [2, 3] which normally use a model of the degradation process in the component allowing for a more a detailed prediction of future downtime [4]. However, deriving models for condition monitoring and predictive maintenance can be a challenging and time-consuming task.

In parallel, the upcoming digitization augments modern production environment with considerably more and real-time information about the status of the machines. Using various sensor measurements over the complete process allows us to gain a detailed view on the condition of the overall plant, allowing for continuous monitoring of the production process and assists in analyzing potential weaknesses. Consequently, data-based maintenance with the framework of Prognostics and Health Management (PHM) [5] provides the necessary tools and methodologies to reliably predict the availability or degradation of an asset, based on multivariate time series analysis approaches. Various approaches

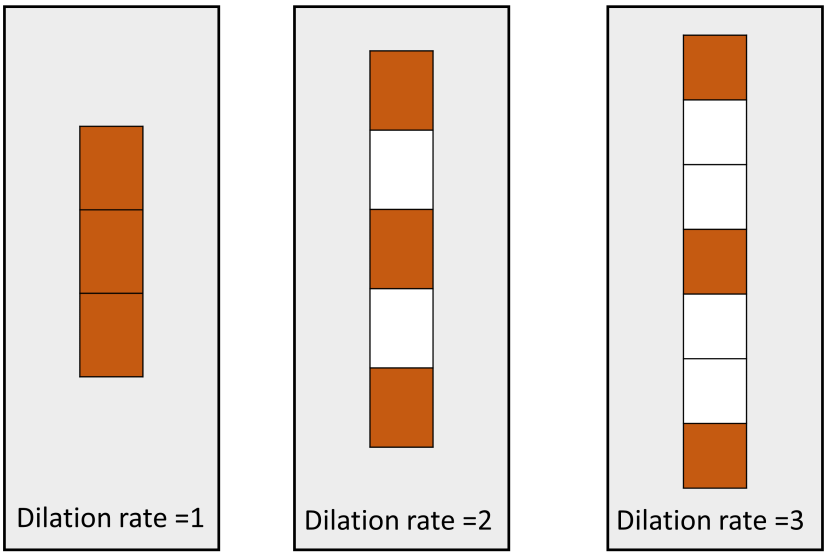


Figure III.1: Dilation kernels with different dilation rates (where rate=1 is the original convolution operation)

have been proposed for data based maintenance, see [6, 5] for overviews. Data-based soft sensor approaches have also been investigated for monitoring of industrial processes in [7]. Among the various approaches, neural networks appeared to be of particular use due to recent advances in the field of deep learning [8]. Particularly, convolutional neural networks (CNN) [9] and recurrent neural networks (RNN) such as long short term memories (LSTM) [10] and gated recurrent units (GRU) [11] have been successfully applied.

In this paper, we present a novel deep learning architecture for multivariate time series data analysis using the concept of dilation in CNNs. Dilated convolution was originally presented for image recognition and image segmentation tasks [12] and is illustrated in the case for a univariate time series in Fig. III.1. Compared to the standard receptive field in CNNs (left), the dilation (middle and right) operation increases the size of the receptive field, while the number of trainable parameters, illustrated as orange squares, are kept the same. The prior fixed dilation rate denotes the relative expansion in the time dimension of the receptive field. The rationale behind dilation for multivariate time series analysis is to provide CNNs with more globally acting receptive fields such that the network can learn and aggregate more distributed features over a prolonged time horizon as compared to the standard convolution operation. However, the dilation rate still is fixed for all the kernels in a convolution layer and an equidistant operation. Furthermore, standard 1-D convolution can only dilate in the temporal dimension, and not in the feature dimension. In this work, we propose a methodology where the dilation is not fixed for each kernel and it can possess a general structure in the temporal as well as in the feature dimension. In this way, Generalized Dilation CNNs (GDCNN) can learn to extract features over a prolonged time horizon as well ignore certain input features, providing a more sparse representation.

Normally, the dilation structure in CNNs is kept fixed during the training. The proposed Generalized Dilation (GD) layer allows training of the dilation structure in an end-to-end manner. The GD layer is defined by masking vectors or matrices for each convolution kernel in the layer, wherein the maskings consist of trainable parameters in the range $[0, 1]$, which mask irrelevant samples from the layer input. Furthermore, these parameters are either constrained with an auxiliary loss function which assures that only a certain amount of parameters are active or they are constrained with a Top-K sampling approach. The proposed methodologies allows for arbitrary dilation structures. Particularly, each convolution filter can learn which part of the input, in the temporal dimension as well as in the feature dimension, is relevant for the prediction task and which is not. The approach is applied to two benchmark data sets in the field of RUL where prediction over a long time horizon and detailed feature selection is crucial. The PRONOSTIA ball bearing data set provided as the data challenge of the 15th PHM conference [13] and the C-MAPSS aircraft engine data set [14] are used as an application example. The results obtained with the proposed GDCNN are compared to standard dilation CNN with fixed dilation and other methods from literature. The results clearly show the superior performance of the approach constituting a new state of the art for the considered data sets compared to existing approaches from the literature.

The contributions of the paper can be summarized as follows:

- We propose GD layers for CNNs designed for multivariate time series data analysis. Particularly, the layer provides adaptive sampling which constitutes a novel way to adjust the structure of the dilated convolution kernels such that the convolution parameters can select the time samples and the sensor channels most relevant for the prediction task. This ability helps in modelling longer input time sequences. The adaptive sampling can vary for each of the convolution kernels in the proposed layer.

- We propose two novel training algorithms for GDCNN which allows for an end-to-end training of the dilation masking parameters in the network. The first algorithm incorporates the use of the barrier function within the standard gradient-based learning process, to learn arbitrary dilation patterns on the time-series data. The second algorithm uses a $Top-K$ sampling approach to impose the hard constraints for the binary optimization problem.

- We apply the proposed approach to two challenging RUL benchmark datasets. We thoroughly analyse the proposed approach on the datasets and compare the results to different baselines as well as to results from the literature where we achieve new state-of-the-art results on these data sets.

The paper is organized as follows. Sec. III.3 discusses the work related to our paper. CNNs and dilated CNNs are described in Sec. III.4 with a special emphasis on time series analysis. In Sec. III.5 we present the newly developed end-to-end training approach for dilated CNNs. In Sec. III.6 and Sec. III.7 we provide results and comparisons on two benchmark data sets. Sec. III.8 concludes the paper.

III.3 Related Work

In this section, we discuss the work related to our paper. We discuss existing approaches for data-based RUL estimation first and then turn to work on dilated CNN.

Data-Based RUL estimation: Data-based RUL estimation is a prominent research topic where various approaches ranging from the application of support vector regression [15], hidden markov models [16], neuro-fuzzy systems [17] to exponential regression [18] have been reported. Refer to [19, 6, 5] for extensive overviews. The work [20] illustrates three RUL estimation methodologies for the ball bearing dataset where the authors employ several feature engineering strategies with time domain and frequency domain features. In [21], a time-frequency based feature extraction method from vibration signals for estimating RUL of bearings is proposed. These extracted features are then used along with curve fitting and extended Kalman filtering algorithms.

In recent years, a lot of work is devoted to the application of deep neural networks like deep belief networks [22], Recurrent Neural Network (RNN) like Long Short Term Memory (LSTM) [23], CNNs [24] and Deep Autoencoder [25] for RUL estimation. This is mainly due to their inherent capacity to train representative features directly from data without the costly tuning of predefined features, thereby reaching new state-of-the-art results. As such, an ensemble method based on multiobjective deep belief networks for RUL estimation has been introduced in [26]. LSTM neural network-based RUL estimation and fault diagnosis techniques for aircraft engines are proposed in [27] where the results reveal that standard LSTMs work better than other models. In [28], LSTM neural networks are used for RUL estimation of aircraft engines and the results are compared with various machine learning models based on RMSE scores. In [29], adaptive kernel spectral clustering is combined with LSTMs to determine the health status of bearings. In [30], RUL prediction is considered as an interval classification problem and ordinal regression based on RNNs is performed. In [23], a novel method which involves the use of recurrent neural networks for establishing a health indicator for bearing RUL estimation problems is proposed. The study [31] uses an Hybrid LSTM-CNN approach where the input is fed parallel to an LSTM model and a CNN model, followed by the fusion in fully connected layers. This is in stark contrast to our approach which contains only CNN layers and fully connected layers. Also, our model contains ≈ 20 times less parameters than the model in [31]. The study [32] proposes a two step approach wherein first a Restricted Boltzmann Machine (RBM) is trained in an unsupervised manner and later an LSTM model is trained in a supervised manner. This is very dissimilar to our proposed model, where we have only one step of supervised training. Also, our model contains $\approx$ 100 times less parameters than the model in [32]. The study [33] also proposes a two step training approach. The first step includes training a RNN autoencoder for feature extraction. These extracted features are employed to map the original training instances into one-dimensional health index (HI) curves which have to be saved in a database. Finally, a similarity-based curve matching approach is adopted for predicting the degradation trend of a test instance via template matching. The approach is strongly orthogonal to our proposed model.

Moreover, RNNs are in general difficult to train due to their sequential processing of data and more prone to overfitting due to their high number of parameters. Particularly, a large amount of labelled data is required to sufficiently train RNNs, which is in general

not the case in industrial time-series data. Furthermore, RNNs computes the hidden representation from the first time-step till the last time-step in a sequential manner. CNNs however, do not require this sequential calculation of hidden representations and can calculate multiple hidden representations in parallel taking advantage of the massive parallel processing available in modern GPU. This leads to faster training times in CNNS.

In contrast to CNNs, RNNs can take as input a different time window. However, RNN models also require to have the same time windows as input for each mini-batch. Therefore, the RNN models can increase or decrease their time range only under the assumption that batch-size is 1, which provides a highly noisy estimate of the gradients during training. Furthermore, in a real industrial environment, the RNN model can take input only with a specific time window which is relatively small because of the vanishing and exploding gradient problem [34]. Methods like Truncated Backpropagation Through Time [35] have to be then employed for bypassing such problems. Moreover, it is rather unlikely that in a real industrial environment, the time-window that the model take as input changes over its lifetime. For e.g. the data received from an industrial PC (IPC) has a fixed length, which is the same as the cycle time of the IPC. At test time, our proposed model and RNNs function very similarly. Therefore, the model behaves identically to RNNs when the model is in production. However, due to the dilation masks (particularly with the better performing hard maskings covered in Sec.III.5.2), it is possible to reduce the number of parameters in deployment, as masked out parameters need not be included in the model during production.

CNNs have been applied to RUL prediction and condition monitoring in various settings. In [24], RUL prediction using a double-CNN architecture has been proposed. A CNN architecture combining RUL prediction and fault recognition using a joint loss function has been presented in [36]. A transfer learning approach for fault diagnosis for rotary machines based on CNNs is given in [37]. The work [38] proposes a novel deep learning architecture using a multi-scale feature extraction scheme using convolutional neural networks along with time-frequency domain information for prognostics and feature extraction and apply this to a ball bearing data set. The study [39] uses a two step approach where a Deep Convolutional Generative Adversarial Network (DCGAN) is trained at the pretraining stage to generate synthetic data in an unsupervised manner. This synthetic data is then used with an LSTM and fully connected model for fine-tuning purpose in a supervised manner. The study [40] proposes a multitask learning approach with CNN by simultaneously learning two tasks i.e. RUL estimation and health condition prediction. Therefore, the study uses an extra set of target labels for estimating the health condition of the engines. These extra target labels have to be generated synthetically as they are not part of the original dataset. Recently, [41] propose an approach combining CNN with LSTM for RUL estimation, where the CNN serves as a feature extractor. In [42], a similar approach is presented. An encoder-decoder structure with LSTM memory is presented in [43].

Common to the mentioned approaches is that they translate well known architectures from other domains like image recognition or speech recognition to RUL estimation without architectural changes which limit their suitability for RUL estimation. Furthermore, it can be observed that most works employ CNNs to data sets with a low number of sensor channels which can be attributed to the strong increase of filter parameters when the widely used 1D kernels are applied. Such architectures are inherently hard to train

due to over-fitting issues. In contrast to the previously mentioned approaches, we develop a novel architecture which uses CNNs as a baseline but is specifically designed for the application in time series data sets. The approach allows to reduce and control the number of active convolution parameters both by adaptively scaling down the input dimensions per convolution channel and by learning to select relevant sensor channels or temporal information. As this requires for novel training algorithms, we develop a barrier function approach which can be extended to other domains like e.g. permutation-based neural networks [44]. The log-barrier method has been used in [45]for a budget aware pruning methodology. On the contrary, we propose the selection of relevant structure in a receptive field for efficient feature extraction. Top-K sampling approach has been presented in [46] in the context of Gumbel-Max trick for sampling from a categorical distribution. On the contrary, we sample the Top-K elements from the masking parameters in the dilation convolution layer.

Dilated CNN: Convolution with dilated filters were first introduced in [47] and [48] in the context of wavelet decomposition. Dilated convolutions have been initially presented in [12] to allow for multi-scale context aggregation without downsampling of the resolution. Since then, dilation networks have been used in semantic segmentation methods due to their ability to capture large context while preserving fine details [49, 50]. Applications of dilated CNNs include time dilated convolutions [51], modelling long-distance genomic dependencies [52], entity recognition and text modeling [53]. Dilated residual networks are introduced in [54]. Application of dilation to time series analysis is provided in WaveNets [55]. However, in all of these works the dilation structure is fixed. In contrast, we keep the dilation structure end-to-end trainable for each convolution channel.

Training of dilation factors has been presented in [56] where a dilation factor is trained for each channel of the convolution layer. However, the dilation factor is defined in $\mathbb{R}$ instead of $\mathbb{Z}_+$ as in the original derivation. This comes at the cost of calculating the output map by utilizing a bi-linear transformation in the regular case of fractional dilation factors. Furthermore, the structure of the dilation is fixed. More general structures are allowed by active convolutions [57], however with low deviations from the original convolution kernel. Deformable convolutional networks as introduced in [58] and further improved in [59] allowing for similar convolution structures as proposed by our approach. There, each point in the convolution grid is augmented with a learnable real-valued offset. As in [56], a bilinear transformation is needed before applying the convolution. This transformation however, renders these approaches unsuitable for multivariate time series analysis for two reasons. First, the interpolation between time samples destroys the equidistant sampling of the time series data. Second and more critical, the bilinear transformation would interpolate between different sensor channel data points which result in meaningless values. Contrary, we hold on to integer-valued dilation factors making learnable dilations applicable to multivariate time series data. Furthermore, we allow for arbitrary structures of the kernels using end-to-end trainable binary masking matrices.

III.4 Convolutional and Dilated Neural Networks

In this section, we first discuss the CNNs from the viewpoint of multivariate time series analysis. We then introduce dilation operations into CNNs and discuss their properties for time series analysis.

III.4.1 Convolutional Neural Networks

The idea of CNN has been introduced in the early nineties in [9], specifically for image recognition. However, its popularity increased only after a deep CNN model had beaten the state-of-the-art model in the ImageNet image classification challenge in [60]. A typical CNNs consists of a sequence of convolution layers with nonlinear activation and subsequent pooling layers. The convolution layer normally consists of several parallel channels which apply the convolution operation similar to standard filter algorithms in signal processing where the filters are trainable. Furthermore, in contrast to standard fully connected networks, CNNs employ weight sharing with tied weights to handle the high dimensional input space provided by images or multivariate time series. Hence, CNNs are proven to be the state-of-the-art architecture for various machine learning tasks ranging from image processing, natural language processing to time series analysis [61]. The input space for multivariate time series analysis consists of individual sensor signals and hence, does not appear in the form of an image.

We consider multivariate a time series signal $\{x_t\}_{t=1}^{T}$ with $x \in \mathbb{R}^m$, m the number of sensor channels and T the length of the time series. The complete dataset consists of N such time-series signals. In supervised learning setting, each of these N time series signals, have the corresponding labels as $\{y_b\}_{b=1}^{N}$ which can be either class labels, i.e. $y \in \mathbb{Z}^m$ or continuous variables, i.e. $y \in \mathbb{R}^d$ with d the output dimension.

In vanilla CNNs for image recognition, the convolution operation is performed by 2D convolutions. Usually a square receptive field or kernel of size $n_r \times n_r$ strides over the $n \times n$ dimensional image where $n_r << n$. Formally, this yields

$$h_{ij} = (X * K)_{ij} = \sum_{f=0}^{n_r-1} \sum_{h=0}^{n_r-1} X_{i+f,j+h} \cdot K_{i+f,j+h} + b, \tag{III.1}$$

where h_{ij} denotes the output of the $(i,j)^{th}$ receptive field in the input, $X_{i+f,j+h}$ are the elements in the receptive field of the input image or sequence, $K_{i+f,j+h}$ is the convolution kernel, b denotes the bias for the convolution kernel and n_k the weight kernel size or the receptive field size. Therefore, the size of each convolution kernel is, $K \in \mathbb{R}^{n_r \times n_r}$.

The 2D convolution holds under the assumption that spatial relationships are available in the input domain like in the case of images. However, this is not the case in time-series data where the sensor channels do not have any spatial relationship. Consequently, 2D convolutions are inappropriate for time-series data as a spatial relation between arbitrary stacked sensor channels is not ensured. Hence, 1D convolutions are predominantly applied to multivariate time series analysis. In 1D-convolution, the convolution kernel ranges over the entire sensor channel size, i.e. the size of each convolution yields $n_r \times m$. Formally, the 1D operation is defined as

$$h_i = (X * K)_i = \sum_{f=0}^{n_r-1} X_{i+f} \cdot K_{i+f} + b, \tag{III.2}$$

where h_i denotes the output of the $(i)^{th}$ receptive field in the input. In essence, the convolution kernels span over the whole column-space with a receptive field and kernel of size $K \in \mathbb{R}^{n_r \times m}$. Evidently, if the number of sensor channels is high, the size of the convolution kernels drastically increases which severely weakens the weight sharing

capabilities of the CNN. Furthermore, as n_r is normally chosen to be small, kernels can extract local information only, i.e. features over short time horizons. Longer horizons can only be tackled by increasing the depth of the CNN resulting in even more parameters or by increasing the fixed dilation rate. In the following section, we propose dilated convolutions mitigating the above drawbacks of 1D convolutions.

III.4.2 Dilated Neural Networks

The 2D dilated convolution [12] can be described by

$$h_{ij} = (X *_l K)_{ij} = \sum_{f=0}^{n_k-1} \sum_{h=0}^{n_k-1} X_{i+l*f,j+l*h} \cdot K_{i+f,j+h} + b \tag{III.3}$$

where the term $*_l$ denotes the dilated convolution and l is the dilation factor. Note that the standard convolution operation is obtained for $l = 1$. By changing the dilation factors, the dilated convolution operator applies the same filter (with the same number of parameters) at different ranges and hence, allows for better multiscale context aggregation [12]. The 1D dilated convolution operation can be formally defined as

$$h_i = (X *_l K)_i = \sum_{f=0}^{n_k-1} X_{i+l*f} \cdot K_{i+f} + b \tag{III.4}$$

where the term $*_l$ denotes the dilated convolution and l is the dilation factor. The effect of the 1D dilation operation for a uni-variate case is illustrated in Fig. III.1, which shows the increasing size of the filter with increasing dilation factor.

In the original work, the dilation factor and its structure is kept fixed during training. Furthermore, different dilation structures within one receptive field cannot be represented by ordinary dilations. To remove these inflexibilities would be beneficial particularly for time series data analysis. Hence, we aim to provide more versatility in that we intend to make the dilation structure variable over the receptive field and make this variability trainable. By generalizing this idea we can allow for even more general patterns in the receptive field.

To this end, we will first derive an alternative representation of the 2D dilation operation as follows. This can be generalized to 1D dilation from Eq. (III.4). The dilation operation can be interpreted as a conventional convolution operation with receptive field size $n_d \times n_d$ with $n_r \leq n_d$. This yields to a convolution weight matrix $W \in \mathbb{R}^{n_d \times n_d}$ in which $(n_d \times n_d) - (n_r \times n_r)$ elements are fixed to zero such that the active weights (weights unequal zero) sum to $n_r \times n_r$. To accomplish this, we define vectors $\psi_l \in \{0,1\}^{n_d}$ and $\psi_r \in \{0,1\}^{n_d}$ and matrices Ψ_l and Ψ_r such that

$$\mathrm{diag}(\Psi_l) = \psi_l, \quad \mathrm{diag}(\Psi_r) = \psi_r. \tag{III.5}$$

Then, we define the new weight matrix to be

$$\tilde{W} = \Psi_l \cdot W \cdot \Psi_r, \tag{III.6}$$

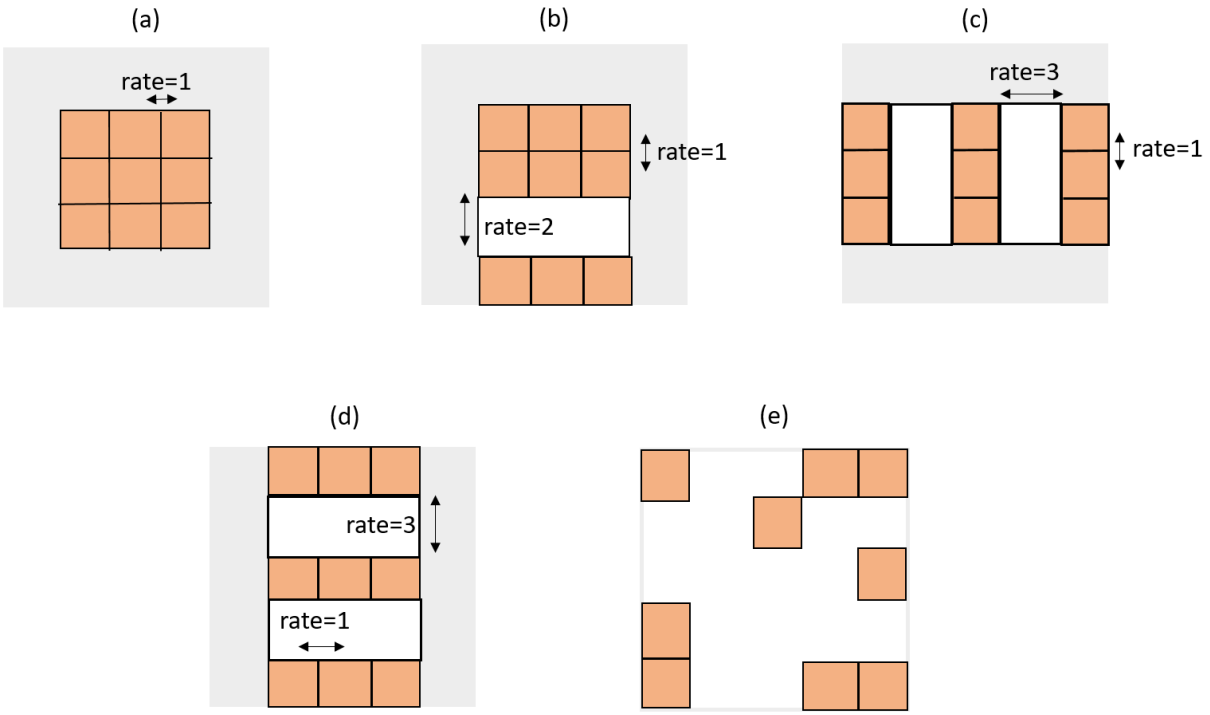


Figure III.2: Different configurations of the parameters: (a) Original convolution, (b) Dilation with varying dilation rates; (c) Dilation in horizontal dimension only; (c) Dilation in vertical dimension only; (e) Arbitrary dilation kernel.

which provides a generalization to the dilation layer recapitulated in the previous section. In fact, the dilation with active weights of size 3×3 and dilation factor of $l = 2$ can be obtained by defining $\psi_l = \psi_r = [1\ 0\ 1\ 0\ 1]^T$. Note that by arbitrarily setting the components of ψ_l and ψ_r to zero and one while at the same time forcing the total number of ones per ψ_l and ψ_r to n_r, arbitrary dilation-like patterns in the weight matrix $\tilde{W}$ can be generated. Formally, this results in imposing constraints

$$\psi_l^T \cdot \mathbf{1} \leq n_r, \quad \psi_r^T \cdot \mathbf{1} \leq n_r, \tag{III.7}$$

where $\mathbf{1}$ denotes the all-one vector.

III.4.3 Generalized Dilated Neural Networks

We can further generalize the dilation operation by defining a matrix $\Psi \in \{0,1\}^{n_d \times n_d}$ and define a new weight matrix

$$\tilde{W} = W \odot \Psi \tag{III.8}$$

where $\odot$ denotes element-wise multiplication and impose constraints

$$\mathbf{1}^T \Psi \cdot \mathbf{1} \leq n_r^2. \tag{III.9}$$

The constraint assures that the number of weights unequal to zero is less than or equal to the kernel size. This reparameterization of the weight matrix $\tilde{W}$ allows for generalized

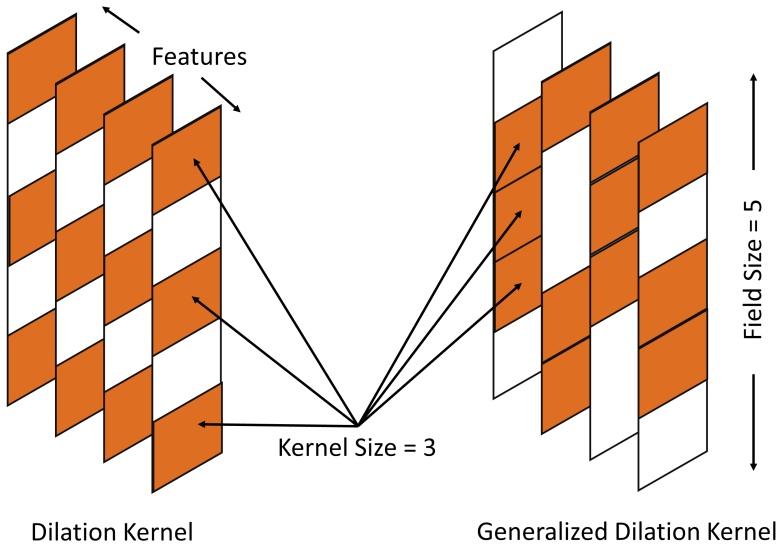


Figure III.3: Comparison between standard dilation structure (left) and generalized dilation structure (right) for uni-variate features and same kernel size

dilation-like patterns. An illustration of some examples for the different possible patterns is given in Figure III.2. Particularly, some choices of the parameters are especially useful in practice. The original convolution operation (Fig. III.2 (a)) is obtained by setting $\psi_l = \psi_r = [0\ 1\ 1\ 1\ 0]$, a dilation with rate 2 yields $\psi_l = \psi_r = [1\ 0\ 1\ 0\ 1]$ and non-equidistant patterns as in Fig. III.2 (b) yields to $\psi_l = [0\ 1\ 1\ 0\ 1]$ and $\psi_r = [0\ 1\ 1\ 1\ 0]$.

However, this equidistant choice of the dilation might not be best to obtain representative features in some cases. Specifically, typical features in time series data might only depend on a subset of sensor channels. For such cases dilation patterns as shown in Fig. III.2 (c) using setting $\psi_l = [0\ 1\ 1\ 1\ 0]$ and $\psi_r = [1\ 0\ 1\ 0\ 1]$ can be beneficial. Similarly, a vertical dilation can be trained (Fig. III.2 (d)), eventually leading to subsampling of the time dimension. Ultimately, arbitrary dilation patterns as illustrated in Fig. III.2 (e) can be obtained by setting

$$\Psi = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{bmatrix}. \tag{III.10}$$

Note that even higher dilation rates and broader patterns can be obtained by applying larger receptive fields with $n_r << n_d$. A comparison between standard dilation structures and generalized dilation structures for univariate input features is illustrated in Fig. III.3, where a receptive field size of 5 and kernel size of 3 is chosen. All the kernels in the standard dilation structure are equidistant, in contrast to the the generalized dilation structure in which each kernel can take arbitrary patterns.

III.5 End-to-end Training of Dilated Neural Networks

The training procedure for the binary masking matrices is detailed in this section. Making the matrices trainable, allows for an adjustment of the matrices during training and could potentially reduce the number of active convolution parameters required for the given problem. Hence, in this section, we derive an end-to-end training scheme for the proposed networks. However, optimizing the masking weight matrices and vectors results in optimizing binary variables which makes the learning problem combinatorical and does not directly allow for end-to-end training. To circumvent this, we follow the standard softening of binary parameters approach as e.g. in [62] and apply a soft binary variable by running continuous vectors $\tilde{\psi}_l, \tilde{\psi}_r \in \mathbb{R}^{n_d}$ and matrices $\tilde{\Psi} \in \mathbb{R}^{n_d \times n_d}$ through a sigmoidal activation function, such that

$$\tilde{\psi}_l = \sigma(\psi_l), \quad \tilde{\psi}_r = \sigma(\psi_r), \quad \tilde{\Psi} = \sigma(\Psi). \tag{III.11}$$

This reparameterization bounds the weights to the interval (0, 1), while the sigmoid functions tends towards its boundaries as training proceeds. The sigmoidal function is used to bound the masking weights to the interval (0, 1) which is required for selecting input samples from the multivariate sensor inputs. Other functions can also be applied for such a reparameterization. However, these functions should have the range (0, 1) and have a real gradient over its domain. Consequently, we end up with a trainable structure consisting of network parameters ω, i.e. kernel weights of the convolution/dilation kernels and weights of the fully connected layers, and the weight vectors or matrices $\tilde{\Psi}$. This can be trained by backpropagation using stochastic gradient descent on the standard loss functions $L_s(\omega, \tilde{\Psi})$ like cross entropy loss for classification or mean squared error for regression tasks.

However, we have to consider the additional constraints on the parameters $\tilde{\Psi}$ as given in Eq. (III.7) and Eq. (III.9). These constraints have to be fulfilled during training and hence, should be imposed as hard or soft constraints. In general, various approaches exist to incorporate hard constraints in stochastic gradient descent algorithms, namely barrier functions, projection methods and active set methods [63]. In this work we propose two methods to apply these constraints namely a.) differentiable barrier functions approach which contribute to the loss function only if the constraints are not satisfied and b.) $Top-K$ sampling approach to assign integer values to the binary masking parameters similar to the approach as in [46]

III.5.1 Barrier Function

We define the total loss of the model $L_s(\omega, \tilde{\Psi}) + L_b(\tilde{\Psi})$ where L_b is the barrier function loss. As the barrier loss only applies to the masking vectors and parameters, $\tilde{\Psi}$ and $\tilde{\psi}_l, \tilde{\psi}_r$ respectively, we can parameterize the barrier loss directly as the gradient of the barrier loss $\nabla_\Psi L_b$ dependent on the layer-inherent masking matrix Ψ in the form

$$\nabla_{\tilde{\Psi}} L_b = b_c(\tilde{\Psi}) + b_r(\tilde{\Psi}) + b_a(\tilde{\Psi}), \tag{III.12}$$

where b_c, b_r and b_a represent the different barrier functions calculated with different amount of penalties on the constraint to be fulfilled. In this work, we define

$$b_c(x) = \left(e^{\alpha_1 \cdot (x - n_r)} - \alpha_2 \cdot (x - n_r)\right) - \alpha_3, \tag{III.13}$$

$$b_r(x) = \max(e^{\alpha_1 \cdot (x - n_r)} \cdot \alpha_2 \cdot (x - n_r), \alpha_3), \tag{III.14}$$

$$b_a(x) = \max(e^{\alpha_1 \cdot \left(x^2 - {n_r}^2\right)} \cdot \alpha_2 \cdot \left(x^2 - {n_r}^2\right), \alpha_3), \tag{III.15}$$

where we experiment with different parameters α_1, α_2 and α_3 as illustrated in Fig. III.4 for the barrier function $b_c(x)$. Finally, the gradient for end-to-end training of the masking parameter $\tilde{\Psi}_{ij}$ yields

$$\nabla_{\tilde{\Psi}_{ij}} = \frac{\partial L_s(\omega, \tilde{\Psi})}{\partial \tilde{\Psi}_{ij}} + \nabla_{\tilde{\Psi}_{ij}} L_b. \tag{III.16}$$

In this way, the gradient of the masking parameters is the sum of how much it deviates from the constraint and their contribution to the classification or regression loss $L_s(\omega, \tilde{\Psi})$.

It can be inferred from Fig. III.4 that regardless of parameters (α_1, α_2 and α_3), the barrier function has an inherent tendency to return higher penalties if the boundary condition $(x - n_r)$ is violated. So, if the sum of the elements of $\tilde{\Psi}_{ij}$ at the beginning of training is large, it will result in higher penalties and thereby larger gradients. This further steers the masking parameters very early in the training process into saturation regions. Therefore, along with the initialization of neural network parameters, the initialization of masking parameters becomes important. The authors in [46] suggests initializing the masking parameters in a basin where the gradients are not huge. Therefore, the parameters can

III.5.2 Top-K Sampling

In this approach, we consider the top-K values or the maximum K values from the $\tilde{\psi}_l, \tilde{\psi}_r$ and matrices $\tilde{\Psi}$, and set them to 1 and the rest to 0. More formally, we modify the elements in the masking matrix as

$$\hat{\Psi}_{ij} = \begin{cases} 1, & \text{if } \tilde{\Psi}_{ij} \in \mathcal{Y}_{max-K} \\ 0, & \text{else .} \end{cases} \tag{III.17}$$

where $\mathcal{Y}_{max-K}$ denoted the K max elements in the masking matrix $\tilde{\Psi}$. In this way, we achieve integer binary values for the masking matrix $\hat{\Psi}$ in the forward pass. For the backward pass or updating the binary parameters, we take the gradient estimates as if the forward pass was according to Eq. (III.11), and update the parameters using Eq. (III.16). This leads to integer binary values in the forward pass but their sigmoidal values are used for the backward pass. An advantage of the top-k operation is that we force the network to select the *exact* weights for the prediction task and loss calculation. This is in contrast to the barrier function approach where due to the sigmoid function, a scaled version of the weights would have been selected.

III.5.3 Implementation and Applicability of GDCNN

The training time for the proposed generalized dilation layer over the standard dilation layer was ≈ 2 seconds more for a complete epoch on an Intel Xeon Silver 4112 processor

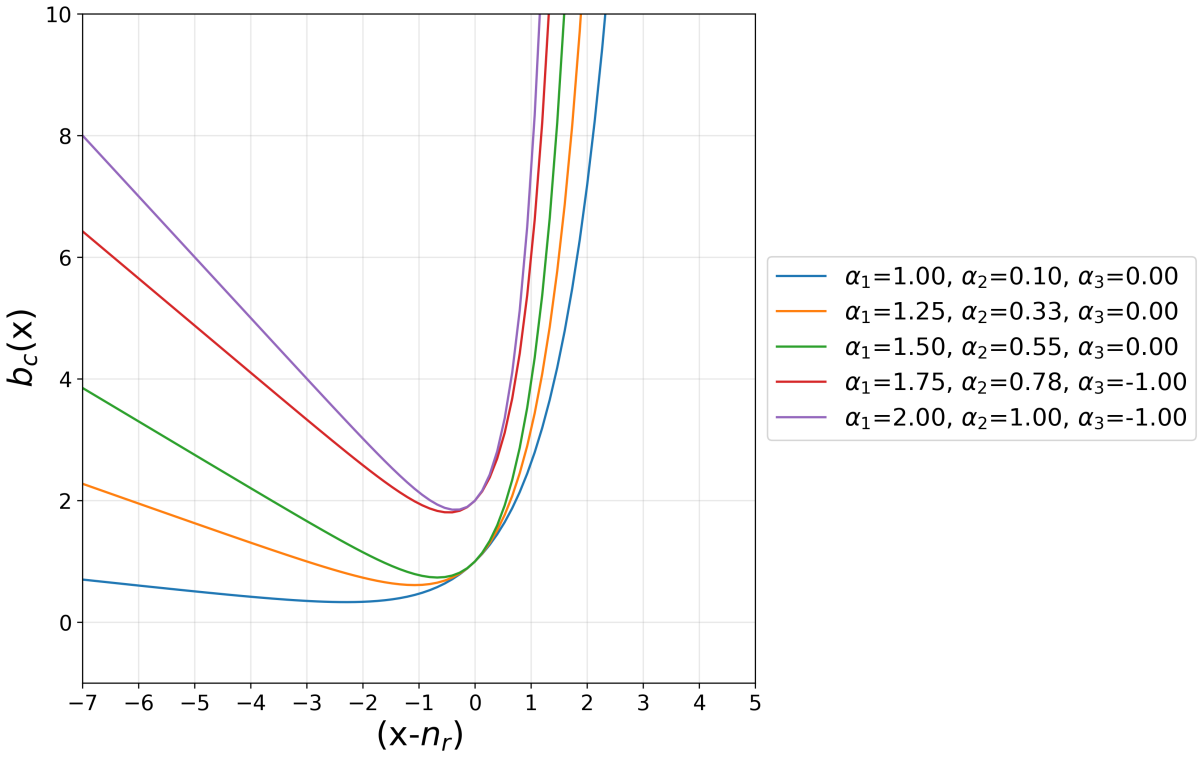


Figure III.4: Barrier functions b_c with different parameters α_1, α_2 and α_3.

on both the tested datasets. The proposed method has been developed and implemented in the PyTorch [64] deep learning framework.

There are extra parameters in the case of the barrier function training approach described in Section III.5.1. The number of extra parameters depends on the kernel size n_r, size of the receptive field n_d, and the number of channels in a layer. Therefore, for the proposed architecture where, there would be $(n_d - n_r) \times (number - of - channels)$ more parameters in the model.

However, in the case of Top-K sampling training approach III.5.2, there is no increase in the number of parameters as compared to the original dilation model because all the extra learnable parameters are zero. Therefore, it should be noted that the size of the receptive field is increased compared to the standard CNN while keeping the number of parameters constant.

The proposed generalized dilation methodology is applicable to every CNN architecture, including yolo [65] and inception [66]. This is because the methodology works in parallel with the convolution operation which is common to every CNN architecture.

III.6 Experimental Results on Bearing Dataset

In this section, we report experimental results on the PRONOSTIA ball-bearing data set and thoroughly discuss the different architectural effects with the proposed dilation networks. Furthermore, we compare the proposed approaches with baseline vanilla convolution neural networks and compare to results of existing approaches from the literature.

III.6.1 Data Sets Description

The dataset was provided by FEMTO-ST Institute, France as part of a series of experiments conducted in their PRONOSTIA laboratory platform [13]. PRONOSTIA is an experimental platform was designed for producing accelerated degradation in ball bearings for testing and validating bearing diagnostics, prognostics and fault detection. The three main parts of the PRONOSTIA experimental rig consists of a rotating part with an asynchronous motor and gearbox, a degradation generation part with a pneumatic jack and a clamping ring and finally a radial force measurement part. The experimental data contain run-to-failure degradation of ball bearings and contains information about the rotating speed, load force, temperature and vibration of the bearings. For the challenge, data representing three different operating conditions was collected. The dataset consists of in total 6 run-to-failure bearing cases i.e 2 for each operating condition, that are used as the training data. The RUL of 11 bearings under the same 3 operating conditions must be estimated using the proposed models. There were no assumptions made regarding the type of bearing failure occurring. The fact that the challenge has noisy training data and high variability in experiment duration (1 to 7 hours), makes accurate predictions extremely challenging. The three different operating conditions for the experiments are 1800 rpm and 4000 N radial load (Cond. 1), 1650 rpm and 4200 N radial load (Cond. 2) and 1500 rpm and 5000 N radial load (Cond. 3) and includes horizontal and vertical vibration signals measured by accelerometers at a sampling frequency of 25.6 kHz. In addition to the vibration information, temperature information was also collected during the experiments. However, as the temperature data is not available for all training and testing bearings, only horizontal and vertical acceleration data is considered.

III.6.2 RUL evaluation metrics

The evaluation metric for RUL estimation of both data sets have already been defined in the corresponding PHM challenge [13, 14] and are used for comparison purposes. The RUL evaluation metric for the PRONOSTIA dataset is given by the error percentage:

$$E_i = 100 \cdot \frac{\mathrm{RUL}_i - \widehat{\mathrm{RUL}}_i}{\mathrm{RUL}_i}, \tag{III.18}$$

where $i = 1, 2, \ldots, 11$ denotes the number of the test bearing, RUL_i is the actual RUL and $\widehat{\mathrm{RUL}}_i$ is the predicted RUL. Positive E_i are considered as early prediction, negative E_i are considered as late prediction. The accuracy score of RUL estimate for bearing i is defined as follows:

$$A_i = \begin{cases} e^{-ln(0.5)\cdot E_i/5}, & \text{if } E_i \leq 0 \\ e^{ln(0.5)\cdot E_i/20}, & \text{if } E_i > 0. \end{cases} \tag{III.19}$$

The final score of all RUL estimates is defined as mean of all bearing RUL estimate scores, i.e.

$$S_i = \frac{1}{11} \sum_{i=1}^{11} A_i. \tag{III.20}$$

III.6.3 Training Settings

Data Preprocessing and Sequence Generation

Since the vibration signals from the bearing dataset consists of a high variability and noise, the sensor data is normalized to be within the range of $[-1, 1]$ using Min-Max normalization method as

$$X_{norm}^{i,j} = (Max - Min) \times \left\{ \frac{x^{i,j} - \hat{x}^j}{\tilde{x}^j - \hat{x}^j} \right\} + Min, \forall i, j \tag{III.21}$$

where $x^{i,j}$ denotes the original i-th data point of the j-th sensor, and $X_{norm}^{i,j}$ is the normalized value of $x^{i,j}$. $\tilde{x}^j$ and $\hat{x}^j$ denote the maximum and minimum values of the original measurement data from the j-th sensor respectively. Min and Max are the normalization range required, in this case -1 and 1 respectively. The train data set has been normalized using Eq. (III.21) and the same scaling parameters are used for the test data set.

The sequence generation step can be interpreted as generating of smaller overlapping local subsections from the original time series for maximum data utilization. Theoretically, a complete single bearing dataset could be used as a single time series. However, that would make the the model extremely complex and could also lead to over-fitting as the model has to remember over the complete length of the dataset. Therefore, we follow a similar approach as in [67], where we divide the original time series into a number of small sequences and thus feeding more historical data to CNN to train on. The length of a sequence is a hyperparameter that needs to be tuned.

Loss and Architecture Definition

The loss measure usually used for training CNN for regression problems in literature is Mean Absolute Error (MAE) [38], Mean Square Error (MSE) [36], and Root Mean Square Error (RMSE). However, due to a large RUL prediction horizon of the RUL i.e. $[0, 28030]$ seconds for bearing 1_1, the loss values obtained from the above mentioned functions are drastically large. This causes too large updates of weights during back-propagation even after adjusting the learning rate, forcing the gradients rapidly into saturated regions and unstable convergence. In order to cater to this special requirement we propose a loss function which is not affected by the large prediction horizon. The Root Mean Square Log Error (RMSLE) is utilized as

$$RMSLE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(RUL_i + 1)) - \log(\widehat{RUL}_i + 1)^2}. \tag{III.22}$$

Here the *log* of the estimated RUL and actual RUL which makes it immune to the exploding losses for large RUL prediction horizon. As per the authors knowledge, this is the first use of RMSLE loss function for RUL prediction.

To derive an optimum CNN architecture, we decided to adopt a random search methodology [68] rather than a grid search and choose the best architecture based on the performance on a held out validation set. The hyper-parameter search space that was experimented with includes but not limited to Number of CNN layers=$\{3, 4, 5, 6\}$, Number of Convolution Channels per layer = $\{8, 16, 32, 64, 128, 256\}$, Kernel size

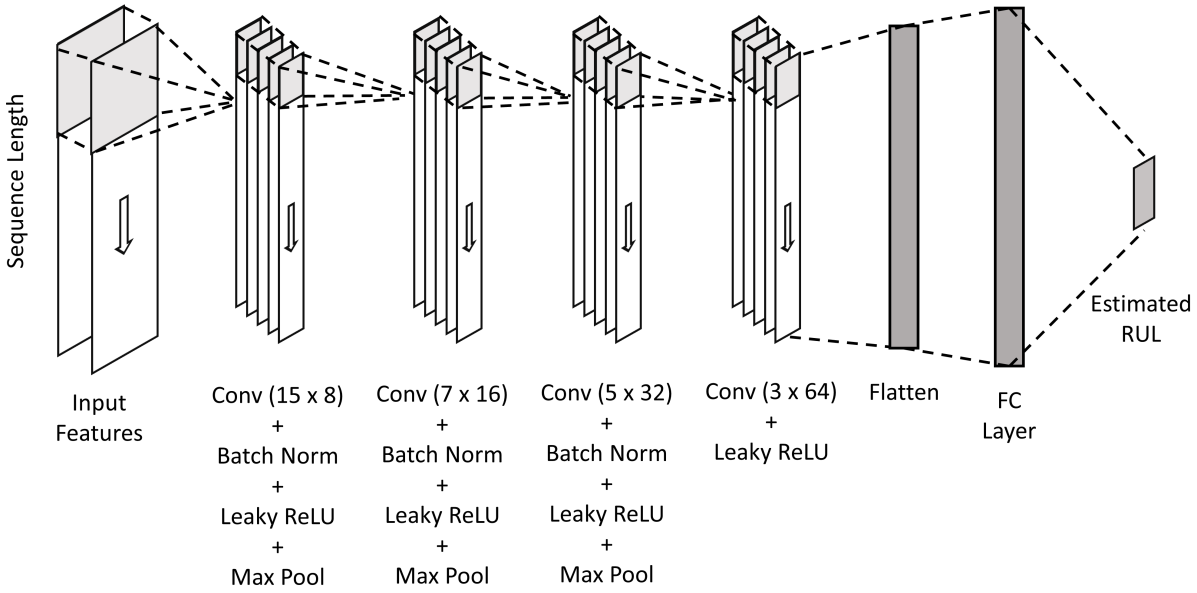


Figure III.5: Proposed Vanilla CNN-1D Architecture for Operating Condition 1&2

$=\{1, 3, 5, 7, 11, 15, 17\}$. The best performing RUL estimation architecture in illustrated in Fig. III.5 for Bearing data set for operating condition 1 & 2, which includes multiple CNN-1D layers and fully connected layer at the end. The notation $Conv$ (15×8) in Fig. III.5 should be inferred as the $(Kernel-size \times Number-of-Channels)$. The input features represent the sequences created as explained in the Sec. III.6.3. Each convolution operation is followed by Batch Normalization [69], the Leaky ReLU [70] activation function and the max-pooling operation. The architecture has a slight modification for operating condition-3, where the ReLU [71] activation function is used in place of the Leaky ReLU function because of overall better generalization performance. An initial learning rate of 0.001 was chosen with an exponential decay factor of 0.99 for every training epoch. The sequence length was chosen as 2560, batch size of 32, and Adam [72] optimizer was chosen for updating the parameters of the model.

III.6.4 Result Comparison with Generalized Dilation

To check the effectiveness of generalized dilation, we replaced the first layer of the architecture shown in Fig. III.5 with generalized dilation layer. All the other network parameters were kept the same as the vanilla CNN architecture. Only the receptive field is set to 29, whereas the kernel size of generalized dilation layer is set to 15. The resulting architecture is illustrated in Fig. III.6. The barrier function as shown in Eq. (III.13) and related parameters for each operating condition have been summarized in the Table III.1.

For qualitative analysis, the prognostic performance of proposed GDCNN architecture on 3 samples of the test bearing data set is presented in Fig. III.7. It can be seen that the model is able to capture the degradation trend by predicting the RUL close to the actual RUL. The prediction performance is especially high on the latter part of bearing life-span compared to the earlier part, which is when bearing is about to fail. The accurate

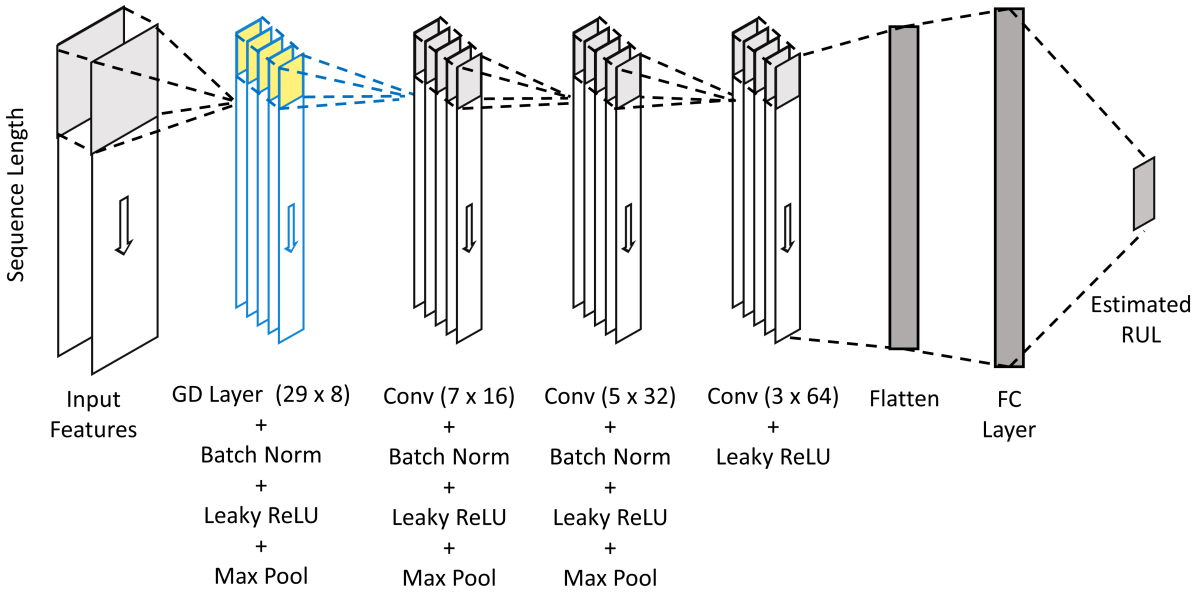


Figure III.6: Proposed GDCNN-1D Architecture for Operating Condition 1&2

Table III.1: Bearing data set : Barrier function parameters

Operating Condition	**Barrier Function**	α_1	α_2	α_3
1	b_r	0.5	0.1	-1
2	b_r	0.5	1	-1
3	b_r	1	0.1	-1

Table III.2: Performance of proposed architecture on Bearing data set

Bearing Name	**Actual RUL (s)**	**Predicted RUL (s)**	**%Er**	**Score**
Bearing 1-3	5730	4086.31	28.68	
Bearing 1-4	339	174	48.67	
Bearing 1-5	1610	3004.42	-86.61	
Bearing 1-6	1460	1458.76	0.084	
Bearing 1-7	7570	2159.85	71.46	
Bearing 2-3	7530	6737.35	10.53	0.38
Bearing 2-4	1390	4245.71	-205.4	
Bearing 2-5	3090	3084.25	0.18	
Bearing 2-6	1290	2478.83	-92.16	
Bearing 2-7	580	4411.95	-660.68	
Bearing 3-3	820	783	4.51	

estimation of RUL here is more valued compared to an initial healthy state. It can be also observed that for bearing 1_4, the estimated RUL has a sudden drop after a point of time. This phenomenon can be explained by the occurrence of an abrupt fault at that point of time in the system which affected the life of bearing.

The length of the sequence and step-size used to create the sequence is an important hyper-parameter for the proposed model's prognostic performance. If the sequence length is too small, then the model will not be able to capture all the useful information as the historical data is too less. On the other hand, large sequence length helps to capture degradation trends over a long time period. However, too large sequence length will increase the network complexity and the training time. It may also lead to over-fitting since the number of parameters in the hidden layers increases as the the sequence length increases. Therefore, to better analyse this effect, the impact of sequence length on the loss for the test dataset and the training time for operating condition-1 is presented in Fig. III.8. The RMSLE value is the sum of the final loss values of all the test bearings in operating condition-1. A similar trend in results was also observed for the other operating conditions. With the increase in sequence length from 512 to 3072, the training times first increase slowly, but a stark increase is observable for sequence length of 3072. As for the losses, the minimum loss occurs for sequence length of 2560 which is proposed for all CNN-1D architectures for this dataset. It is also evident that increasing sequence length beyond 2560 results in over-fitting where the test loss increases.

To visualize the training of the proposed masking matrices in a clear way, probability density plots of the initial and learned masking parameters for operating condition 1 with the barrier function methodology is presented in the Fig. III.9. The final learned distributions can be attributed to the specific configuration of barrier functions. It can be inferred as the optimum solution for a chosen set of hyper-parameters α. Similar behaviour was observed also for the other operating conditions. It can also be observed from Fig. III.9 that with barrier function training methodology, the masking parameters are not entirely binary. To achieve the goal of binary parameters, the top-K sampling approach has to be used which will be covered in the next section.

The predicted RUL and corresponding score for all the 11 bearings of test data set in

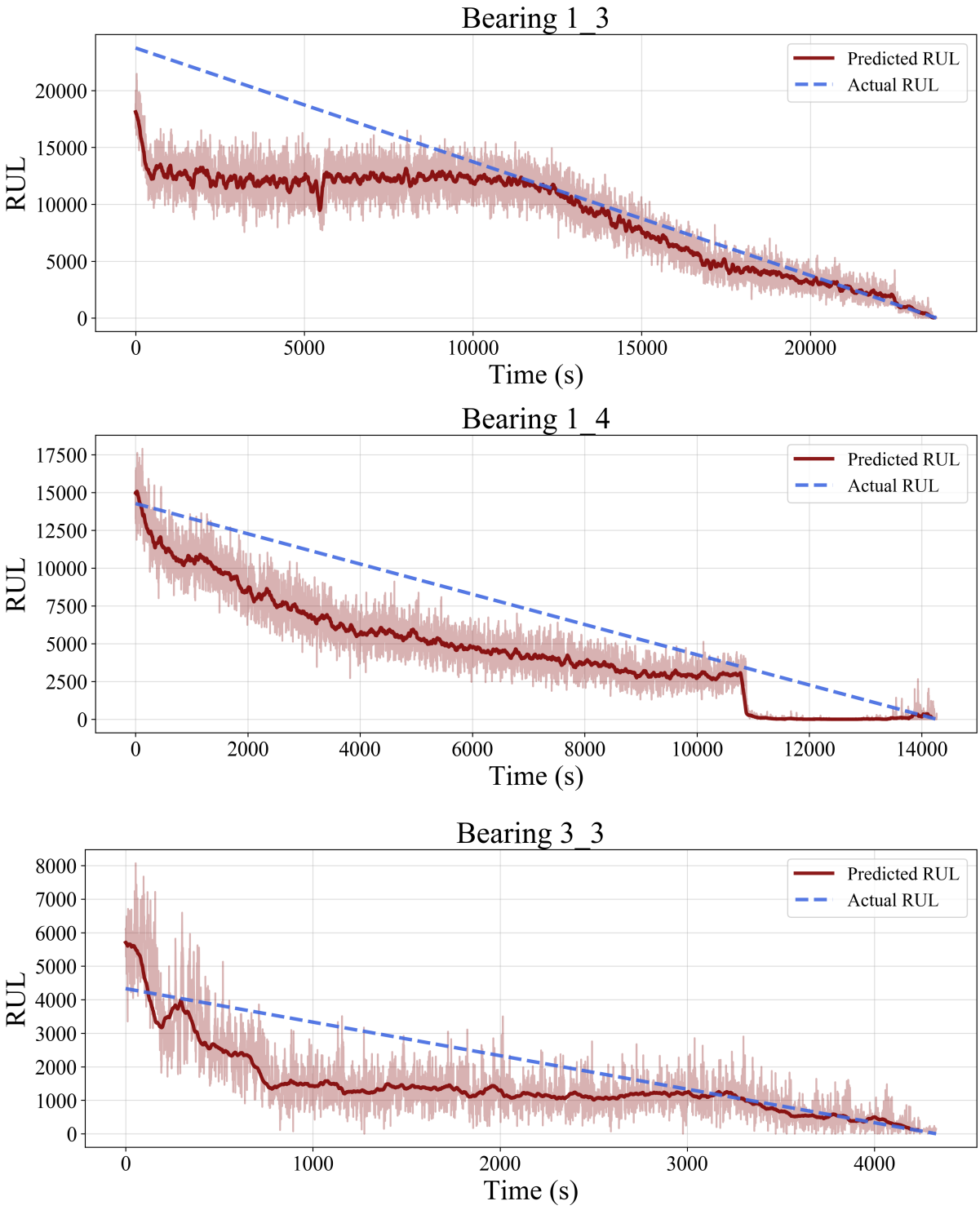


Figure III.7: Qualitative Generalization Performance on the Bearing data set

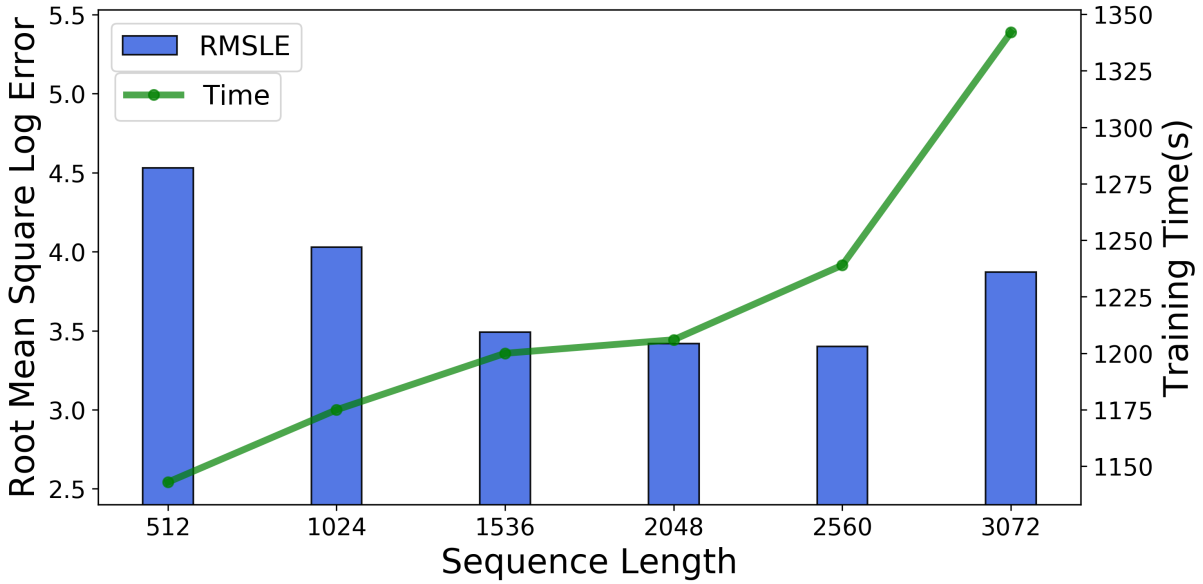


Figure III.8: Effect of sequence length on prognostics performance and training time of operating condition-1 test dataset

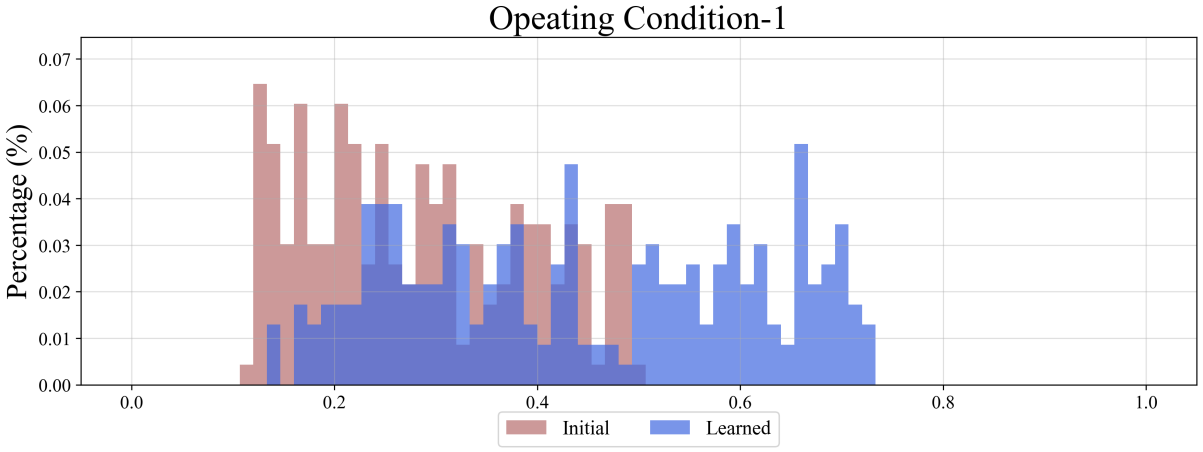


Figure III.9: GDCNN-1D - Distribution of initial and learned masking parameters ($\sigma(\Psi)$)

Table III.3: Performance comparison of the proposed architectures and related papers on Bearing data set

Method	Score
RNN based health indicator [23]	0.25
Approach based on non-trendability behavior [73]	0.28
Vibration frequency signature anomaly detection and survival time ratio [20]	0.30
Combinatorial feature extraction method [74]	0.35
Porposed Vanilla CNN	**0.35**
CNN + LSTM [42]	038
Proposed GDCNN	**0.38**

summarized in the Table III.2. To present a robust set of results, we ran the same configuration 6 times and presented the average predictions from these configurations. The comprehensive results of the prognostic performance of proposed architecture for Bearing data set in comparison with the state of the art results obtained by other published works are presented in the Table III.3. It should be noted that the score of both the proposed CNN-1D architectures outperforms several already published works. Moreover, GD-CNN outperforms Vanilla CNN-1D, which establishes the superiority of the proposed methodology. To the author's knowledge, there is no published architecture for the Bearing Dataset which uses only CNN for such performance, where only it demonstrated results of the same level with LSTM and CNN architecture together.

III.7 Experimental Results on Aircraft Engine Dataset

In this section, we report experimental results on the aircraft engine data set, and thoroughly discuss the different architectural effects with the proposed dilation networks.

III.7.1 Dataset Description

As a second benchmark, we use the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) dataset [14]. It contains information from 21 sensors in 3 operational settings which include different types of engine degradation processes. The C-MAPSS data set includes 4 sub-datasets namely FD001, FD002, FD003, FD004. Each sub data set is further divided into train and test data set. The four datasets each include several degradation engines split into training and testing data. Table III.4 summarizes the structure of the complete data set.

Moreover, run-to-failure information from multiple engines collected under various operating conditions and fault modes are included. The engines are assumed to start with various degrees of initial wear but are considered healthy at the start of each record. The engines then begin to deteriorate until they cannot perform their function such that they are considered unhealthy. Note that unlike the training datasets, the testing datasets contain temporal data that terminates before a system failure occurs. FD001 and FD003 contain one operating condition while FD002 and FD004 contain up to six operating

Table III.4: C-MAPSS Dataset Information [75]

Dataset Description	C-MAPSS			
	FD001	**FD002**	**FD003**	**FD004**
Engine Trajectories for Training	100	260	100	249
Engine Trajectories for Testing	100	259	100	248
Operating Conditions	1	6	1	6
Fault Modes	1	1	2	2

conditions or regimes which depend on different combinations of altitude $(0 - 42000\ ft)$, throttle resolver angle $(20° - 100°)$ and Mach $(0 - 0.84)$. Similarly, FD001 and FD002 contain one fault mode (HPC degradation) and FD001 and FD002 contain two fault modes (HPC degradation and Fan degradation).

III.7.2 RUL Evaluation Metric

The RMSE loss function for training the proposed architecture on the CMAPSS dataset

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(RUL_i) - (\widehat{RUL}_i)^2}. \tag{III.23}$$

The CMAPSS dataset is evaluated according to the scoring function

$$S = \begin{cases} \sum_{n=1}^{N} e^{-\frac{c_i}{a_1}} - 1, & \text{if } c_i < 0 \\ \sum_{n=1}^{N} e^{\frac{c_i}{a_2}} - 1, & \text{if } c_i \geq 0. \end{cases} \tag{III.24}$$

where $a_1 = 13$, $a_2 = 10$ and $c_i = \widehat{\text{RUL}}_i - \text{RUL}_i$, i.e. the difference between predicted and observed RUL values. Note that both metrics penalize late predictions stronger as early predictions are preferable so that the component can be removed or replaced without causing severe damage to the rest of the process.

III.7.3 Training Settings

For the C-MAPSS dataset, we present results with two convolutional architectures namely, the standard 1-D CNN architecture and a shared kernel CNN-1D architecture. In the shared kernel approach, each convolution kernel is shared for all the input sensor channels. This helps in reducing the convolution parameters when the number of input sensor channels is high, which is the case in the C-MAPSS dataset. The architectural details for the proposed model are given in section III.7.3.

Data-preprocessing

In order to attain the shared kernel CNN 1-D approach, the inputs are prepared in the shape of 2D feature map, $T \times m$ where T is the temporal dimension and m is the feature dimension. The sequences from sensor signals are generated based on the same apporach as in Sec. III.6.3. A single sequence length can only contain data of a single-engine.

Table III.5: Selected features for C-MPASS data set

Sub-data set	**Selected Features**
FD001 FD004 FD003	S2, S3, S4, S7, S8, S9, S11, S12, S13, S14, S15, S17, S20, S21
FD002	S2, S3, S4, S7, S8, S9, S11, S12, S13, S14, S15, S17, S20, S21, History of Operating Regimes

We use the same piece-wise linear degradation as target for the C-MAPSS data set as proposed in [76], with the knee point $R_{early} = 125$. Since the sub-dataset FD002 and FD004 have 6 different operating regimes, consecutive data points in these datasets can belong to different operating regimes. Therefore, to extract more meaningful information, the sensor signals are normalized per operating regime similar to the approach in [77]. Using K-means clustering we identified the 6 operating regimes and normalized them individually per the operating regime as defined in Eq. (III.21). The scaled operating regime clusters are concatenated to form the final time-series.

In addition to the dataset scaling, the different operating regimes have a significant impact on the degradation of the engine. Thus, it is useful to give information about operating mode history to the model. This can be achieved by adding operational mode history as a feature to the model as proposed in [78]. So the data sets which have multiple operating modes, an additional 6 columns as features were added where each feature represents the number of cycles spent in the respective operating regime since the start of the time-series. Moreover, out of the 21 sensors, not all of them provide useful information about system degradation pattern. Hence, relevant features were selected for the sub-datasets based on previous works [76, 77, 78]. The selected features are listed in the Table III.5.

Architecture Definition

We propose three different network architectures in this section, namely

- Shared Kernel CNN-1D
- Shared Kernel GDCNN-1D
- GDCNN-1D

The motivation for proposing a shared filter architecture for the C-MAPSS dataset is that the feature space is considerably larger than the bearing dataset. The dataset consists of 21 features ranging over 6 separate operating regimes, and for sub-datasets FD002 the history of the operating regimes was added to the feature space. The standard CNN-1D architecture was not efficient while working with such large feature spaces and the prediction results were not satisfactory. Therefore, we compare the results from the shared kernel approach with only the GDCNN-1D architecture. The proposed shared filter CNN-1D network architecture for the C-MAPSS dataset is shown in Fig. III.10

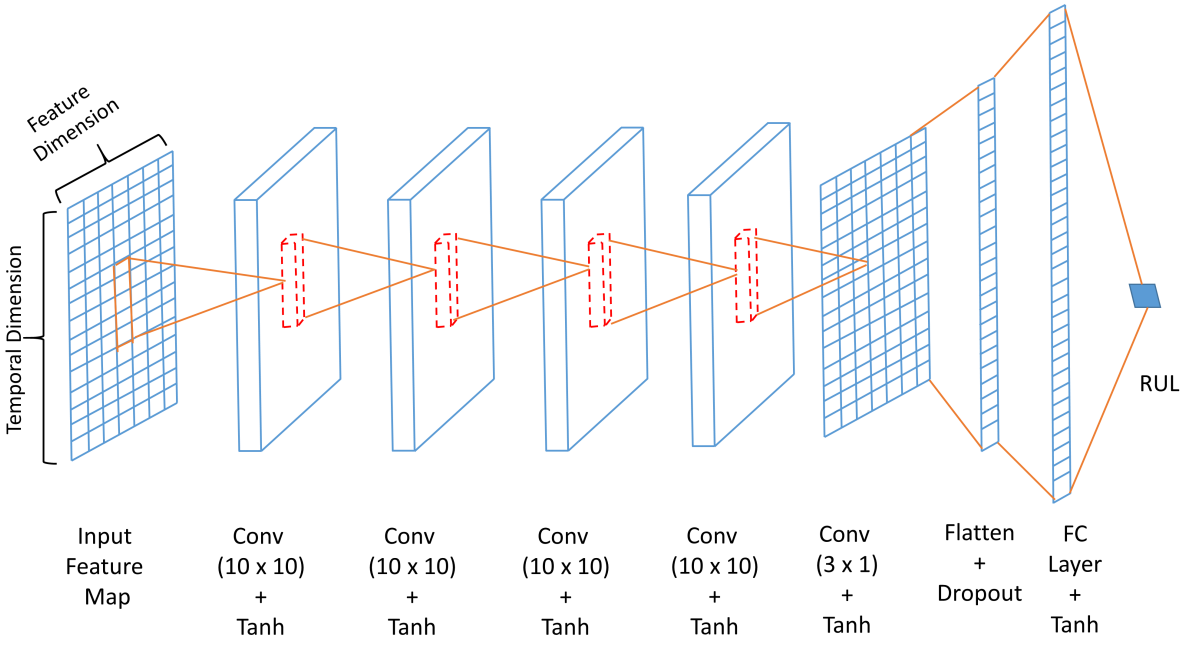


Figure III.10: Proposed Shared kernel CNN-1D Architecture

consisting of 5 convolution layers and 1 fully connected layer. It should be noted that the convolution operation is executed only in the temporal dimension of each feature individually.

The notation $Conv(10 \times 10)$ should be inferred as $Conv(Kernel\ Size \times Number\ of\ Kernels)$. The sequence length of the T for the sub-datasets FD001 to FD004 was chosen as 30, 20, 30, 15 respectively. A batch size of 512 and 250 training epochs were chosen with the Adam optimizer. The learning rate is set at 0.001 initially up to 200 epochs and then reduced to 0.0001 for the last 50 epochs to get stable convergence. The loss function is the standard root mean square error (RMSE).

Taking the architecture as shown in Fig. III.10 as the baseline, we propose the shared kernel GDCNN-1D architecture, shown in Fig. III.11, where the first layer convolution layer was replaced with the GD layer. In order to be consistent with the previous architecture, we set the kernel size for a GD layer $n_r = 10$ and set the receptive field size $n_d = 19$.

Lastly, the GDCNN-1D architecture is proposed without the shared kernel approach, resulting in a bigger kernel size in the first layer. However, in this approach, the generalized dilation approach has the possibility to select the relevant features for the RUL estimation task. Since there are many more features in the C-MAPSS dataset in comparison to the bearing dataset, this approach allows for not only dilation in the temporal dimension, but also in the feature dimension. This is especially important in the multivariate setting that deep neural network excel in.

Training with Barrier Function and Top-K Sampling

Along with the three different architectures proposed, we implement the two different training methods as presented in Sec. III.5.1 and Sec. III.5.2 , for incorporating the con-

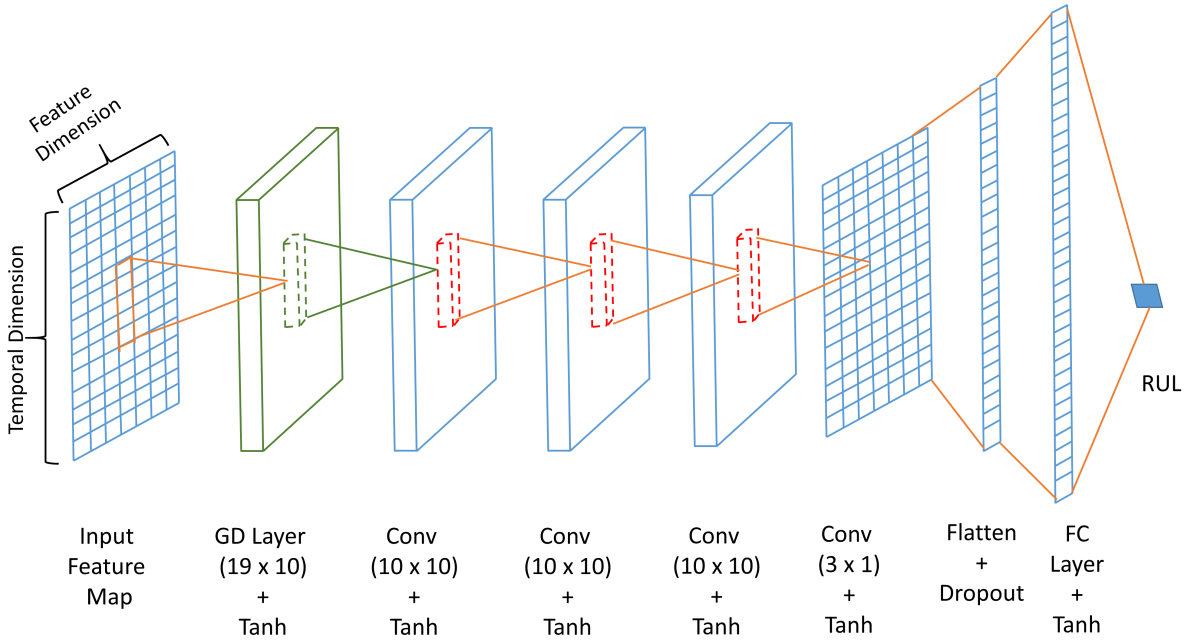


Figure III.11: Proposed Shared kernel GDCNN-1D Architecture

Table III.6: Barrier Function Parameters for C-MAPSS

Sub-data set	**Barrier Function**	α_1	α_2	α_3
FD001	b_r	0.5	1.5	-1
FD002	b_r	0.5	1.5	-1
FD003	b_c	1	0.1	-1
FD004	b_r	0.5	1.5	-1

straints in the GD training. The barrier function and related parameters for each sub-data set have been summarized in the Table III.6. It must be noted that the Top-K Sampling approach enables integer binary values in the masking matrices Ψ, in contrast to the barrier function approach.

III.7.4 Results and Comparison Study

For qualitative analysis of the shared kernel architectures, the prognostic performance for the three architectures on the test engine unit 24 from the sub-data FD001 has been shown in Fig. III.12a, Fig. III.12b and Fig. III.12c respectively. The actual RUL and the predicted RUL of one engine trajectory from the test engine has been presented. Notice that the RUL estimations of the last parts of the engine unit life-time are not shown. This is because, in order to validate the prognostic performance on the testing dataset, the last parts of the sensor measurements are not provided. The actual RUL values for the last recorded cycles are given in the dataset, and the corresponding RUL labels can be obtained accordingly. It can be observed that in the early periods in all the cases, the proposed model manages to estimate the RUL values as close to the constant R_{early}. Subsequently,

Table III.7: Performance comparison of the proposed shared kernel architectures on C-MAPSS dataset

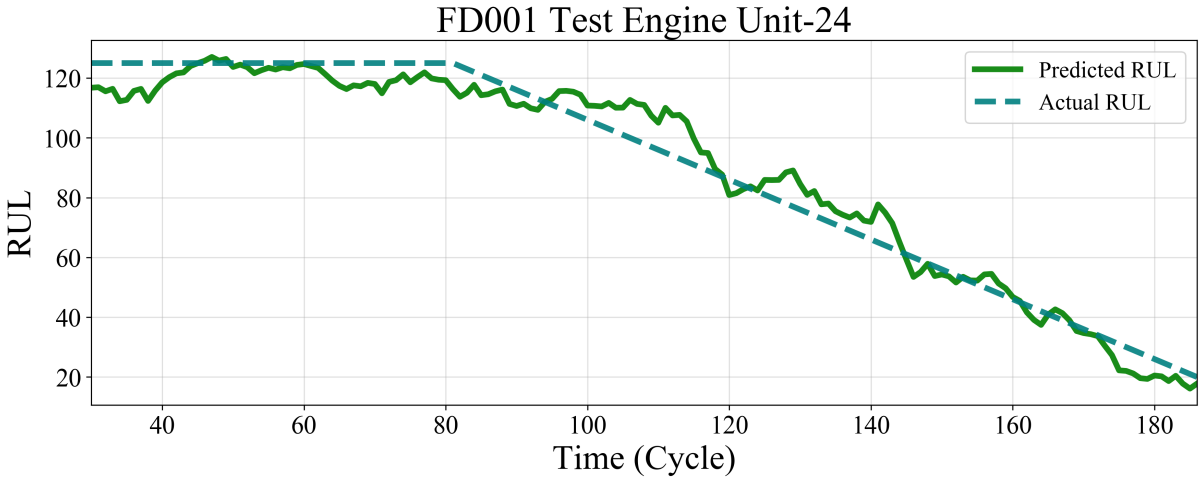
Method	**FD001**	**FD002**	**FD003**	**FD004**
CNN-1D	317.7	11053	336.7	8122
GDCNN-1D with Barrier	316.8	10273	304.4	7250
GDCNN-1D with Top-K	274.6	10256.6	285.1	7213.8

the estimations are decreasing linearly with time until the end of the available testing samples. Though some noticeable error exists between the predictions and the actual RUL values, in general, the prognostic accuracy is high especially when the engine units are close to failure for the proposed GDCNN-1d architectures. Especially for the case with the Top-K sampling, the estimation of the RUL at the end of the engine life is very close to the actual RUL. This can also be seen from the scores on each of the sub-dataset using the shared kernel approach in Table III.7. It is clear from the analysis that the GDCNN-1D approach performs better than the standard CNN-1D approach since the lower the score, the better it is. Furthermore, the Top-K Sampling results in a drastic improvement in prognostic performance where the Top-100 elements from the receptive field have been sampled as 1 and the rest are sampled as 0. This hard sampling helps with the integer binary values of the masking parameters. The distribution of initial and learned masking parameters within the range of (0, 1) is presented in Fig. III.13a and Fig. III.13b for the barrier function approach and Top-K sampling approach respectively. The motivation for showing this plot is to compare the two training methodologies' of soft and hard masking capabilities,i.e. Training with Barrier Function and Training with Top-K Sampling. The figures illustrate the hard constraints being achieved by the Top-K sampling approach, in contrast to the soft constraints of the Barrier function. The authors presume that this hard constraint from the Top-K approach is the reason behind its superior performance.

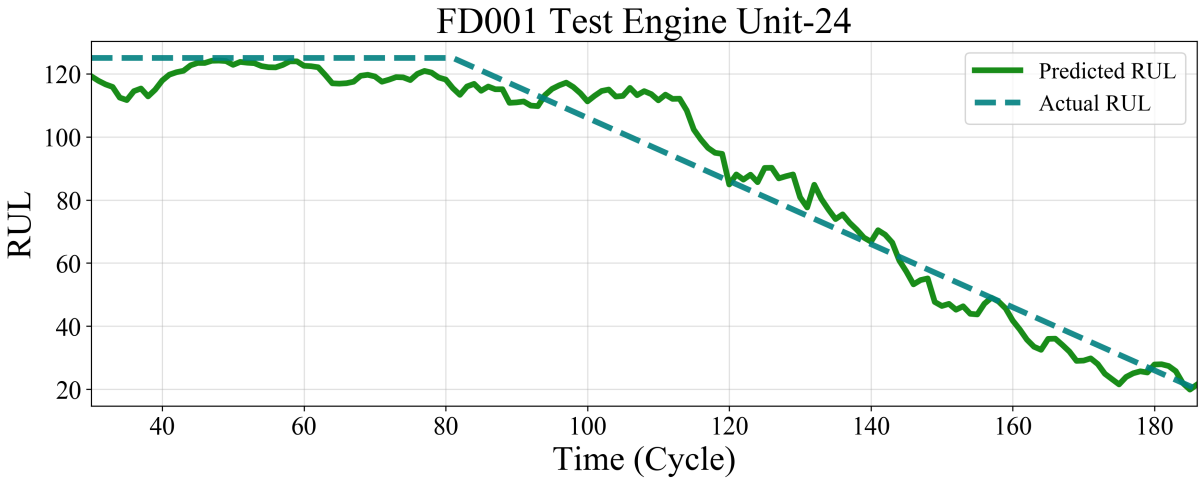
Therefore, the GDCNN-1D experiments were performed only with the Top-K sampling approach only. The prognostic performance from the proposed GDCNN-1D is illustrated in Fig. III.14, Fig. III.15, Fig. III.16 and Fig. III.17. Evidently the deviation between the predicted and actual RUL has reduced more notably. This is also evident from the final scores achieved by the proposed architectures for all the sub-datasets in Table III.8. It should be noted that the score of all proposed architectures outperforms several published works. It is evident that the GDCNN approaches outperform the standard CNN approaches and furthermore, Top-K sampling approach for GDCNN returns better performance than its barrier function counterpart.

III.8 Conclusions

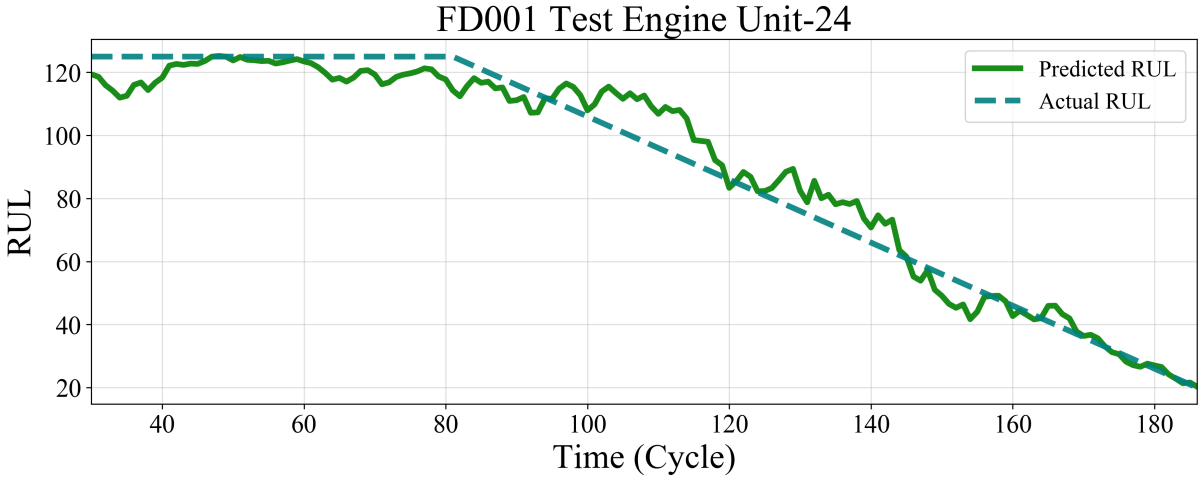
In this paper we propose a novel approach for multivariate time series data analysis based on a novel end-to-end learnable dilated convolutional neural networks. We generalize the vanilla dilated CNNs with particular emphasis on the requirements of time series analysis which allows for the representation of arbitrary dilation patterns in a unified way. We propose two end-to-end training approaches for the proposed architecture utilizing the barrier function and Top-K sampling methodologies. The developed architectures are used



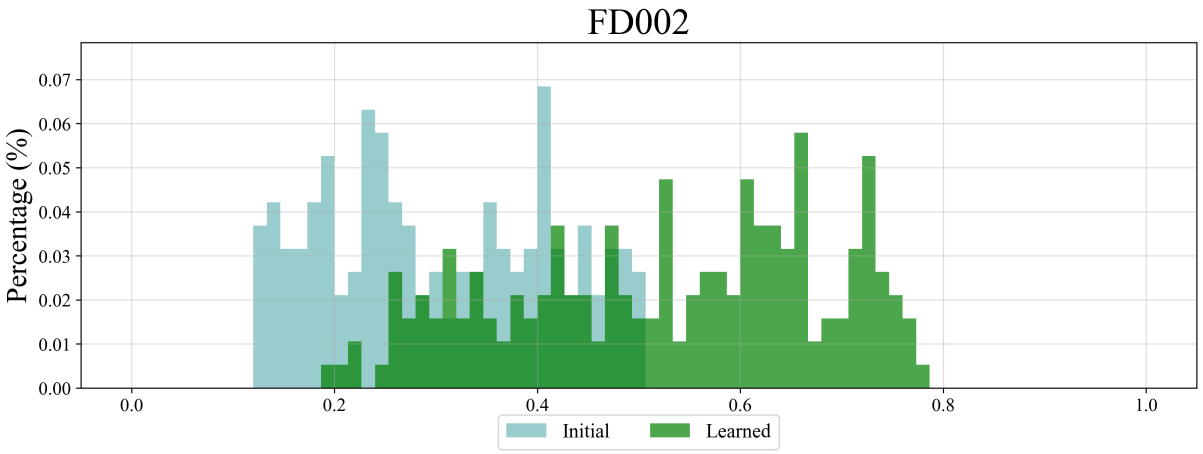
(a) Prognostic Performance from the Shared Kernel CNN-1D architecture on a test engine



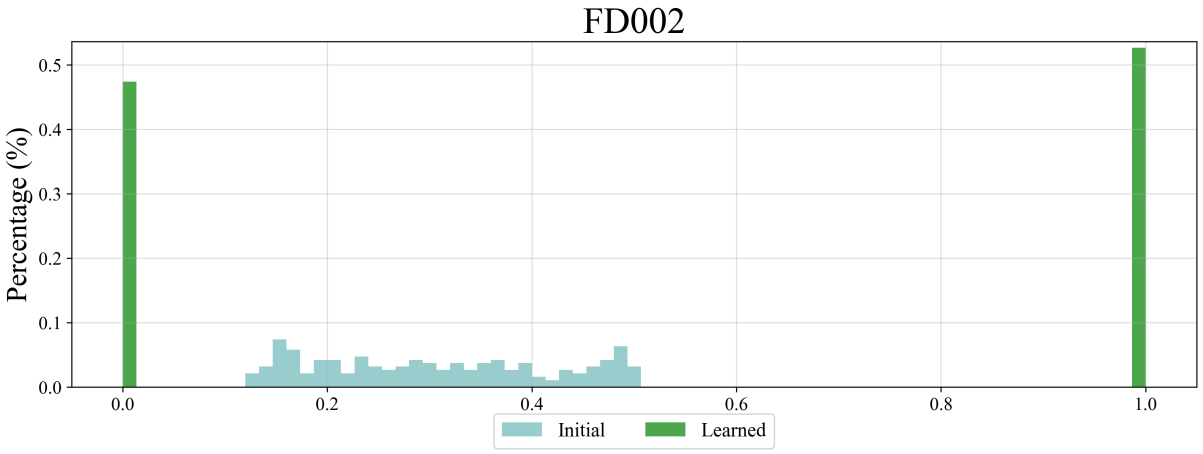
(b) Prognostic Performance from the Shared Kernel GDCNN-1D architecture with Barrier Function on a test engine



(c) Prognostic Performance from the Shared Kernel GDCNN-1D architecture with Top-K Sampling on a test engine



(a) Distribution of initial and learned masking parameters ($\sigma(\Psi)$) in the case of Barrier Function



(b) Distribution of initial and learned masking parameters ((Ψ)) in the case of Top-K Sampling

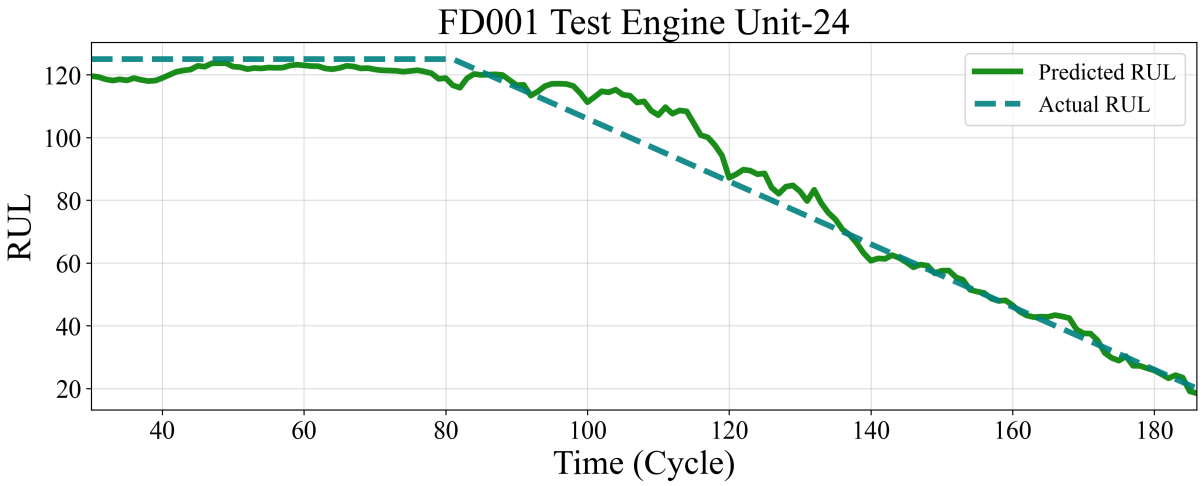


Figure III.14: Prognostic performance from the GDCNN-1D architecture with Top-K sampling on a test engine from FD001

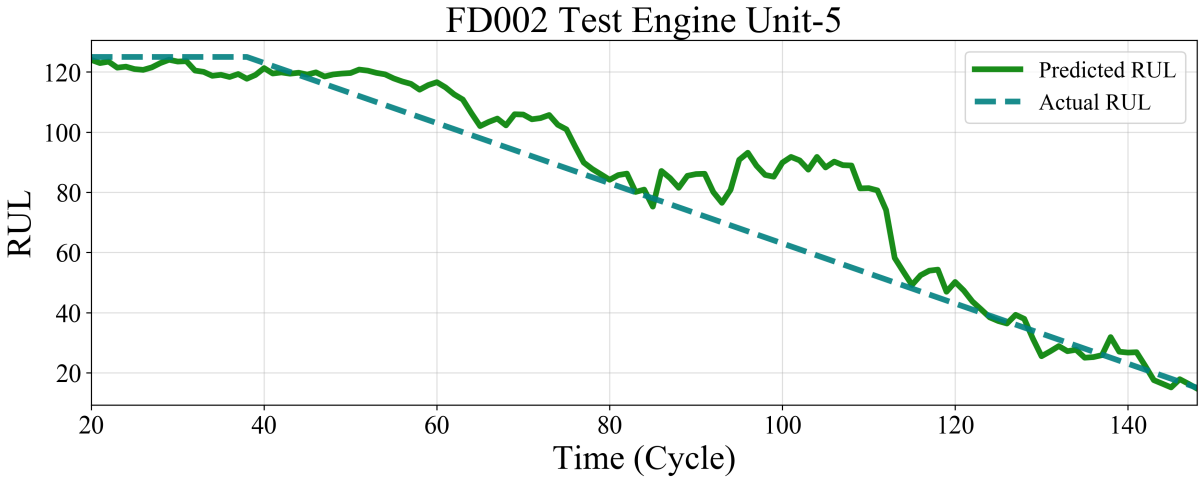


Figure III.15: Prognostic performance from the GDCNN-1D architecture with Top-K sampling on a test engine from FD002

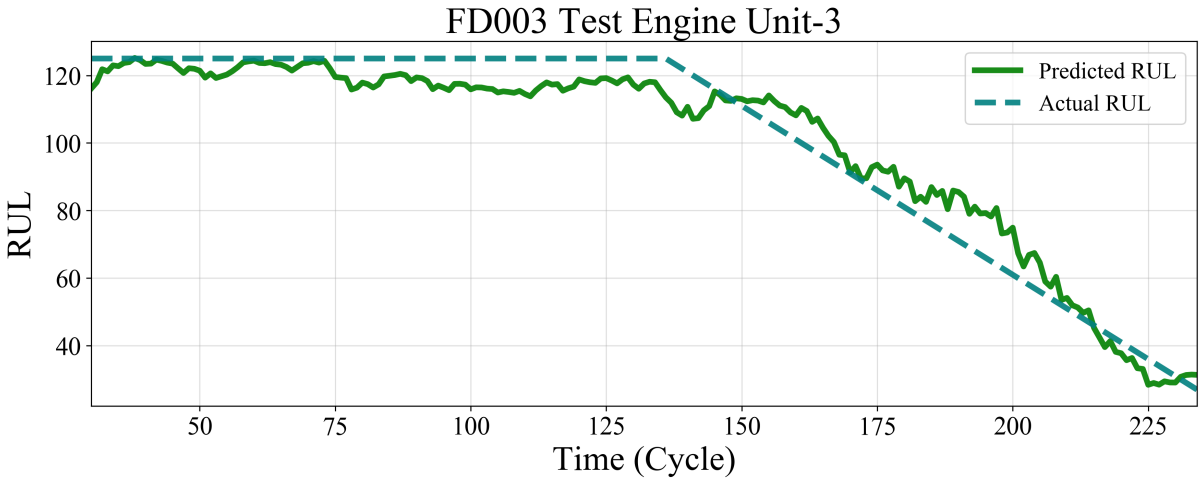


Figure III.16: Prognostic performance from the GDCNN-1D architecture with Top-K sampling on a test engine from FD003

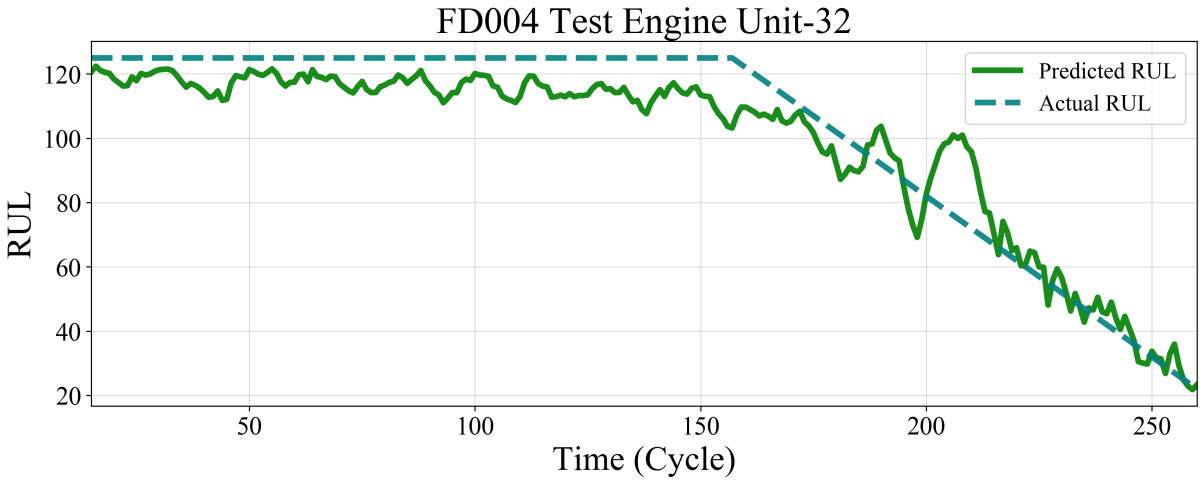


Figure III.17: Prognostic performance from the GDCNN-1D architecture with Top-K sampling on a test engine from FD004

for the remaining useful life estimation problem where the generalized dilation enhances the ability of convolutional neural networks to capture long term dependencies. The proposed methodology can learn the degradation trends over time which can be mapped to the current RUL of the equipment from the Bearing Dataset and the C-MAPSS Dataset. A comparison of the obtained results with the results from the literature shows the superior performance of the proposed approach.

In future work, we intend to explore alternative ways to tackle the binary parameter optimization problem. Particularly, probabilistic approaches like Gumbel-softmax. Furthermore, efforts to develop an attention-like dilation network, where receptive field size is equal to the size of the input sequence, will be made. This will allow the network to attend to the most relevant samples from the complete multivariate time series and not just in the receptive field. The proposed methodology will also be investigated on images in the future studies.

Table III.8: Performance comparison of the proposed architectures and related papers on C-MAPSS Dataset

Method	**FD001**	**FD002**	**FD003**	**FD004**
CNN [75]	1280	13500	1590	7880
CNN [79]	661	12643	1412	7482
LSTM [28]	388.68	10654	822.19	6370.6
LSTM [80]	338	4450	852	5550
CNN [81]	273.7	10412	284.1	12466
Shared Kernel CNN-1D	**317.7**	**11053**	**336.7**	**8122**
Shared Kernel GDCNN-1D with Barrier	**316.8**	**10273**	**304.4**	**7250**
Shared Kernel GDCNN-1D with Top-K	**274.6**	**10256.6**	**285.1**	**7213.8**
GDCNN-1D with Top-K	**267.8**	**10868.9**	**307.8**	**7111.1**

Bibliography

[1] Y. Jiang, S. Yin, and O. Kaynak, "Performance supervised plant-wide process monitoring in industry 4.0: A roadmap," *IEEE Open Journal of the Industrial Electronics Society*, vol. 2, pp. 21–35, 2021.

[2] S. Nandi, H. A. Toliyat, and X. Li, "Condition monitoring and fault diagnosis of electrical motors—a review," *IEEE Transactions on Energy Conversion*, vol. 20, no. 4, pp. 719–729, 2005.

[3] K. F. Martin, "A review by discussion of condition monitoring and fault diagnosis in machine tools," *International Journal of Machine Tools and Manufacture*, vol. 34, no. 4, pp. 527–551, 1994.

[4] R. Isermann, "Model-based fault-detection and diagnosis – status and applications," *Annual Reviews in Control*, vol. 29, no. 1, pp. 71–85, 2005.

[5] K. L. Tsui, N. Chen, Q. Zhou, Y. Hai, and W. Wang, "Prognostics and health management: A review on data driven approaches," *Mathematical Problems in Engineering*, vol. 2015, no. 6, pp. 1–17, 2015.

[6] A. K. Jardine, D. Lin, and D. Banjevic, "A review on machinery diagnostics and prognostics implementing condition-based maintenance," *Mechanical Systems and Signal Processing*, vol. 20, no. 7, pp. 1483–1510, 2006.

[7] Y. Jiang, S. Yin, J. Dong, and O. Kaynak, "A review on soft sensors for monitoring, control and optimization of industrial processes," *IEEE Sensors Journal*, p. 1, 2020.

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. Adaptive computation and machine learning, Cambridge, Massachusetts: The MIT Press, 2016.

[9] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, pp. 396–404, Morgan Kaufmann, 1990.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014.

[12] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[13] P. Nectoux, R. Gouriveau, K. Medjaher, E. Ramasso, and et. al, "Pronostia: An experimental platform for bearings accelerated life test," in *IEEE International Conference on Prognostics and Health Management, Denver, CO, USA*, 2012.

[14] 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*, pp. 1–9, IEEE, 06/10/2008 - 09/10/2008.

[15] R. Khelif, B. Chebel-Morello, S. Malinowski, E. Laajili, F. Fnaiech, and N. Zerhouni, "Direct remaining useful life estimation based on support vector regression," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 3, pp. 2276–2285, 2017.

[16] K. Medjaher, D. A. Tobon-Mejia, and N. Zerhouni, "Remaining useful life estimation of critical components with application to bearings," *IEEE Transactions on Reliability*, vol. 61, no. 2, pp. 292–302, 2012.

[17] E. Ramasso and R. Gouriveau, "Remaining useful life estimation by classification of predictions based on a neuro-fuzzy system and theory of belief functions," *IEEE Transactions on Reliability*, vol. 63, no. 2, pp. 555–566, 2014.

[18] F. Di Maio, K. L. Tsui, and E. Zio, "Combining relevance vector machines and exponential regression for bearing residual life estimation," *Mechanical Systems and Signal Processing*, vol. 31, pp. 405–427, 2012.

[19] F. Ahmadzadeh and J. Lundberg, "Remaining useful life estimation: review," *International Journal of System Assurance Engineering and Management*, vol. 5, no. 4, pp. 461–474, 2014.

[20] E. Sutrisno, H. Oh, A. S. S. Vasan, and M. Pecht, "Estimation of remaining useful life of ball bearings using data driven methodologies," in *2012 IEEE Conference on Prognostics and Health Management*, pp. 1–7, IEEE, 18/06/2012 - 21/06/2012.

[21] R. K. Singleton, E. G. Strangas, and S. Aviyente, "Extended kalman filtering for remaining-useful-life estimation of bearings," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 3, pp. 1781–1790, 2015.

[22] J. Deutsch and D. He, "Using deep learning-based approach to predict remaining useful life of rotating components," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 1, pp. 11–20, 2018.

[23] L. Guo, N. Li, F. Jia, Y. Lei, and J. Lin, "A recurrent neural network based health indicator for remaining useful life prediction of bearings," *Neurocomputing*, vol. 240, pp. 98–109, 2017.

[24] B. Yang, R. Liu, and E. Zio, "Remaining useful life prediction based on a double-convolutional neural network architecture," *IEEE Transactions on Industrial Electronics*, vol. 66, no. 12, pp. 9521–9530, 2019.

[25] M. Xia, T. Li, T. Shu, J. Wan, C. W. de Silva, and Z. Wang, "A two-stage approach for the remaining useful life prediction of bearings using deep neural networks," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3703–3711, 2019.

[26] C. Zhang, P. Lim, A. K. Qin, and K. C. Tan, "Multiobjective deep belief networks ensemble for remaining useful life estimation in prognostics," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2306–2318, 2017.

[27] M. Yuan, Y. Wu, and L. Lin, "Fault diagnosis and remaining useful life estimation of aero engine using lstm neural network," in *2016 IEEE International Conference on Aircraft Utility Systems (AUS)*, pp. 135–140, IEEE, 10/10/2016 - 12/10/2016.

[28] C.-S. Hsu and J.-R. Jiang, "Remaining useful life estimation using long short-term memory deep learning," in *2018 IEEE International Conference on Applied System Invention (ICASI)*, pp. 58–61, IEEE, 2018.

[29] Y. Cheng, H. Zhu, J. Wu, and X. Shao, "Machine health monitoring using adaptive kernel spectral clustering and deep long short-term memory recurrent neural networks," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 987–997, 2019.

[30] T. Vishnu, P. Malhotra, L. Vig, and G. Shroff, "Deep ordinal regression for remaining useful life estimation from censored data," in *Joint Workshop on Deep Learning for Safety-Critical Applications in Engineering at ICML-AAMAS-IJCAI, Stockholm, Sweden*, pp. 13–19, 2018.

[31] A. Al-Dulaimi, S. Zabihi, A. Asif, and A. Mohammadi, "A multimodal and hybrid deep neural network model for remaining useful life estimation," *Computers in Industry*, vol. 108, pp. 186–196, 2019.

[32] A. Listou Ellefsen, E. Bjørlykhaug, V. Æsøy, S. Ushakov, and H. Zhang, "Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture," *Reliability Engineering & System Safety*, vol. 183, pp. 240–251, 2019.

[33] W. Yu, I. Y. Kim, and C. Mechefske, "An improved similarity-based prognostic algorithm for rul estimation using an rnn autoencoder scheme," *Reliability Engineering & System Safety*, vol. 199, p. 106926, 2020.

[34] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[35] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.

[36] R. Liu, B. Yang, and A. G. Hauptmann, "Simultaneous bearing fault recognition and remaining useful life prediction using joint-loss convolutional neural network," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 1, pp. 87–96, 2020.

[37] Z. Chen, K. Gryllias, and W. Li, "Intelligent fault diagnosis for rotary machinery using transferable convolutional neural network," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 1, pp. 339–349, 2020.

[38] X. Li, W. Zhang, and Q. Ding, "Deep learning-based remaining useful life estimation of bearings using multi-scale feature extraction," *Reliability Engineering & System Safety*, vol. 182, pp. 208–218, 2019.

[39] G. Hou, S. Xu, N. Zhou, L. Yang, Q. Fu, and A. D. Doulamis, "Remaining useful life estimation using deep convolutional generative adversarial networks based on an autoencoder scheme," *Computational Intelligence and Neuroscience*, vol. 2020, p. 9601389, 2020.

[40] T. S. Kim and S. Y. Sohn, "Multitask learning for health condition identification and remaining useful life prediction: deep convolutional neural network approach," *Journal of Intelligent Manufacturing*, 2020.

[41] W. Mao, J. He, J. Tang, and Y. Li, "Predicting remaining useful life of rolling bearings based on deep feature representation and long short-term memory neural network," *Advances in Mechanical Engineering*, vol. 10, no. 12, 2018.

[42] A. Z. Hinchi and M. Tkiouat, "Rolling element bearing remaining useful life estimation based on a convolutional long-short-term memory network," *Procedia Computer Science*, vol. 127, pp. 123–132, 2018.

[43] N. Gugulothu, T. Vishnu, P. Malhotra, L. Vig, P. Agarwal, and G. Shroff, "Predicting remaining useful life using time series embeddings based on recurrent neural networks," *International Journal of Prognostics and Health Management*, vol. 9, no. 1, 2018.

[44] G. E. Mena, D. Belanger, S. W. Linderman, and J. Snoek, "Learning latent permutations with gumbel-sinkhorn networks," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, OpenReview.net, 2018.

[45] C. Lemaire, A. Achkar, and P.-M. Jodoin, "Structured pruning of neural networks with budget-aware regularization," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9100–9108, 2018.

[46] W. Kool, H. van Hoof, and M. Welling, "Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement," in *International Conference on Machine Learning*, pp. 3499–3508, 2019.

[47] M. Holschneider, R. Kronland-Martinet, J. Morlet, and P. Tchamitchian, "A real-time algorithm for signal analysis with the help of the wavelet transform," in *Wavelets*, pp. 286–297, Springer, 1990.

[48] M. J. Shensa, "The discrete wavelet transform: wedding the a trous and mallat algorithms," *IEEE Transactions on Signal Processing*, vol. 40, no. 10, pp. 2464–2482, 1992.

[49] L.-C. Chen, Y. Yang, J. Wang, W. Xu, and A. L. Yuille, "Attention to scale: Scale-aware semantic image segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3640–3649, 2016.

[50] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, 2018.

[51] T. Sercu and V. Goel, "Dense prediction on sequences with time-dilated convolutions for speech recognition," 2016.

[52] A. Gupta and A. M. Rush, "Dilated convolutions for modeling long-distance genomic dependencies," *bioRxiv*, 2017.

[53] E. Strubell, P. Verga, D. Belanger, and A. McCallum, "Fast and accurate entity recognition with iterated dilated convolutions," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, (Copenhagen, Denmark), pp. 2670–2680, Association for Computational Linguistics, 2017.

[54] F. Yu, V. Koltun, and T. Funkhouser, "Dilated residual networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 472–480, 2017.

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

[56] Y. He, M. Keuper, B. Schiele, and M. Fritz, "Learning dilation factors for semantic segmentation of street scenes," in *German Conference on Pattern Recognition*, pp. 41–51, 2017.

[57] Y. Jeon and J. Kim, "Active convolution: Learning the shape of convolution for image classification," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1846–1854, 2017.

[58] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 764–773, 2017.

[59] X. Zhu, H. Hu, S. Lin, and J. Dai, "Deformable convnets v2: More deformable, better results," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9308–9316, 2019.

[60] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

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

[62] A. Trask, F. Hill, S. E. Reed, J. Rae, C. Dyer, and P. Blunsom, "Neural arithmetic logic units," in *Advances in Neural Information Processing Systems*, pp. 8035–8044, 2018.

[63] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[64] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, pp. 8024–8035, 2019.

[65] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[66] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[67] G. S. Chadha, A. Panambilly, A. Schwung, and S. X. Ding, "Bidirectional deep recurrent neural networks for process fault classification," *ISA transactions*, 2020.

[68] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[69] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 448–456, JMLR.org, 2015.

[70] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[71] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, and G. E. Hinton, "On rectified linear units for speech processing," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3517–3521, 2013.

[72] D. P. Kingma and J. L. Ba, "Adam: Amethod for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2014.

[73] S. Porotsky and Z. Bluvband, "Remaining useful life estimation for systems with non-trendability behaviour," *2012 IEEE Conference on Prognostics and Health Management*, 2012.

[74] S. Hong, Z. Zhou, E. Zio, and K. Hong, "Condition assessment for the performance degradation of bearing based on a combinatorial feature extraction method," *Digital Signal Processing*, vol. 27, pp. 159–166, 2014.

[75] S. B. Giduthuri, Z. Peilin, and L. Xiao-Li, "Deep convolutional neural network based regression approach for estimation of remaining useful life," 2016.

[76] F. O. Heimes, "Recurrent neural networks for remaining useful life estimation," in *2008 international conference on prognostics and health management*, 2008.

[77] O. Bektas, J. A. Jones, S. Sankararaman, I. Roychoudhury, and K. Goebel, "A neural network filtering approach for similarity-based remaining useful life estimation," *The International Journal of Advanced Manufacturing Technology*, vol. 101, no. 1-4, pp. 87–103, 2019.

[78] L. Peel, "Data driven prognostics using a kalman filter ensemble of neural network models," *International conference on prognostics and health management, 2008*, pp. 1–6, 2008.

[79] J.-R. Jiang and C.-K. Kuo, "Enhancing convolutional neural network deep learning for remaining useful life estimation in smart factory applications," *Proceedings of the 2017 IEEE International Conference on Information, Communication and Engineering (IEEE-ICICE 2017)*, pp. 120–123, 2017.

[80] S. Zheng, K. Ristovski, A. Farahat, and C. Gupta, "Long short-term memory network for remaining useful life estimation," in *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*, pp. 88–95, IEEE, 2017.

[81] X. Li, Q. Ding, and J.-Q. Sun, "Remaining useful life estimation in prognostics using deep convolution neural networks," *Reliability Engineering & System Safety*, vol. 172, pp. 1–11, 2018.

IV. Shared Temporal Attention Transformer for Remaining Useful Lifetime Estimation

Outline

Bibliographic Information

Gavneet Singh Chadha, Sayed Rafay Bin Shah, Andreas Schwung, Steven X. Ding (2022, July): Shared Temporal Attention Transformer for Remaining Useful Lifetime Estimation. In IEEE Access 10, pp. 74244–74258.
doi: 10.1109/ACCESS.2022.3187702.

Author's contribution

The author contributed in the conceptualization, methodology, in data curation, software development, validation, formal analysis, results investigation, writing the original draft, writing the review and editing and making visualisations.

Copyright Notice

IV.1 Abstract

This paper proposes a novel deep learning architecture for estimating the remaining useful lifetime (RUL) of industrial components, which solely relies on the recently developed transformer architectures. The RUL estimation resorts to analysing degradation patterns within multivariate time series signals. Hence, we propose a novel shared temporal attention block that allows detecting RUL patterns with the progress of time. Furthermore, we develop a split-feature attention block that enables attending to features from different sensor channels. The proposed shared temporal attention layer in the encoder fulfils the goal of attending to temporal degradation patterns in the individual sensor signals before creating a shared correlation across the feature range. We develop two transformer architectures that are specifically designed to operate with multivariate time series data based on these novel attention blocks. We apply the architectures to the well known C-MAPSS benchmark dataset and provide various hyperparameter studies to analyse their impact on the performance. In addition, we provide a thorough comparison with recently presented state-of-the-art approaches and show that the proposed transformer architectures outperform the existing methods by a considerable margin.

IV.2 Introduction

With recent advancements in industrial systems, the need for predictive maintenance solutions has increased proportionally. In the era of "Industry 4.0", equipment reliability is of paramount importance to keep up with the world's continuous demands and challenges. Maintenance of equipment after failure occurrence is gradually becoming outdated by Prognostics and Health Management (PHM) solutions [1, 2]. PHM involves using machine data to generate the health status of the machine, allowing the user to schedule maintenance or prepare in any way for a possible halt in operation. The PHM solutions result in various advantages such as inventory management, prediction-based planned maintenance, and a system's life cycle optimisation.

One of PHM's key methodologies is the estimation of the remaining useful lifetime (RUL) [3] of a machine using historical data. By predicting a machine's RUL, it is easy to determine its remaining cycles before the occurrence of failure. Recent development in deep learning methodologies [4] has provided more dynamic and efficient ways of performing RUL estimations. Contrary to model-based approaches, where domain knowledge of the inherent system is necessary, deep neural networks (DNN) analyse historical sensor data to learn abstract information for predicting RULs. Moreover, the non-linearity of deep neural networks enhances their ability to extract the complex degradation pattern from a machine's sensor readings. In previous works, different deep learning approaches have been presented, ranging from convolutional neural networks (CNN) [5] to recurrent neural networks (RNN) [6, 7], see [3, 1] for overviews. However, the increase of performance results in overly complex architectures with a high number of parameters. This increased complexity, however, requires enormous labelled datasets which are hard to obtain in practice.

This paper introduces the novel concept of predicting sequence RUL values instead of a single RUL for a specific cycle using transformer architectures [8]. We hypothesise that sequence RUL points can better map sudden degradation or anomaly from raw sensor data,

thus providing a better representation of a degradation pattern. On the contrary, averaging the time-series input into a single RUL data point over-generalises the degradation features. To allow for sequence RUL prediction, we develop two novel transformer architectures. A Shared Temporal Attention Transformer **(STAT)** and Feature-Represented Shared Temporal Attention Transformer **(FeaR-STAT)** are proposed to estimate accurate RULs by combining solely fully connected (FC) and self-attention layers without any additional convolutional or recurrent layer. The proposed models use a novel shared temporal self-attention technique to extract individual sensor degradation patterns over time for better abstraction. Additionally, the lack of recurrent and convolutional layers makes the models more robust, with fewer trainable parameters in each of them. We apply the approach to the remaining useful life prediction of C-MAPSS turbofan engines. We provide various hyperparameter studies and compare them with existing literature results where both architectures outperform all other approaches with a substantial margin. The contributions of the paper can be summarised as follows:

- We present a novel sequence-to-sequence neural network approach for RUL estimation based on transformer architectures that provide a better representation of degradation patterns than a point RUL estimator.
- We present two transformer architectures specifically designed for analysing a general multivariate time series dataset. The proposed transformer architectures consist of a Shared Temporal Attention methodology along with Split-Temporal Multi-Head attention and Split-Feature Multi-Head attention blocks to enforce a higher emphasis on the sequence of the encoder input signal, thus enabling the encoder to correlate the degradation patterns of the machine in multiple instances.
- We apply the approach to the well known C-MAPSS dataset and report a new state-of-the-art result with much simpler and lightweight architectures.

The paper is organized as follows. Sec.IV.3 discusses the previous works related to our approach. Sec.IV.4 presents the basics of transformer architecture. Sec.IV.5 present the novel transformer architectures for the sequence-to-sequence RUL estimation. In Sec.IV.6, we provide results on the C-MAPSS data set. Sec.IV.7 concludes the paper.

IV.3 Related Work

We discuss the related literature for RUL estimation as well as transformer architecture in the sequel.

Deep Learning based RUL estimation: We concentrate on the recently prevalent deep learning approaches for RUL estimation and refer to [2, 1] for overviews on classical methods. Notably, two types of DNN have been used in RUL, namely RNNs and CNNs.

A deep-stacked LSTM network followed by multiple FC layers is proposed in [9] while a stacked bidirectional LSTM network with additional FC layers is proposed in [10] for RUL prediction of C-MAPSS dataset. Zhao et al. [11] constructed trend features before feeding them to a stacked LSTM network for predicting RULs. The architecture proposed in [12] introduces an unsupervised pre-training stage using a Restricted Boltzmann machine (RBM) to extract complex raw input features combined with Genetic Algorithm approach

for hyperparameter tuning. After pre-training, supervised training is performed of the RBM-LSTM architecture for RUL prediction. To enhance the learning ability of recurrent encoder-decoder layers, Bahdanau et al. [13] and Luong et al. in [14] proposed attention mechanisms that allow each decoder state to attend to all the encoder hidden states before generating the next output. Such an attention-based LSTM network is proposed in [15], where handcrafted features are extracted, concatenated with the LSTM output and fed to a regression layer to predict RULs.

CNNs were first used for PHM and RUL prediction tasks in [16] which serves as the base for future implementations of CNN-based models. The deep CNN network proposed in [17] performs convolution operation with kernels of unit width. This unit width kernel allows for sharing kernel weights across raw sensors and enhancing the network's ability to learn abstract feature information. An attention-based CNN approach is proposed in [18] where the CNN filters extract features across multiple temporal axes, which are further analyzed by the attention layer before generating the RUL. The attention mechanism in [18] replaces the Softmax activation with a Sigmoid activation, intending to use additional multivariate features for estimating RUL. The attention-based stacked LSTM RUL predictor proposed in [19] implements the global Luong concatenation method to calculate attention alignment scores to predict RULs for the C-MAPSS dataset. An attention-based deep CNN-LSTM architecture is proposed in [20], where a CNN is used for raw input feature extraction for the following stacked LSTM network. The LSTM output from all timesteps is attended with respect to the last hidden state via an attention layer to generate RUL of a rotatory machine finally. Raw sensor data is pre-processed to generate a 1D-health indicator matrix passed to a hybrid CNN-LSTM-NN RUL predictor in [21]. The extracted spatial and temporal features from CNN and LSTM networks respectively are fused and passed to another CNN layer in [22] for predicting RULs of C-MAPSS sub-datasets FD001 and FD003. Liu et al. [23] proposed an encoder-decoder model for RUL prediction, where the encoder is made up of stacked BiLSTM layers, followed by multiple CNNs with intermediate pooling layers and the decoder is a network of three fully connected layers. Another hybrid network of parallel CNN and LSTM paths is proposed in [24], to reduce the influence of CNN extracted features on series-connected LSTMs. However, the model includes an additional LSTM network that processes the previous CNN and LSTM paths' fused outputs and predicts RUL values. In a later work in [25], Liu et al. utilises a self-attention mechanism for feature extraction. The feature-attended output is fed to a Bidirectional Gated Recurrent Unit (BiGRU)-CNN encoder, followed by a decoder with flattening and fully connected layers. Notably, the most successful architectures require many learnable parameters, making their application difficult for small RUL datasets.

Transformer Architectures: The transformer architectures date back to Vaswani et al. [8], which proposed the transformer as a combination of fully connected layers with a new multi-head self-attention mechanism in both the encoder-decoder layers. Since then, several extensions have been developed [26, 27, 28] with primary applications in NLP and image recognition. The transformer model has not yet been widely tested for PHM and RUL prediction. Very recently, a gated CNN layer for feature extraction on top of the transformer encoder layer has been proposed [29]. The model architecture replaces the decoder with a fully connected regression layer with Sigmoid activation.

To summarize, the proposed transformer architectures distinguish themselves from

existing work in the following aspects:

- The existing multi-head self-attention methodologies do not split the multivariate time-series signal when performing the multi-head self-attention. In this study, the multivariate time-series signal is segmented into univariate signals in the Shared Temporal Attention Block wherein the weights for each input feature are shared. Additionally, the Split-Temporal Multi-Head Attention splits the multivariate time-series along the time dimension to form a number of temporal heads.
- Similarly, the Split-Feature Multi-Head Attention creates feature heads by splitting the input features into multiple smaller heads.
- The proposed architectures have significantly fewer amount of parameters because of the shared weights in the temporal attention block for all the input features.
- This reduction in parameters consequently resulted in a drastic improvement in the performance of the models on a benchmark RUL estimation dataset.

IV.4 Basics of Transformer Architecture

The transformer architecture proposed in [8] deploys a self-attention mechanism where it computes the relevance between each input with respect to the other inputs. The architecture uses parallel inputs instead of the sequential input method employed in RNN/LSTM structures to perform the intra-attention operation. Additionally, by adopting the parallel input technique, the transformer successfully avoids the vanishing and exploding gradient problems.

The transformer architecture proposed in [8] is illustrated in Fig. IV.1. The input data in the encoder goes through a self-attention layer involving multiple heads, where all the other input data points are taken into context while encoding a certain data point. A residual connection and normalization block follow this, then an independent feed-forward layer, followed by another residual connection and normalization block. Normalization is applied to avoid the internal covariate shift [30, 31] through layer normalization. Self-attention is performed not only on the current encoder inputs but also on the shifted decoder outputs. The decoder structure performs almost the same operations as the encoder, including a second attention layer. All decoder units receive the key-value output pair from the final encoder unit while maintaining the query from its first attention layer to the second. The entire encoder structure in a transformer consists of a stack of N identical encoder units, where the units do not share weights. The same approach applies to the decoder structure [8]. The components involved in a transformer model are briefly discussed in the next subsection.

IV.4.1 Embedding Layer

The trainable embedding layers used in the transformer model in [8] serve to map sentences or words to numerical vectors in a natural language processing (NLP) task. The input/output embedding layers are removed in this work since the input is numeric time-series data and not text/speech. Instead, the input data is either directly fed to the positional encoding layer or passed on to create query, key and value matrices.

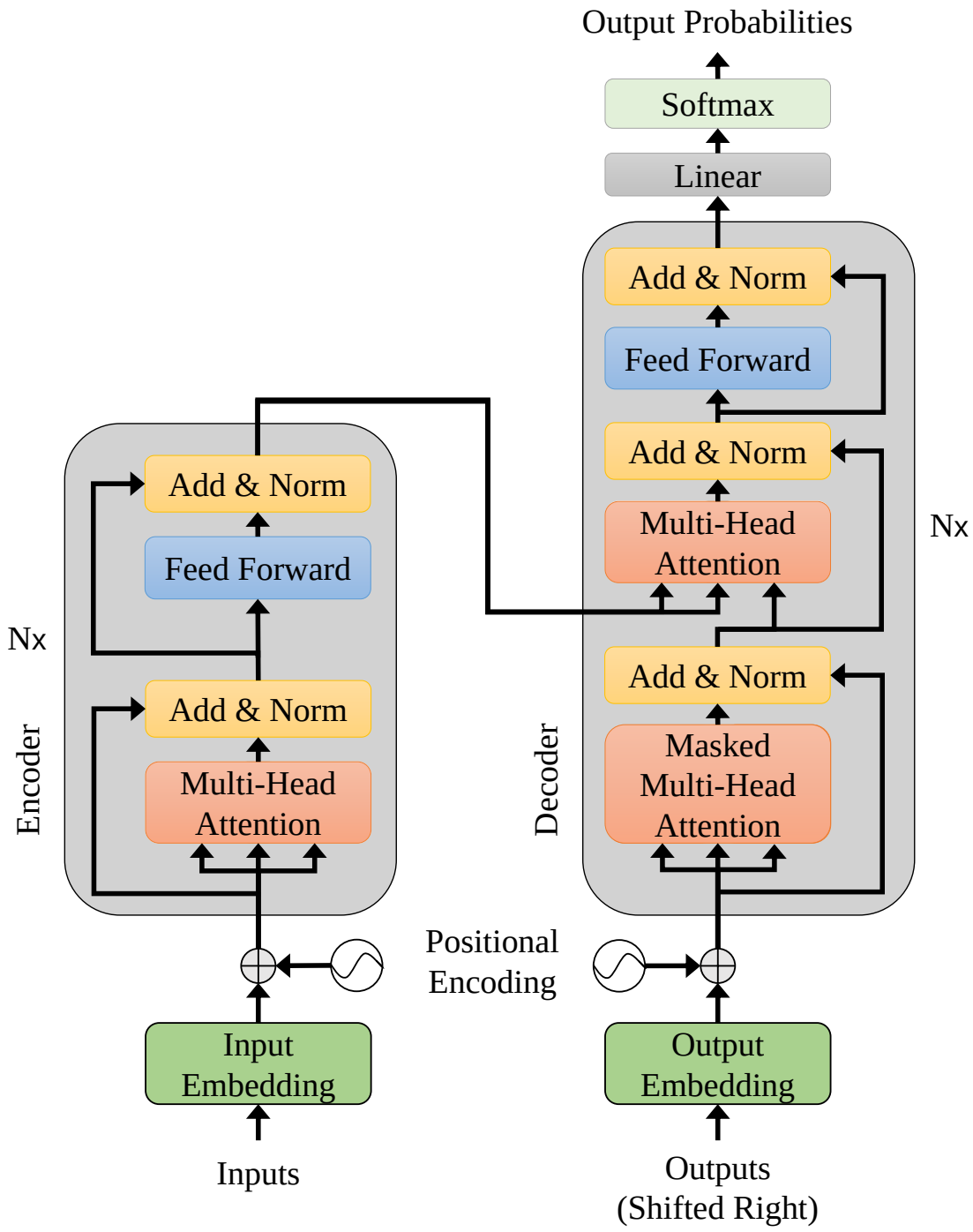


Figure IV.1: Transformer Architecture [8].

IV.4.2 Positional Encoding

Positional encoding introduces information about the relative/ absolute position of the input data. Positional encoding is a valuable tool for realizing the order of information in time-series data since transformer models receive inputs in parallel and not in sequences. Similar to the equations proposed in [8], the transformer models presented in this work also applies a sinusoidal representation along the feature dimension of the input data as shown in Eqn.IV.1 and IV.2.

$$PE_{(\mathrm{pos,2i})} = sin(pos/10000^{(2i/d_{model})}) \tag{IV.1}$$

$$PE_{(\mathrm{pos,2i+1})} = cos(pos/10000^{(2i/d_{model})}) \tag{IV.2}$$

where, *pos* is the temporal position and i is the current feature dimension of the input data. d_{model} refers to the total no. of dimensions in the feature space and it varies for the encoder and decoder layers. The aforementioned equations create a fixed geometric progression from 2π to $10000 \cdot 2\pi$ [8], with d_{model} equalling to the number of features of the input of the layer it is being applied to (i.e. encoder or decoder). This matrix is then added to the input, thus creating positional encoded input data.

A drawback of the positional encoding above is that for increasing number of features in the positional matrix, the sinusoidal angles in Eq. IV.1 and IV.2 tend to be 0 due to the presence of a large denominator. Thus, a sine or cosine operation on such an angle produces either 0 or 1, resulting in a non-uniform positional encoding. This raises the need for a non-uniform positional encoding on input data with small feature size. The transformer models proposed in this work are implemented with and without positional encoding to test the need for it in the C-MAPSS dataset.

IV.4.3 Self-Attention

Self-attention requires query *(Q)*, key *(K)* and value *(V)* vectors, created by passing the incoming input, I through three separate weight matrices W_q, W_k, $W_v \in \mathbb{R}^{d_{\mathrm{i}} \times d}$, i.e. $Q = W_{\mathrm{q}} \cdot I$, $K = W_{\mathrm{k}} \cdot I$ and $V = W_{\mathrm{v}} \cdot I$. Here d_i is the no. of input features and d is the corresponding layer hidden dimension with,

$$d = \begin{cases} d_{\mathrm{e}}, & Encoder \\ d_{\mathrm{d}}, & Decoder \end{cases} \tag{IV.3}$$

Self-attention is then computed as a dot-product between Q and K vectors to generate the alignment score which is scaled by a scaling factor of $\frac{1}{\sqrt{d_k}}$ and passed to a softmax layer [8] yielding,

$$W = Softmax(\frac{Q \cdot K^T}{\sqrt{d_{\mathrm{k}}}}) \tag{IV.4}$$

where, d_k = key *(K)* dimensions in a specific layer. The generated attention weights are then multiplied to the value (V) vector, thus creating a context between a certain query and all the other values in that sequence. Therefore, the scaled dot-product attention (SDA) is finally given by,

$$\mathrm{SDA}(Q, K, V) = Softmax(\frac{Q \cdot K^T}{\sqrt{d_{\mathrm{k}}}}) \cdot V \tag{IV.5}$$

IV.4.4 Multi-Head Attention

Multi-head attention (MHA) denotes the parallel computation of SDA for a specified number of heads, *h*. The query, key and value vectors, each with feature size of *d*, are multiplied with individual weight matrices, $W_i^Q \in \mathbb{R}^{d \times d_q}$, $W_i^K \in \mathbb{R}^{d \times d_k}$ and $W_i^V \in \mathbb{R}^{d \times d_v}$, where, *i* denotes each head ranging from *1,2,...h*. The weight matrices do not share weights across the heads. *Q, K* and *V* are thus sub-divided into *h* sets i.e. $(Q,K,V)_1$, $(Q,K,V)_2$,...,$(Q,K,V)_h$. The resulting dimensions are equal in shape and, $d_q{=}d_k{=}d_v{=}d/h$. Each of these sets are mapped into self-attention outputs for *h* separate times. The attention from all *h* self-attention blocks are concatenated and passed through a weight matrix, $W_a \in \mathbb{R}^{d_k \times d}$ to produce the final attention. This operation is represented as,

$$\mathrm{MHA}(Q, K, V) = Concat(head_1, head_2, ..., head_\mathrm{h}) \cdot W_\mathrm{a} \tag{IV.6}$$

where,

$$head_\mathrm{i} = \mathrm{SDA}(QW_i^Q, KW_i^K, VW_i^V). \tag{IV.7}$$

MHA requires the same computation cost as a single head SDA, due to fewer dimensions in each head than higher dimensions in the latter. Moreover, the individual self-attention heads jointly attend to information originating from multiple representation subspaces, form a context and map them to a combined representation subspace [8]. We subsequently perform MHA on both the feature dimension and temporal dimension. In feature attention, instead of passing the Q, K, V vectors through individual linear layers, the dimensions are reduced by splitting them physically into h different matrices along the features space. In temporal attention, the Q, K, V vectors are divided along the timestep dimension to create matrices of shorter time-length but with full dimensionality. Due to this split nature, the attention procedures are termed as Split-Feature MHA (Sec.IV.5.3) and Split-Temporal MHA (Sec.IV.5.1).

IV.4.5 Masking

In the transformer model, look-ahead masks are applied in the first multi-head attention block of the decoder. As the decoder performs self-attention on its previously generated target sequence, the future tokens must be concealed from this operation. The attention must only tend to the present and past tokens. Therefore, a look ahead mask that matches the shape of the QK^T dot product is created. The mask contains '0' in positions that must be revealed and $-\infty$ in the positions to be concealed. This mask is then added to the QK^T vector before passing it to the softmax layer. This restricts the attention from "looking ahead" in the future and uses only the past and present information to predict the subsequent output.

IV.4.6 Layer Normalization

The problem of Internal Covariate Shift (ICS) [30, 31] can be solved by normalizing the outputs from the previous layer before entering the current hidden layer. Layer Normalization normalizes the input across the feature dimension i.e. normalizes each features to zero mean and unit variance. In this method, the mean μ^l and standard

deviation σ^l are calculated across all the hidden nodes in a specific l^{th} layer [30]. The equations for layer normalization are given as,

$$LayerNorm_i = \gamma \hat{a}_i + \beta \tag{IV.8}$$

where

$$\hat{a}_i = \frac{a_i - \mu_i^l}{\sqrt{\sigma_i^2 + \epsilon}} \tag{IV.9}$$

and

$$\mu_i^l = \frac{1}{H} \sum_{i=1}^{H} a_i^l \tag{IV.10}$$

$$\sigma_i^l = \sqrt{\frac{1}{H} \sum_{i=1}^{H} (a_i^l - \mu^l)^2}. \tag{IV.11}$$

Here a_i^l is the vector of inputs of all H nodes in the l^{th} hidden layer. Layer normalization eliminates a major drawback of batch normalization [31], which is the constraint of not being able to train a model with input data of batch/mini-batch size of 1. The transformer architectures proposed in this work also deploy layer normalization in a similar manner.

IV.4.7 Position-wise Feed-Forward Networks

The feed-forward networks (FFN), as shown in Fig. IV.1 consists of two fully connected (FC) linear layers with a ReLU activation after the first FC layer. The FFNs are independent and do not share weights [8]. The FFNs deployed in our proposed architectures replace the ReLU activation with a Leaky ReLU activation. We deploy FFNs in our architecture as- one in the Shared Temporal Attention (STA) block (Sec.IV.5.2), and one each in the central encoder and decoder layers.

$$\begin{aligned} FFN(x) = (&max(0, x \cdot W_1 + b_1) \\ &+ 0.1 \cdot min(0, x \cdot W_1 + b_1)).W_2 + b_2 \end{aligned} \tag{IV.12}$$

Where, x is the input to the FFN and $x \in \mathbb{R}^d$. W_1, b_1 and W_2, b_2 are the first and second linear layer weights and biases. $W_1 \in \mathbb{R}^{d \times d \cdot m}$, $b_1 \in \mathbb{R}^{d \cdot m}$, $W_2 \in \mathbb{R}^{d \cdot m \times d}$, $b_2 \in \mathbb{R}^d$. d = no. of features of x and $m \in \mathbf{Z}^+$ (a positive integer) is a multiplier to increase the inner dimensionality of the FFN layer.

IV.5 Temporal Attention-based Transformer for Sequence RUL Prediction

The original transformer architecture has been developed for natural language processing problems and has later been extended to image processing. However, the sequence RUL prediction is differing to the aforementioned domains as its input is typically represented by multivariate time-series, i.e. sensor data. Hence, we require for novel methodologies to perform self-attention suitable for multivariate time series data. We tackle this by two

types of self-attention, namely temporal attention on the univariate sensor signal and split-feature attention for attending on information from different channels. Furthermore, we present a robust and computation-efficient approach for creating feature heads. Finally, we propose two novel transformer architectures for sequence RUL prediction using the above attention blocks.

IV.5.1 Split-Temporal Multi-Head Attention (STMHA)

Temporal heads are created from an input signal by splitting it into a divisible number of shorter signals. The original input sequence is divided into $heads_t$ no. of mini-sequences while maintaining the actual order of the time series data. The mini-sequences are fed to the STMHA layer according to the original order. The hidden representations for these mini-sequences are calculated consecutively in the temporal heads where the weights are shared. Therefore, there is no information loss since all the mini-sequences are mapped to their respective hidden representations and subsequently concatenated to produce the final attended output. A signal of length, $t = t_1,t_2,...,t_T$ and features, $f = f_1,f_2,...,f_i$ is passed through weight matrices, $W^Q \in \mathbb{R}^{i \times d_e}$, $W^K \in \mathbb{R}^{i \times d_e}$ and $W^V \in \mathbb{R}^{i \times d_e}$, to form query, key and value *(Q, K, V)* vectors. Here, i = size of input feature dimension and d_e = size of encoder hidden dimension. Each of the Q, K and V vectors are then split along the time dimension to form $heads_t$ number of temporal heads. Self-attention is then performed on each of the temporal heads to produce attended vectors $\{Z_1^{attn}, Z_2^{attn}, .., Z_{heads_t}^{attn}\} \in \mathbb{R}^{t_{\text{heads}} \times d_e}$. A temporal-attended output, $Z^{attn} \in \mathbb{R}^{T \times d_e}$ is finally formed by concatenating the attended vectors along the time-dimension. STMHA is used along with additional components on the encoder of the proposed transformer models presented below.

IV.5.2 Shared Temporal Attention Block for Transformer Encoder

A novel concept of raw feature segmentation and univariate STMHA is presented in this section, with the aim to reducing noise or interference from adjacent sensor signals while attending the timesteps in a specific feature. As shown in Fig. IV.2, a multivariate input signal is segmented into univariate signals, and each of those signals flows through a Shared Temporal Attention (STA) block. Note, that for segmented univariate signals, the input feature dimension in the STMHA block now becomes *i=1*. The univariate temporal attended output passes through a FFN layer, followed by a residual addition and layer normalization, yielding to the following equations for any univariate temporal attended vector Z_i^{attn}:

$$\begin{aligned} FFN(Z_i^{attn}) = (&max(0, (Z_i^{attn} \cdot W_1 + b_1)) \\ &+ 0.1 \cdot min(0, (Z_i^{attn} \cdot W_1 + b_1))).W_2 \\ &+ b_2 \end{aligned} \tag{IV.13}$$

$$Z_i^{norm} = LayerNorm(Z_i^{attn} + FFN(Z_i^{attn})) \tag{IV.14}$$

$$Z_i = W_{\text{flat}} \cdot Z_i^{norm} + b_{\text{flat}} \tag{IV.15}$$

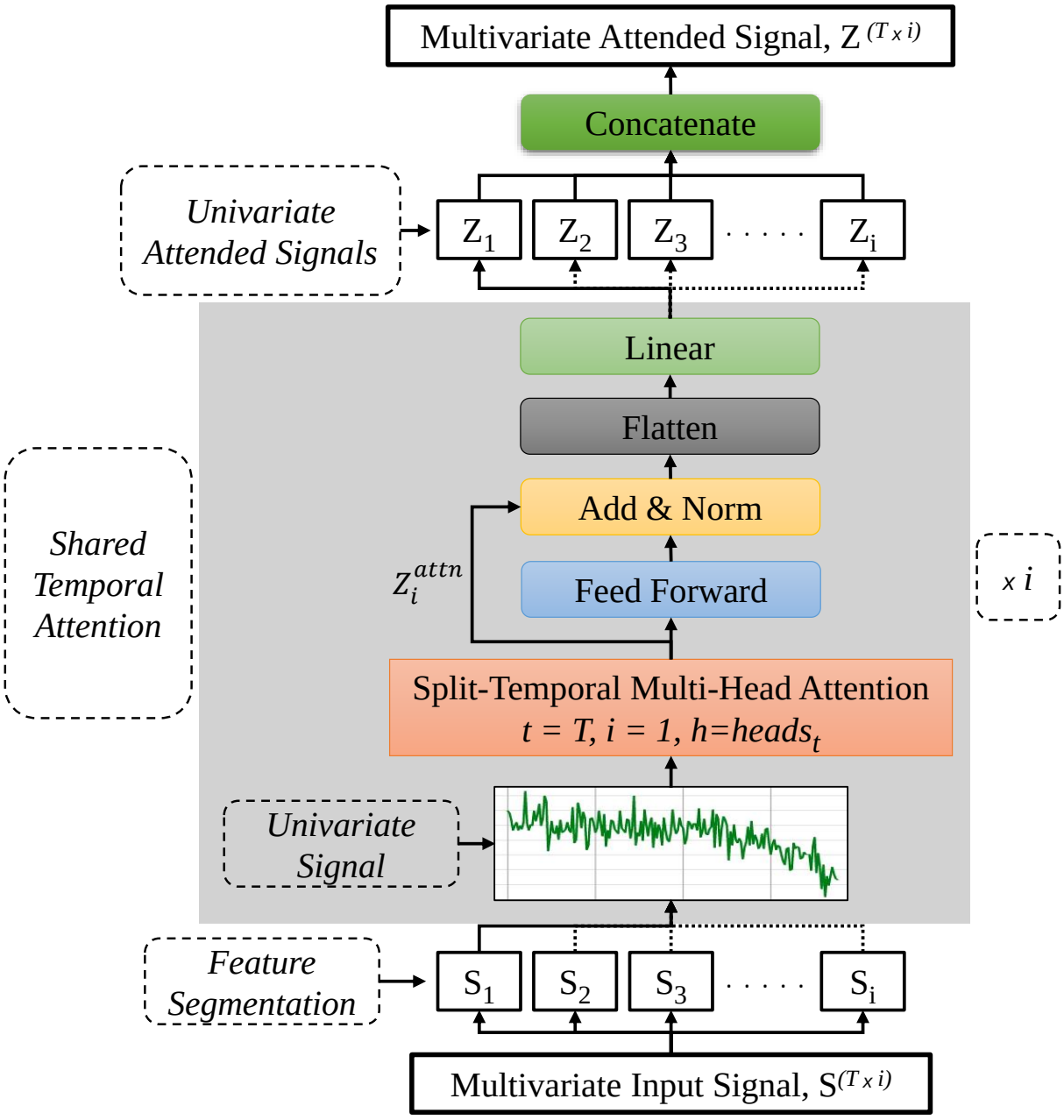


Figure IV.2: Shared Temporal Attention Block for Transformer Encoder.

Where, W_1, W_2 and b_1, b_2 are the weights and biases of the two feed-forward layers and $W_1 \in \mathbb{R}^{de \times (m \cdot de)}$, $W_2 \in \mathbb{R}^{(m \cdot de) \times de}$. $m \in \mathbf{Z}^+$ is a multiplier that increases the inner dimensionality of the FFN layer. The normalized vector, Z_i^{norm} is flattened and multiplied by a weight matrix, $W_{flat} \in \mathbb{R}^{(T \cdot de) \times T}$ to produce a shared temporal-attended univariate signal $Z_i \in \mathbb{R}^{T \times 1}$. For a multivariate raw input signal $S \in \mathbb{R}^{T \times i}$, the STMHA block operates for i no. of segmented sensor signals to produce shared univariate attended signals, $\{Z_1, Z_2, ..., Z_i\} \in \mathbb{R}^{T \times 1}$. These signals are concatenated along the feature dimension to form a multivariate shared-temporal attended output, $Z \in \mathbb{R}^{T \times i}$.

$$Z = concat(Z_1, Z_2, Z_3, ..., Z_i) \tag{IV.16}$$

The term 'shared' signifies that all the layers involved in the STA block share their weights across the segmented raw univariate signals. The STA layer splits each univariate signal into multiple segments of shorter length while maintaining the original sequence, and then performs self-attention on these segments. For each sensor, this layer attends to the life-cycle information from all the split segments of the time-series signal and extracts the degradation pattern to an attended univariate output. The learned weights from each sensor during this process is shared across the other sensors. This shared architecture is particularly suitable for a diverse set of raw sensor parameters that provide non-uniform patterns of decay throughout an engine's life cycle. The shared weights allow the model to comprehend the individual sensor relationships more effectively without the cost of an increased number of parameters.

IV.5.3 Split-Feature Multi-Head Attention

To further reduce computation cost and training time, we propose a new concept of creating feature heads by splitting the input features, named as Split-Feature Multi-Head Attention (SFMHA). *Q, K and V* vectors are formed by passing the input vector of length, $t = t_1, t_2, ..., t_T$ and features, $f = f_1, f_2, ..., f_i$ through weight matrices, $W^Q \in \mathbb{R}^{i \times d_d}$, $W^K \in \mathbb{R}^{i \times d_d}$ and $W^V \in \mathbb{R}^{i \times d_d}$, where d_d is the decoder hidden size. The *Q, K and V* vectors are then split along the feature dimension into ($heads_f$) feature heads, with the new split vectors containing a dimension size, i_{heads}. Self-attention is performed on the split feature heads to produce corresponding attended vectors $\{Z_1^{attn}, Z_2^{attn}, .., Z_{heads_f}^{attn}\} \in \mathbb{R}^{T \times i_{\text{heads}}}$. The shared technique allows the network to find a correlation along the feature space without the need for increasing the number of trainable parameters and computation time. This is widely useful in prediction tasks of multivariate datasets with a wide array of input features, such as the C-MAPSS dataset. A feature-attended output vector, $Z^{attn} \in \mathbb{R}^{T \times d_{\text{d}}}$ is finally created by concatenating the individual feature head attention outputs along the feature dimension. SFMHA replaces the conventional multi-head attention mechanism in the decoder section of the proposed transformer architectures. Additionally, in one of the proposed models shown in Sec.IV.5.5, SFMHA is also incorporated in the encoder section.

IV.5.4 Shared Temporal Attention Transformer *(STAT)*

Having presented the novel attention blocks, we now propose our novel transformer architecture called Shared Temporal Attention Transformer (STAT) shown in Fig. IV.3. The architecture resembles the encoder-decoder format of the transformer architecture

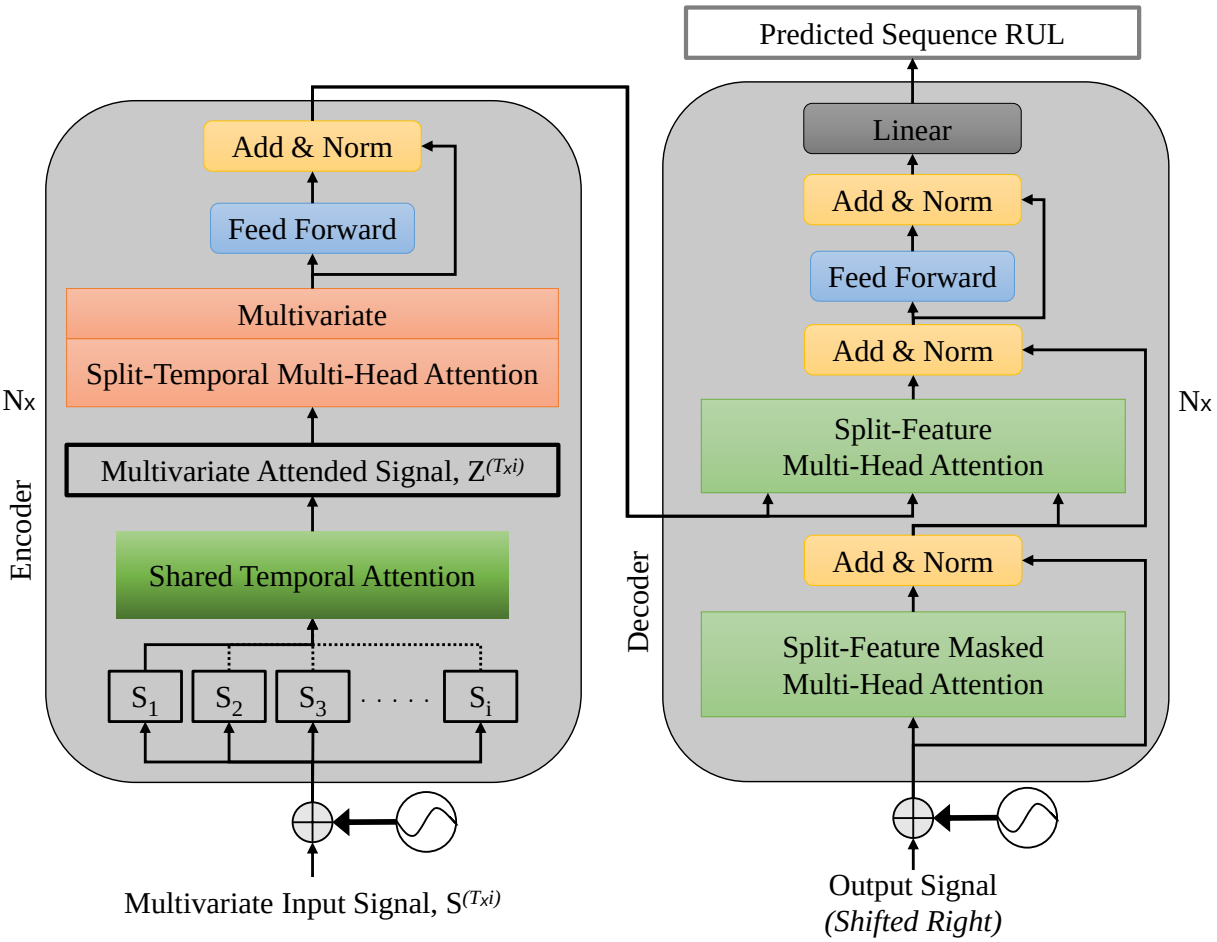


Figure IV.3: Shared Temporal Attention Transformer *(STAT)*.

proposed in [8], where the encoder and decoder sections are formed off of $N\times$ stacked encoder and $N\times$ stacked decoder layers respectively. Weights are not shared across the individual encoder layers and, the same applies for the decoder layers as well. Positional encoding (PE) is optionally included before the first encoder and decoder layer to introduce information about the relative or absolute position of each input data point.

Encoder

In the encoder layer, the features in a multivariate input signal, $S \in \mathbb{R}^{T\times i}$ are first segmented to form individual univariate signals, $\{S_1,\ S_2,\ ..,\ S_i\} \in \mathbb{R}^{T\times i}$. These segmented univariate signals are passed through the Shared Temporal Attention (STA) layer to form a multivariate attended signal $Z \in \mathbb{R}^{T\times i}$. A second phase of attention in the form of STMHA is performed on the multivariate attended signal Z. The key difference between the first and second attention block is that, in the first attention phase, STMHA is performed in a shared recurrent manner on each of the i segmented feature sensors whereas, in the second attention phase, STMHA is performed on a multivariate signal as a whole. Sharing weights across all the segmented features in the first attention phase allows the encoder to learn high-level feature representations across the sensors in the input data, while at the same time learn the inherent degradation pattern within each sensor, without the interference from another. The multivariate attended signal, Z now possesses more coherence across the feature domain due to the shared attention mechanism. STMHA is then performed on the multivariate signal, Z, which allows interaction between the coherent features while performing self-attention along the temporal dimension. This layer is followed by a FFN layer, a residual connection and layer normalization. For any multivariate STMHA output, Z_{stmha}, we have

$$\begin{aligned} FFN(Z_{\text{stmha}}) = (&max(0, (Z_{\text{stmha}} \cdot W_1 + b_1)) \\ &+ 0.1 \cdot min(0, (Z_{\text{stmha}} \cdot W_1 + b_1))).W_2 \\ &+ b_2 \end{aligned} \tag{IV.17}$$

$$Z^{norm}_{stmha} = LayerNorm(Z_{\text{stmha}} + FFN(Z_{\text{stmha}})) \tag{IV.18}$$

where, W_1, W_2 and b_1, b_2 are the weights and biases of the two linear layers in the FFN. $W_1 \in \mathbb{R}^{i\times(m\cdot i)}$, $W_2 \in \mathbb{R}^{(m\cdot i)\times i}$ and $m \in \mathbf{Z}^+$ is the dimensionality multiplier of the FFN layer. This normalized output from each encoder layer is passed on to the next encoder layer where the aforementioned attention steps are repeated. This process repeats itself for N times until the N-th encoder layer is reached. The output of the N-th encoder layer is multiplied by weight matrices W^k_{ED}, $W^v_{ED} \in \mathbb{R}^{i\times(d_{\text{d}})}$, to form key and value inputs for the decoder layers.

Decoder

The first decoder layer in the stack is initiated with a tensor of zeros, that serves as the start-of-sequence, $< SOS >$ token. The first block in the decoder layer is a split-feature masked multi-head attention operation with look-ahead masking to prevent the decoder from using future information and process only the information obtained from the current and previous timesteps. The feature attended output from this layer passes through a

residual connection and then a layer normalization. The normalized output is the query vector Q for the second SFMHA type attention block where key K and value V vectors are obtained from the output of the N-th encoder layer. This is followed by a residual connection, layer normalization and an FFN layer composed of two FC layers with non-linear Leaky ReLU activation after the first FC layer. For any normalized input $Y_D^{norm1} \in \mathbb{R}^{T_o \times d_e}$ to the FFN layer, we have,

$$\begin{aligned} FFN(Y_D^{norm1}) = (max(0, (Y_D^{norm1} \cdot W_1 + b_1)) \\ + 0.1 \cdot min(0, (Y_D^{norm1} \cdot W_1 + b_1))).W_2 \\ + b_2 \end{aligned} \tag{IV.19}$$

$$Y_D^{norm2} = LayerNorm(Y_D^{norm1} + FFN(Y_D^{norm1})) \tag{IV.20}$$

where, W_1, W_2 and b_1, b_2 are the weights and biases of the two linear layers in the decoder FFN and, $W_1 \in \mathbb{R}^{d_d \times (m \cdot d_d)}$, $W_2 \in \mathbb{R}^{(m \cdot d_d) \times d_d}$. $m \in \mathbf{Z}^+$ is the dimensionality multiplier of the decoder FFN layer and d_d is the decoder hidden dimension. The final step in a decoder layer is a linear layer of weight, $W_{RUL} \in \mathbb{R}^{d_e \times d_{RUL}}$, that predicts a sequence output with dimension, $d_{RUL} = 1$. This predicted output is sent to the next decoder layer and the process repeats until the N-th decoder layer finally produces the sequence RUL labels. The N decoder layers do not share weights just like the encoder layers. Every decoder layer receives the same K and V vectors from the N-th encoder layer to perform the second attention phase with the Q vector generated from the masked SFMHA attention phase.

IV.5.5 Feature-Represented Shared Temporal Attention Transformer *(FeaR-STAT)*

We alternatively propose Feature-Represented Shared Temporal Attention Transformer (FeaR-STAT) as shown in Fig. IV.4. This architecture resembles the proposed STAT architecture (Sec.IV.5.4) except for the encoder layer. As seen in the STAT architecture, the encoder layer fully relies on STMHA type attention while, the decoder on SFMHA type attention. FeaR-STAT removes the absolute dependency of the encoder on temporal attention by replacing the second attention phase in the encoder with an SFMHA type attention. By performing SFMHA on the STA multivariate output, the individually attended features are allowed to perform self-attention across the feature dimension for the entire length of the input signal. This enhances the encoder's ability to attend to the complex degradation pattern throughout all the features across the full input signal length and consequently allows the encoder to provide K and V vectors to the decoder layers with effectively learned feature and temporal representations for predicting the RUL sequence.

The STAT and FeaR-STAT encoders resemble one another until after the STA layer. Next, the multivariate attended signal, Z is passed through a FC linear layer with a weight matrix, $W_{expand} \in \mathbb{R}^{i \times d_\mathrm{d}}$, to create $Z \in \mathbb{R}^{T \times d_\mathrm{d}}$. The expansion in feature size is necessary since, in SFMHA, the input signal Z is split across the feature domain to create $heads_f$ signals, each of length T and $i_{heads} = d_d/heads_f$ features, where, i_{heads}, d_d, $heads_f \in \mathbf{Z}^+$. SFMHA is performed on the feature expanded signal Z to produce $Z_{sfmha} \in \mathbb{R}^{T \times d_\mathrm{d}}$.

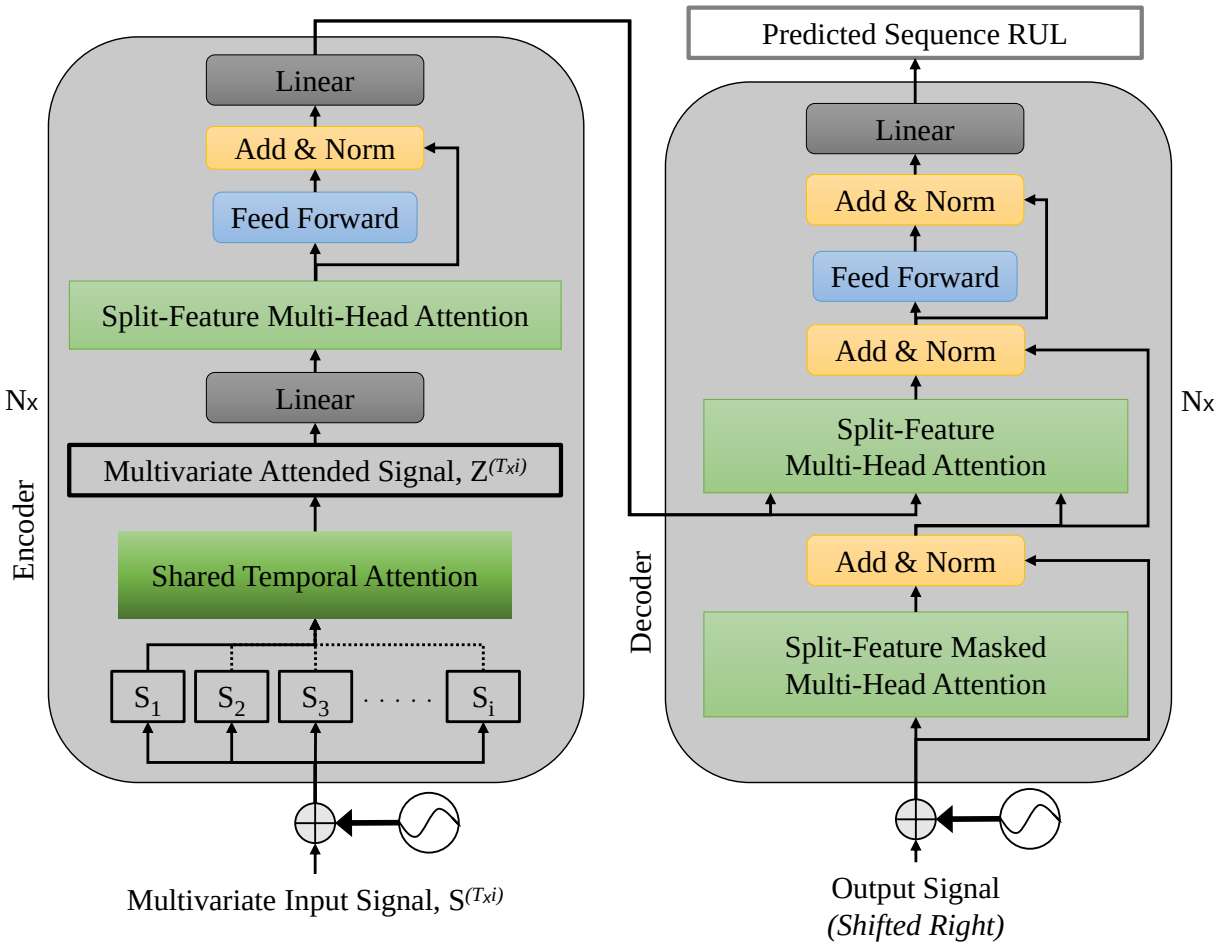


Figure IV.4: Feature-Represented Shared Temporal Attention Transformer *(FeaR-STAT)*.

This feature attended output passes through a FFN layer, residual connection and layer normalization yielding,

$$\begin{aligned} FFN(Z_{sfmha}) = (&max(0, (Z_{sfmha} \cdot W_1 + b_1)) \\ &+ 0.1 \cdot min(0, (Z_{sfmha} \cdot W_1 + b_1))).W_2 \\ &+ b_2 \end{aligned} \tag{IV.21}$$

$$Z^{norm}_{sfmha} = LayerNorm(Z_{sfmha} + FFN(Z_{sfmha})) \tag{IV.22}$$

where, W_1, W_2 and b_1, b_2 are the weights and biases of the two linear layers in the encoder FFN and, $W_1 \in \mathbb{R}^{d_d \times (m \cdot d_d)}$, $W_2 \in \mathbb{R}^{(m \cdot d_d) \times d_d}$, and $m \in \mathbf{Z}^+$ is the dimensionality multiplier of the encoder FFN. The final step in a FeaR-STAT encoder layer is passing the Z^{norm}_{sfmha} output through a linear layer with weights, $W_{reduce} \in \mathbb{R}^{d_\mathrm{d} \times i}$, to create the encoder output $Z^{norm}_{sfmha} \in \mathbb{R}^{T \times i}$. The FeaR-STAR encoder operation is recurrent i.e., the output from one layer is passed on to the next for N times until the N-th encoder layer generates a latent representation for the decoder layers.

The decoder receives this encoder latent representation in the form of K and V vectors for the second SFMHA attention phase. Similar to STAT architecture, the N encoder layers do not share weights and neither do the N decoder layers.

IV.6 Experimental Results and Comparison

In this section we present the experimental results of our novel transformer architecture. In this work, the Turbofan Engine Degradation Dataset (referred to as C-MAPSS dataset henceforth) is used to evaluate the model performances.

IV.6.1 Dataset Description

C-MAPSS is a model-based simulation program prepared by NASA and provides the simulation of a large commercial turbofan engine, used in aircraft propulsion [32]. It was used to generate the Turbofan Engine Degradation Simulation Dataset and the PHM 2008 Challenge Dataset [33].

The C-MAPSS dataset consists of four different working environments FD001, FD002, FD003 and FD004. Each sub-dataset consists of a train and a test dataset, with a corresponding RUL data containing the last run-time for each test engine. In each of the four sub-datasets, the engines start operating in a healthy state and run until the system fails due to gradual degradation. In the test samples, the sensor measurements are trimmed from a certain period (as provided in the RUL database) before system failure and that corresponding period is taken for evaluation and scoring. Initial wear are considered in all the train and test engines [33, 34]. Each of the train and test C-MAPSS sub-datasets contain outputs from 21 sensors. These measurements are sensor responses to 14 different health parameter inputs that simulate degradation scenarios in any of the engine's rotating components [33]. Sub-datasets FD001 and FD003 operate with one operational setting, thus resulting in one unique operating condition whereas, FD002 and FD004 operate with three different operational settings. The three operational settings

result in six unique operating conditions. This variation in operating conditions creates fluctuations in sensor readings, since data points in consecutive cycles may not belong to the same operating condition.

Following previous work [16], we exclude Sensors *S1, S5, S6, S10, S16, S18, S19* as they show monotonous pattern in their readings. Additionally, the inclusion of the history of operating regimes as features along with the selected sensor readings to optimize model performance is suggested in [16]. The approach is implemented for the sub-dataset FD002. However, in order to demonstrate the effectiveness of the proposed STA layer in understanding individual sensor relationships and overall degradation pattern, we separately provide performance scores by the STAT and FeaR-STAT architectures using the complete set of sensor readings as input from the C-MAPSS dataset.

Further, we use Min-Max Scaling to the range [-1, 1]. Due to the six different operating conditions in FD002 and FD004, the magnitude and distribution of the data points within each condition vary and hence, they require to be scaled separately [35]. The individual sensor data from each of the six conditions are segregated, scaled and then combined according to their original pre-scaling positions.

Performance Evaluation Metric

The scoring functions [33] for the C-MAPSS dataset is given by

$$s_{\mathrm{i}} = \begin{cases} e^{\frac{d_i}{a_1}} - 1, & d_{\mathrm{i}} < 0 \\ e^{\frac{d_i}{a_2}} - 1, & d_{\mathrm{i}} \geq 0 \end{cases} \tag{IV.23}$$

$$S = \sum_{i=1}^{N} s_{\mathrm{i}} \tag{IV.24}$$

where, $a_1 = 13$, $a_2 = 10$, d_i = *Predicted RUL - Target RUL* for the *i-th* engine, s_i = score for the *i-th* engine, N = total engines in a test sub-dataset and S is the final performance score for that sub-dataset [17]. This is an inverse scoring function i.e. a lower total score represents better performance by the model and a higher score denotes the opposite. Additionally, the scoring function exclusively depends on the generated RUL for the final input sequence, and not on the RUL obtained throughout the prediction spectrum.

IV.6.2 Optimizer Warmup and Learning Rate

We utilize the Adam optimizer [36] with parameters $(\beta_1, \beta_2) = (0.9, 0.98)$ and $\epsilon = 10^{-08}$. Instead of using a fixed learning rate LR, we use a warm-up scheduler where it is increased linearly for the first $warmup_{steps}$ training steps and, decreased proportionally to the inverse square root to the $warmup_{steps}$ [8]. Thus, LR is governed by

$$LR = d_d^{-0.5} \cdot min(step_{num}^{-0.5}, step_{num} \cdot warmup_{steps}^{-1.5}) \tag{IV.25}$$

where, d_d is the decoder hidden size, $step_{num}$ is the instantaneous training iteration and $warmup_{steps}$ is the total number of training iterations for the architecture. Contrary to [8], the $warmup_{steps}$ is not kept constant in this work and it is dependent upon a certain hyperparameter called Optimizer Warmup percentage, $W_{opt}\%$. Depending on a certain

$W_{opt}\%$, Eq. IV.26 calculates the $warmup_{steps}$ iterations as an approximate no. of epochs, e and schedules the learning rate accordingly.

$$warmup_{steps} \approx e \cdot \frac{N}{BS} \cdot W_{\text{opt}}\% \tag{IV.26}$$

where, N= total no. of training samples in mini-batches, BS= mini-batch size.

IV.6.3 Hyperparameters

The hyperparameters range for optimally training the STAT model are presented in Table IV.1 where Grid Search algorithm is used to find the set of hyperparameters for best possible performance. The model performance is evaluated with multiple possible temporal and feature heads *(*$heads_t$ and $heads_f$*)*, with multiple stacked encoder and decoder layers, L_{ED}. A fixed mini-batch size, BS of 512 is selected. Teacher forcing, TF is constantly set to 0% to test the model's ability of predicting RULs without being dependent on actual RUL labels. Multiple optimizer warmup percentages, $W_{opt}\%$ are assessed to understand the role of a non-uniform learning rate in the performance of the STAT and FeaR-STAT architectures. All weights are initialized using Xavier Uniform method [37] and biases are initialized with zeros.

Table IV.1: List of Hyperparameters for Proposed STAT and FeaR STAT Architectures

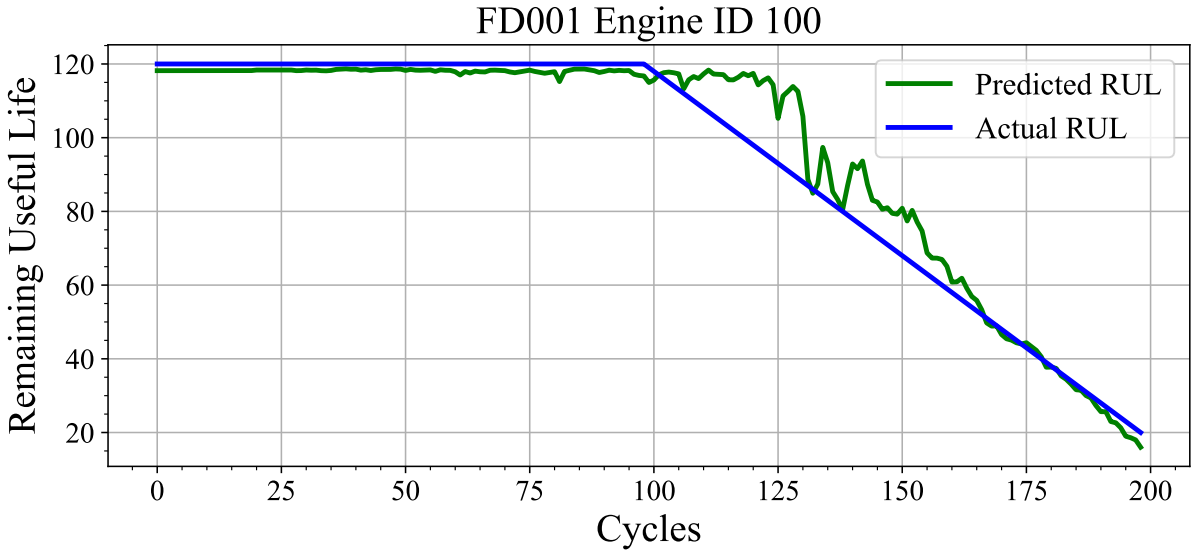
Hyperparameters	**Symbol**	**Range**
Temporal Heads	$heads_t$	{2, 4, 6, 8}
Feature Heads	$heads_f$	{4, 8, 16, 32}
Encoder Hidden Size	d_e	{4, 8, 16}
Decoder Hidden Size	d_d	{32, 64, 128}
Number of Layers	L_{ED}	{2, 3, 4}
Optimizer Warmup	W_{opt} %	{30, 50, 70}

IV.6.4 Temporal Heads & Sequence Lengths

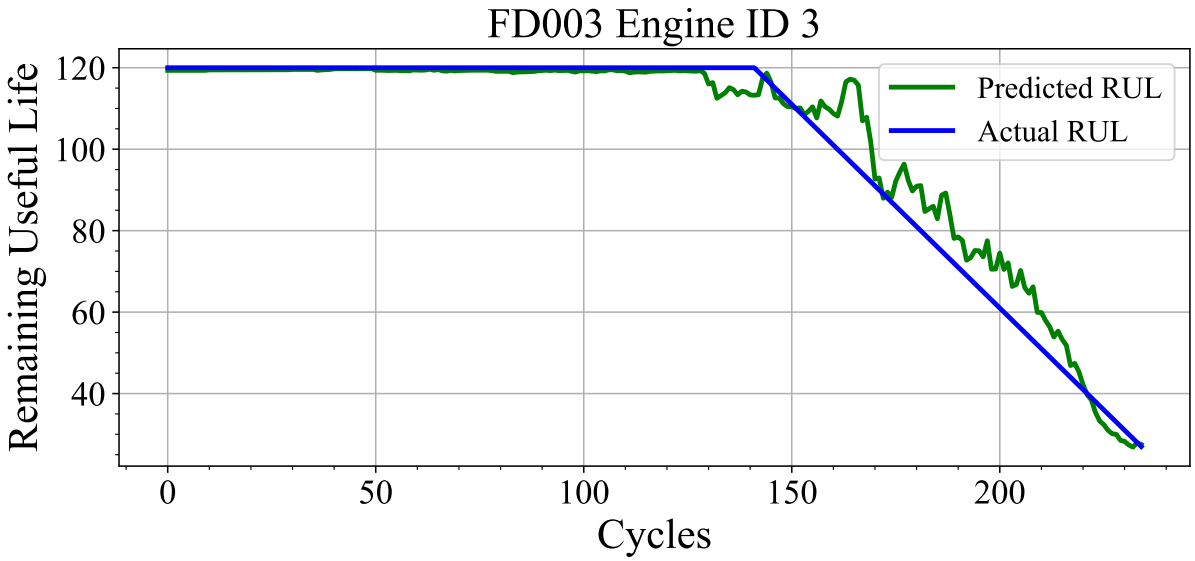
The proposed temporal attention technique in the STAT architecture requires splitting the raw input sequence of length T_i into multiple temporal heads, $heads_t$ such that *(*$T_i/heads_t$*)* $\in \mathbf{Z}^+$. For all possible $heads_t$ = *{2, 4, 6, 8}* that are used to tune the STAT model in this work, the set of input sequence lengths, T_i for the four C-MAPSS sub-datasets are selected correspondingly in a way that they fulfill the aforementioned condition. This is shown in Table IV.2. Additionally, the RUL sequence length, T_{RUL} is kept constant at 10 for all sub-datasets and for all temporal heads. After a grid search on these hyperparameter ranges, the best performing hyperparameter setting for STAT and FeaR-STAT models are shown in Table IV.3.

Table IV.2: Selection of Input Sequence length (T_i) depending on $heads_t$ for Proposed STAT and FeaR-STAT Architectures

$\mathbf{heads_t}$	$\mathbf{T_i}$
2	(30, 20, 36, 18)
4	(28, 20, 36, 16)
6	(30, 18, 36, 18)
8	(24, 16, 32, 16)

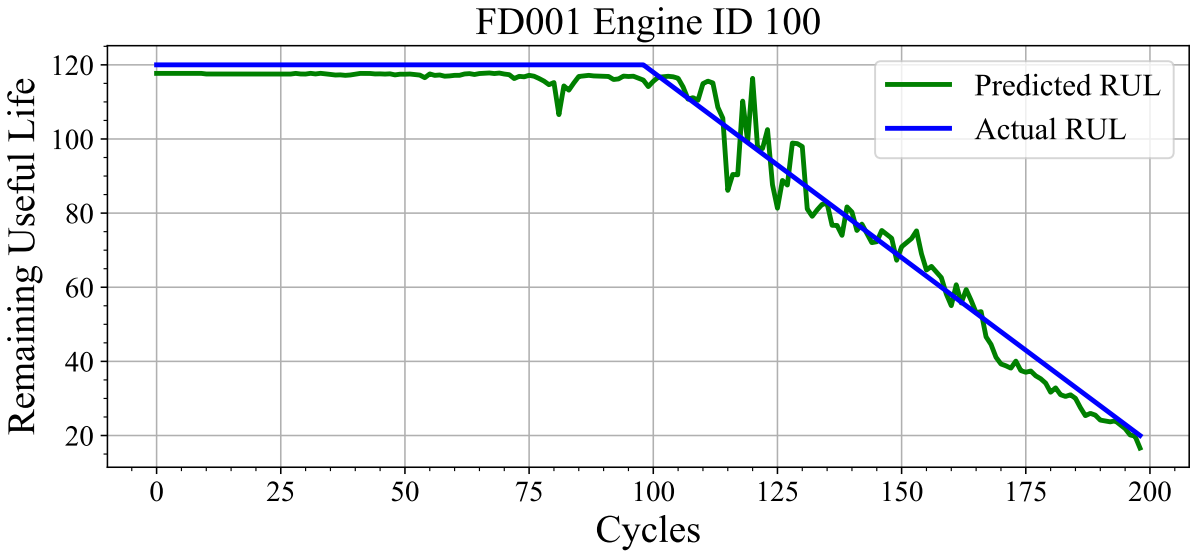


(a) FD001 Engine ID 100.

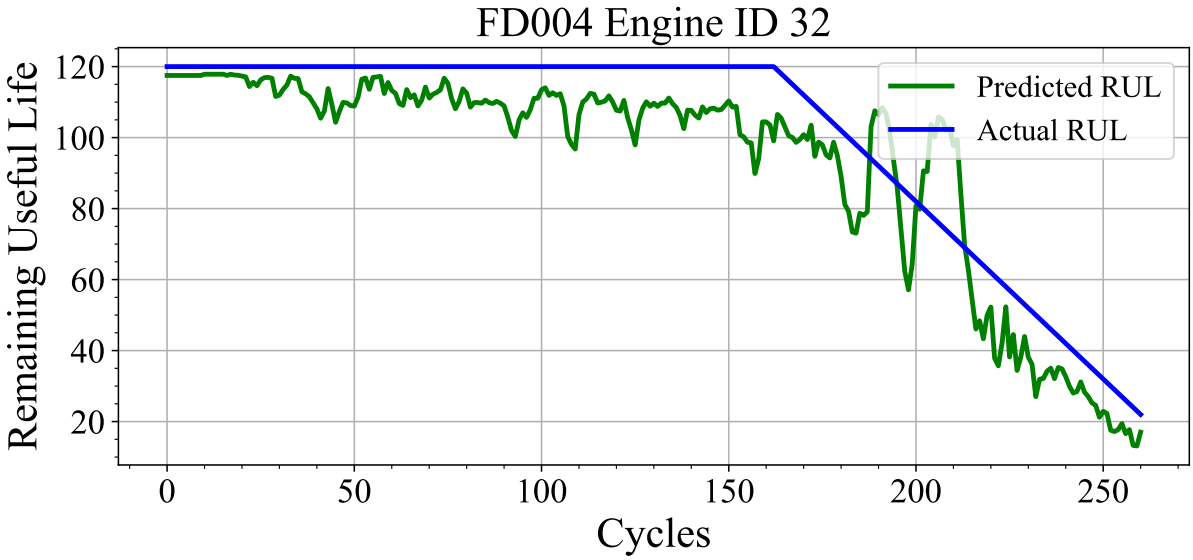


(b) FD003 Engine ID 3.

Figure IV.5: Remaining Useful Lifetime (RUL) estimation Plots by proposed STAT architecture.



(a) FD001 Engine ID 100.



(b) FD004 Engine ID 32.

Figure IV.6: Remaining Useful Lifetime (RUL) estimation Plots by proposed FeaR-STAT architecture.

Table IV.3: Hyperparameter sets for best scores in C-MAPSS Dataset by Proposed STAT & FeaR-STAT Model Architectures

Symbol	**Hyperparameter Values**			
	Pre-Selected Sensors		**Complete Sensors**	
	STAT	**FeaR-STAT**	**STAT**	**FeaR-STAT**
$heads_t$	4	4	4	4
$heads_f$	16	8	8	8
d_e	4	4	8	4
d_d	32	32	32	32
L_{ED}	2	2	2	2
$W_{opt}\%$	50	30	30	30

IV.6.5 Performance Analysis by Proposed STAT and FeaR-STAT Architectures using C-MAPSS Dataset

The results of the proposed STAT and FeaR-STAT architectures on the C-MAPSS dataset are presented in this section. The performance scores by the models are presented in Table IV.4 for input data with pre-selected sensors, and in Table IV.5 for input with the complete set of sensors. The corresponding set of hyperparameters for these scores are provided in Table IV.3. It is evident from the results that the proposed STA layer in both the models is capable of deducing the individual sensor relationships and the non-uniform degradation pattern from the diverse set of sensors. In both cases, the models can generate a balanced set of scores across all four sub-datasets. However, as the results provided in the literature are generated using pre-selected sensors, we opt for the results with a similar approach for further hyperparameter sensitivity analysis and literature comparison in the following sections.

Fig. IV.5 and Fig. IV.6 represent the RUL estimation plots by the proposed STAT and FeaR-STAT models respectively. It is evident from the plots that both models effectively capture the global degradation pattern in the sub-dataset engines. Early prediction can be observed in both the plots, denoting the ability of the models to detect faults in good time from the run-time data of the engines. However, the proposed STAT model provides more uniform predictions than the FeaR-STAT model, creating slight fluctuations in the degrading phase.

For the four sub-datasets FD001, FD002, FD003 and FD004 with pre-selected sensors, the approximate training duration required by the STAT and FeaR-STAT architectures for 120 epochs are presented in Table IV.6. Correspondingly, the inference time for one forward pass in the testing loop for both architectures are given in Table IV.7. The proposed models are trained on a system equipped with NVIDIA Quadro RTX 5000 GPU, Intel Core-i7 CPU with 2.5GHz clock speed and 32 GB RAM.

Table IV.4: Proposed STAT & FeaR-STAT Architectures Performance Scores using C-MAPSS Dataset with pre-selected sensors

Model	**Scores**			
	FD001	**FD002**	**FD003**	**FD004**
STAT	184	1098	189	1824
FeaR-STAT	200	1023	199	1587

Table IV.5: Proposed STAT & FeaR-STAT Architectures Performance Scores using C-MAPSS Dataset with complete set of sensors

Model	**Scores**			
	FD001	**FD002**	**FD003**	**FD004**
STAT	196	968	185	1919
FeaR-STAT	206	1115	182	1639

Table IV.6: Training time for 120 Epochs with STAT and FeaR-STAT architectures using C-MAPSS Dataset.

Model	**Training time (mins)**			
	FD001	**FD002**	**FD003**	**FD004**
STAT	7.5	24.2	9.24	22.1
FeaR-STAT	6.6	22.8	8.4	20.4

Table IV.7: Inference time for one forward pass with STAT and FeaR-STAT architectures using C-MAPSS Dataset.

Model	**Inference time (sec)**			
	FD001	**FD002**	**FD003**	**FD004**
STAT	0.03	0.036	0.035	0.024
FeaR-STAT	0.023	0.035	0.03	0.025

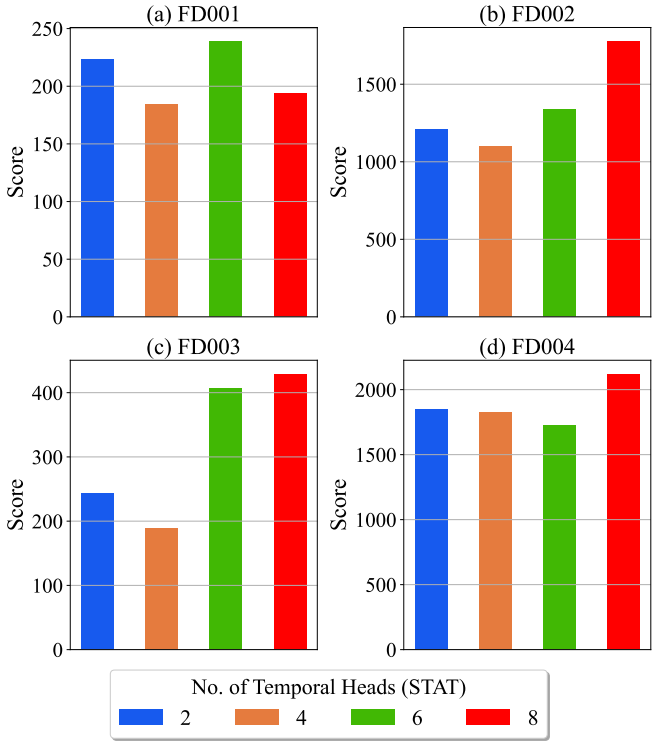


Figure IV.7: Effect of Temporal Heads *(heads$_t$)* in proposed STAT Architecture.

IV.6.6 Results on Hyperparameter Sensitivity

We start the experimental results by analysing the performance of STAT and FeaR-STAT with different hyperparameter settings on the C-MAPSS dataset with pre-selected sensors. The hyperparameters include the number of temporal and feature heads, encoder and decoder hidden sizes, number of encoder and decoder layers and optimizer warmup rate. Finally, we compare the results with existing approaches in the literature.

Effect of Temporal Heads and Feature Heads

The impact of changing the number of Temporal Heads, $heads_t$ in the STAT and the FeaR-STAT models is shown in Fig. IV.7 and in Fig. IV.8 respectively. In both the cases, keeping all other hyperparameters constant, a $heads_t$ of 4 results in the overall best performance across all the sub-datasets in both models. Further increasing $heads_t$ degrades overall performance for both the STAT and FeaR-STAT architectures.

However, by analysing the effect of the number of Feature Heads, $heads_f$ in the STAT model, it can be observed from Fig. IV.9 that increasing the number of $heads_f$ up to 16 leads to an improvement in the overall prediction performance. A further increase in $heads_f$ highly deteriorates overall results especially in FD001 and FD003 sub-datasets. It is to be noted here that Feature Heads is required only in the decoder part of the STAT model. Fig. IV.10 shows the effect of the $heads_f$ in the FeaR-STAT model. It is evident from Fig. IV.10 that increasing $heads_f$ does not assist in the model performance, rather keeping them at 8 results in the best overall performance across all the sub-datasets.

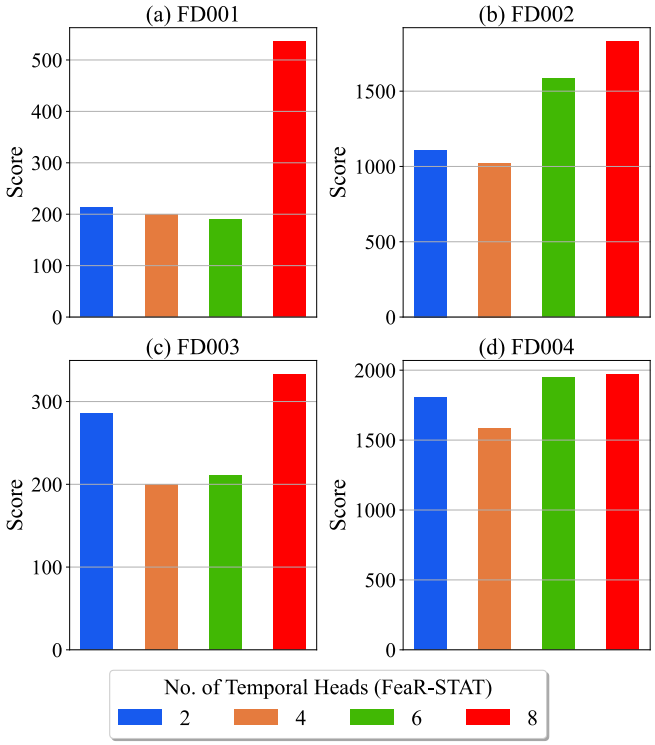


Figure IV.8: Effect of Temporal Heads *(heads$_t$)* in proposed FeaR-STAT Architecture.

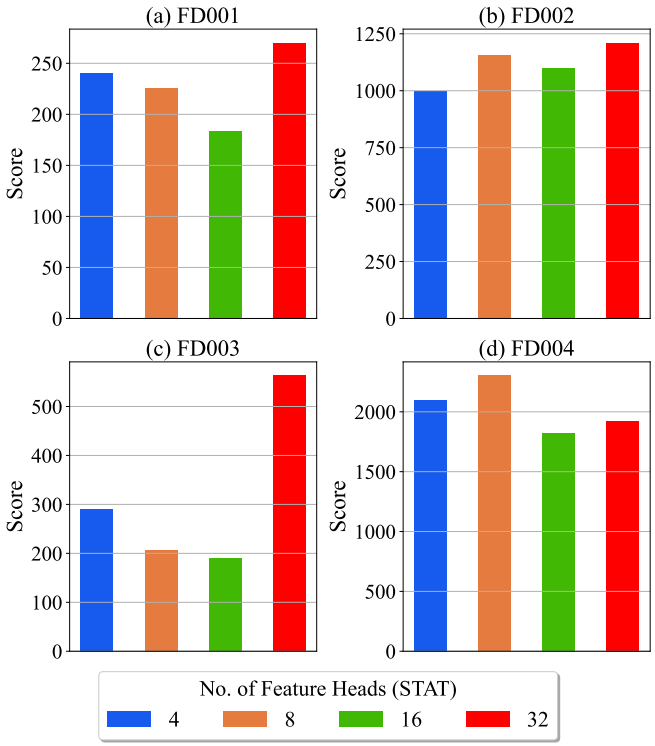


Figure IV.9: Effect of Feature Heads *(heads$_f$)* in proposed STAT Architecture.

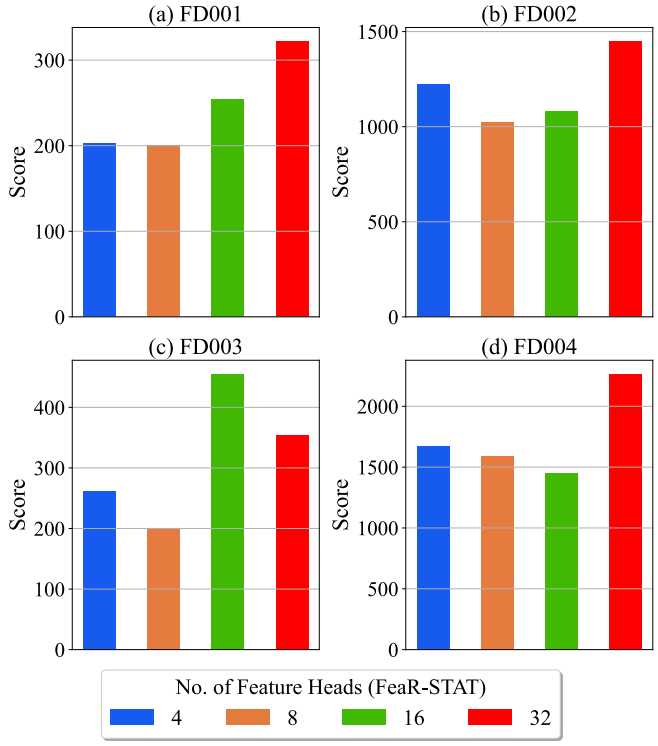


Figure IV.10: Effect of Feature Heads *(heads$_f$)* in proposed FeaR-STAT Architecture.

Effect of Encoder and Decoder Hidden Size

Increasing the encoder-decoder hidden sizes *(d_e, d_d)* declines the performance of the STAT model in all four sub-datasets as shown in Fig. IV.11 and Fig. IV.13. The deep architecture of STAT provides enough model complexity to learn complex features from the input data and generate corresponding RULs. Hence, larger hidden sizes may not be necessary for improving the model performance. A similar behaviour can be observed in the Fear-STAT model from Fig. IV.12 and Fig. IV.14 where increasing the encoder-decoder hidden size results in a deterioration in the performance.

Effect of No. of Encoder & Decoder Layers

The STAT model achieves on average the best performance by using the least number of stacked encoder and decoder layers, L_{ED} as shown in Fig. IV.15. Since the stacked layers do not share weights, hence increasing L_{ED} proportionally increases the number of trainable parameters and eventually plummets the model performance. The use of least number of trainable parameters is one of the biggest attributes of the STAT architecture. The performance drop is even more drastic in the case of the FeaR-STAT model as illustrated in Fig. IV.16, where the performance sharply plummet due to increasing the number of stacked encoder and decoder layers.

Effect of Optimizer Warmup Rate

The $W_{opt}\%$, plays quite a significant role in the performance of the STAT architecture. As shown in Table IV.8, a $W_{opt}\%$ of 50% for LR scheduling, results in the overall best

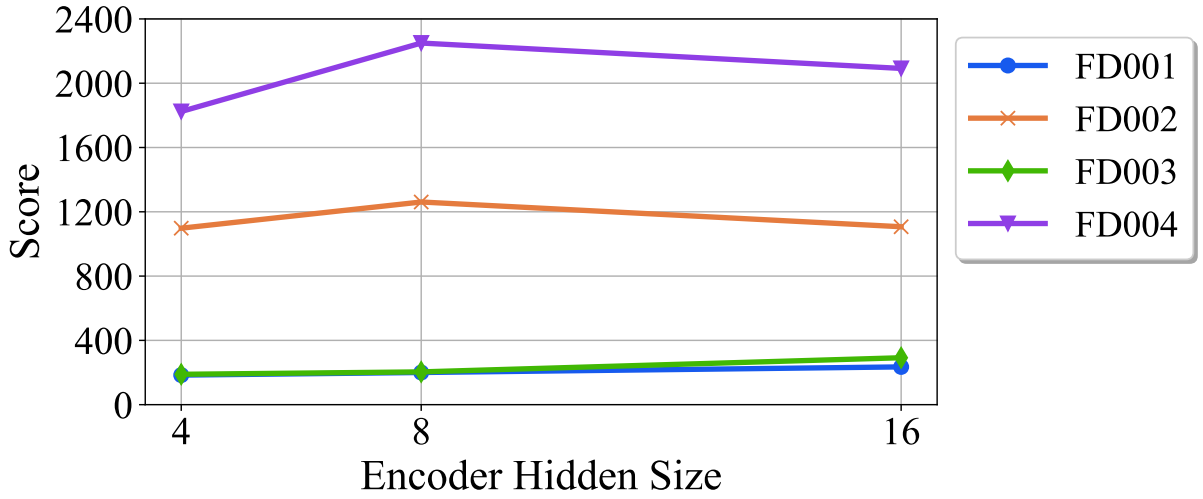


Figure IV.11: Effect of Encoder Hidden Size (d_e) in proposed STAT Architecture.

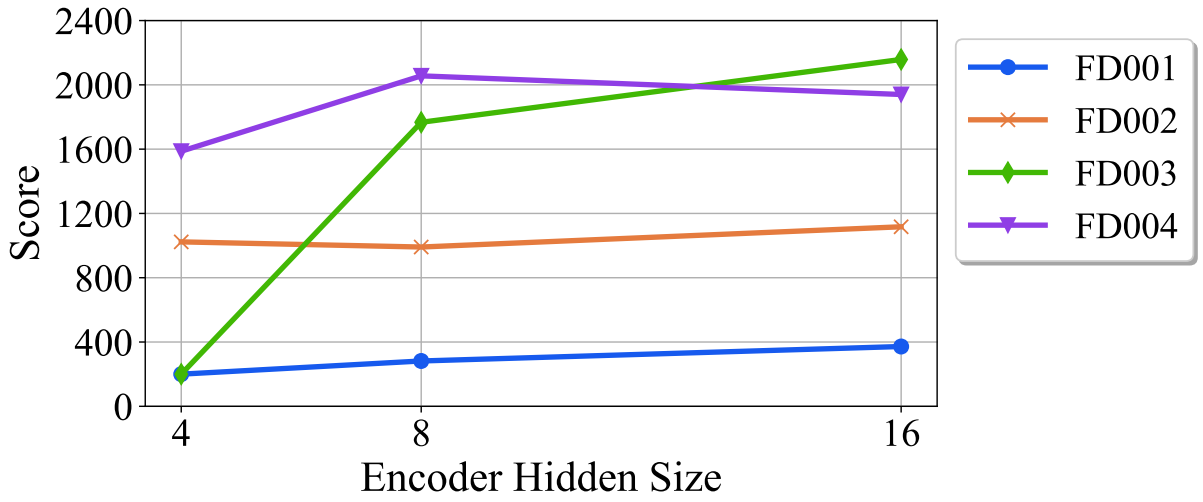


Figure IV.12: Effect of Encoder Hidden Size (d_e) in proposed FeaR-STAT Architecture.

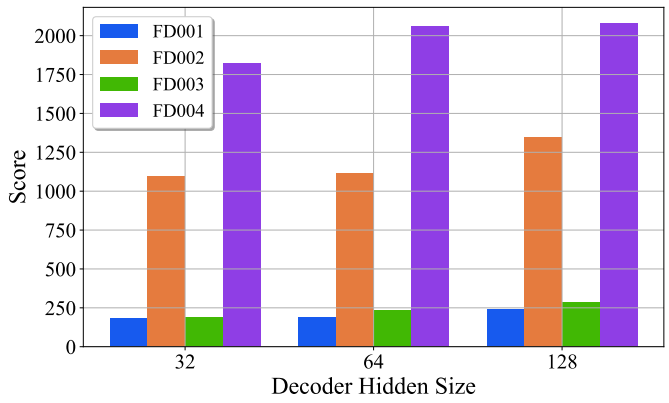


Figure IV.13: Effect of Decoder Hidden Size (d_d) in proposed STAT Architecture.

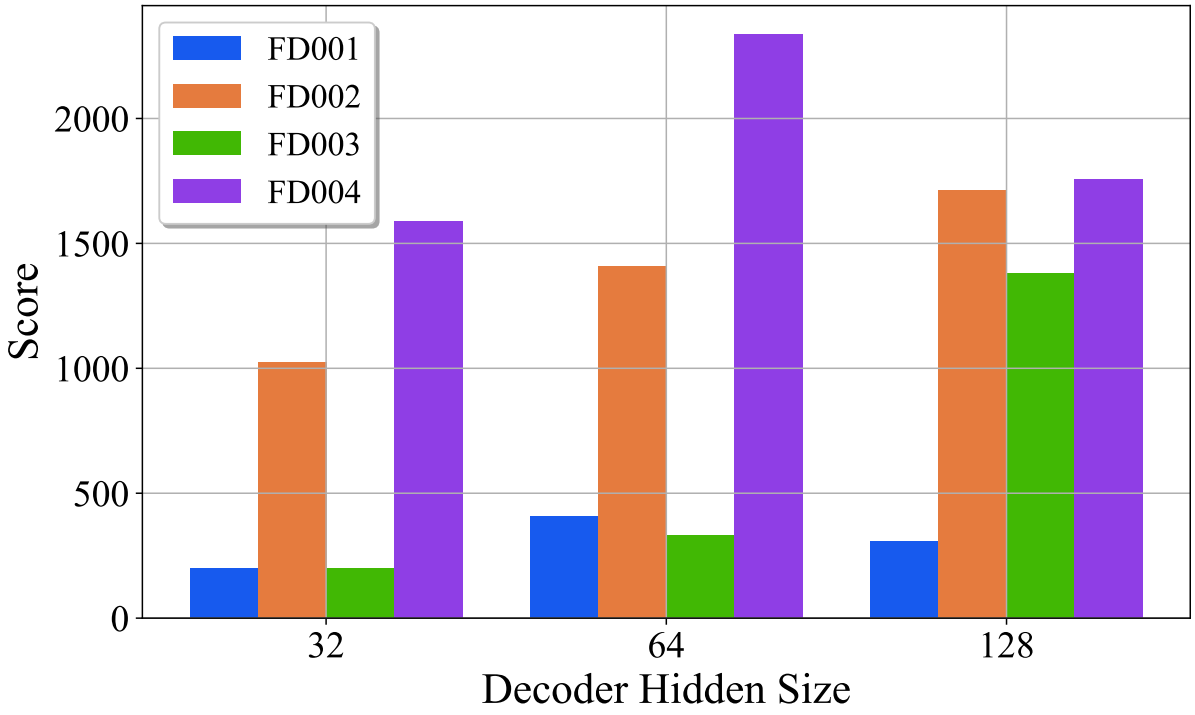


Figure IV.14: Effect of Decoder Hidden Size (d_d) in proposed FeaR-STAT Architecture.

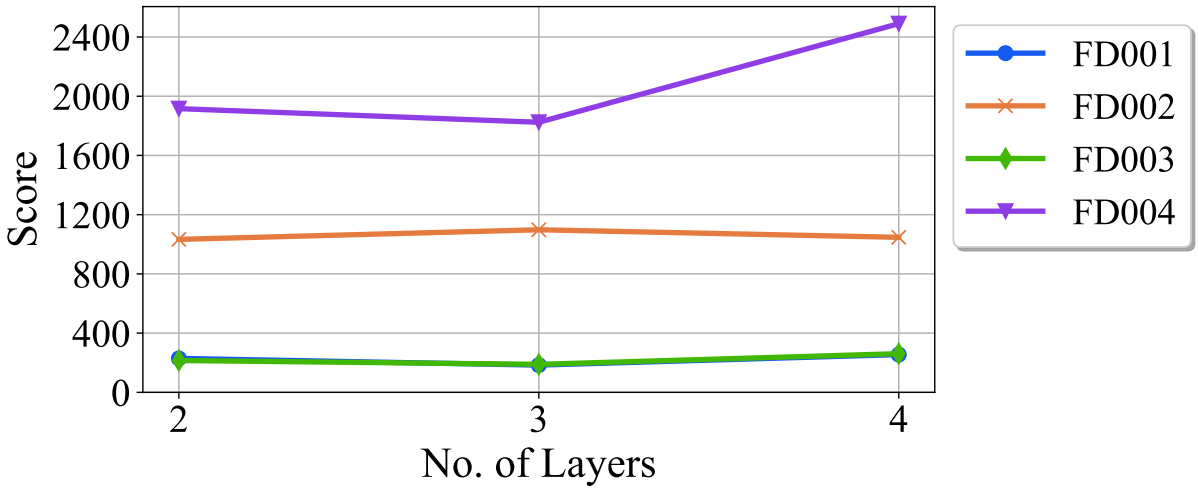


Figure IV.15: Effect of no. of Layers (L_{ED}) in proposed STAT Architecture.

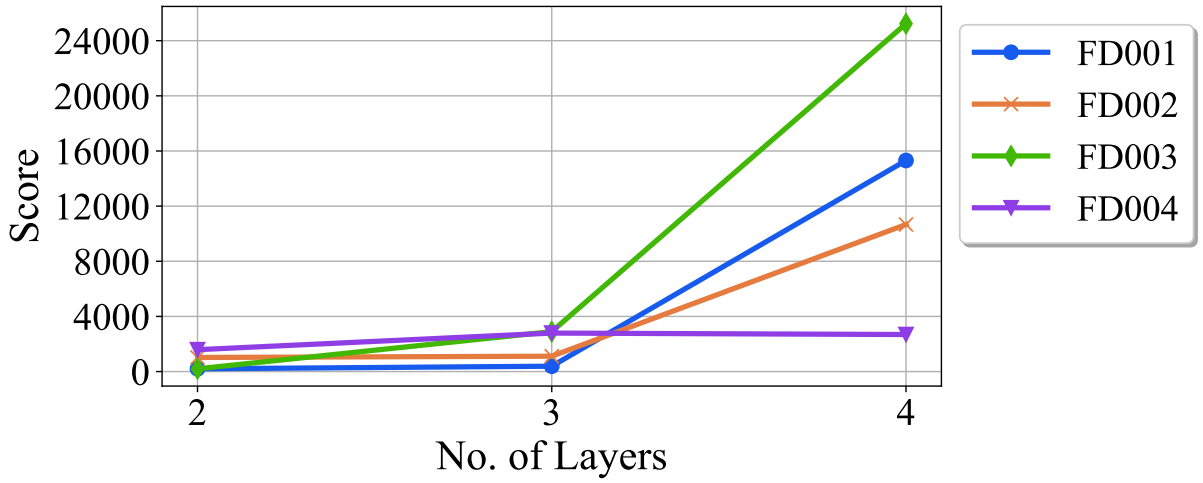


Figure IV.16: Effect of no. of Layers *(L_{ED})* in proposed FeaR-STAT Architecture.

performance across all the other sub-dataset. However, a $W_{opt}\%$ of 30% for *LR* scheduling works best in the case the of the FeaR-STAT model as shown in Table IV.9.

Table IV.8: Comparison of STAT Performance *(Scores)* based on Optimizer Warmup rate

Optimizer Warmup *($W_{opt}\%$)*	**Score**			
	FD001	**FD002**	**FD003**	**FD004**
30	229	1033	215	1916
50	184	1098	189	1824
70	255	1047	261	2490

IV.6.7 Comparison with Related Literature

In this section, we evaluate the sequence RUL predictor models presented in this paper against the literature model performances based on their generated scores (Table IV.10) and RMSE losses (Table IV.11) for the C-MAPSS Turbofan Engine Dataset. In both tables for each sub-dataset, the best values are formatted in bold and the second-best ones are underlined.

As evident in Table IV.10, the proposed STAT and FeaR-STAT models outperform the best literature score [18] for all the sub-datasets, with the exception of FD001 for the FeaR-STAT model. The STAT model strikes a fine balance throughout the sub-datasets. A further improvement from the STAT performance can be observed in the proposed FeaR-STAT, which significantly improves the FD004 score. A thorough observation of Table IV.10 confirms that the proposed STAT and FeaR-STAT outmatches every literature model when providing a highly balanced performance throughout all the C-MAPSS sub-datasets. The Transformer encoder model with Gated CNN [29] does not deploy the scoring function as a performance metric and hence, scores from this literature are not available for comparison.

Table IV.9: Comparison of FeaR-STAT Performance *(Scores)* based on Optimizer Warmup rate

Optimizer Warmup *(W_{opt}%)*	**Score**			
	FD001	**FD002**	**FD003**	**FD004**
30	200	1023	200	1587
50	285	1407	283	2353
70	256	1838	267	2196

A distinctive analysis can be made in Table IV.11 by comparing the models based on their RMSE losses during the testing phase. The RMSE loss is an indication of the model's capability of providing accurate RUL predictions throughout the life cycle of an engine. This reflects how well a model can capture the degradation pattern from the start till the end of life of any machine. However, the RMSE loss is different from the C-MAPSS performance score since, the scoring function depends exclusively on the final RUL cycle of the engine whereas, the RMSE loss function depends on all RUL predictions throughout the life-cycle of the engines. The RMSE for all methods with attention mechanism, which perform similarly to the STAT and FeaR-STAT w.r.t scores are shown in Table IV.11. The STAT and FeaR-STAT transformers fall slightly behind the Transformer-encoder with Gated CNN model [29], DAG [24] and the CNN + Attention [18] for the FD001 RMSE. However, proposed transformer RMSEs outperform in all other operating conditions compared to existing literature underlining the capability of the approaches.

We provide a comparison of the trainable parameters of the architectures. The average score on all the operating conditions in the C-MAPSS dataset in relation to the total number of trainable parameters in the best performing models from literature and our best performing model is shown in Fig. IV.17. The best performing STAT and FeaR-STAT models require similar number of trainable parameters as the ones in literature, but performs remarkably better with an average score of 823.75 and 752.25 respectively. The obtained results are superior to the current state of the art.

Finally, the training duration required by the proposed models and the literature are compared with their corresponding performance scores for the C-MAPSS sub-dataset FD001 in Fig. IV.18. On evaluating this with Fig. IV.17, it is to observe that the computation time of DAG [24] is the least since it consists of fewer trainable parameters. However, the performance score from the proposed models is sufficiently higher than all the models presented in the literature. Furthermore, although the STAT model contains almost the same number of parameters as in AGCNN [25], and FeaR-STAT consists of virtually the same as in DCNN [17], both the proposed models complete training faster and performs better than the compared literature.

IV.7 Conclusion

In this paper we proposed a novel approach for remaining useful lifetime estimation using transformer neural networks. Particularly, we propose two novel transformer blocks,

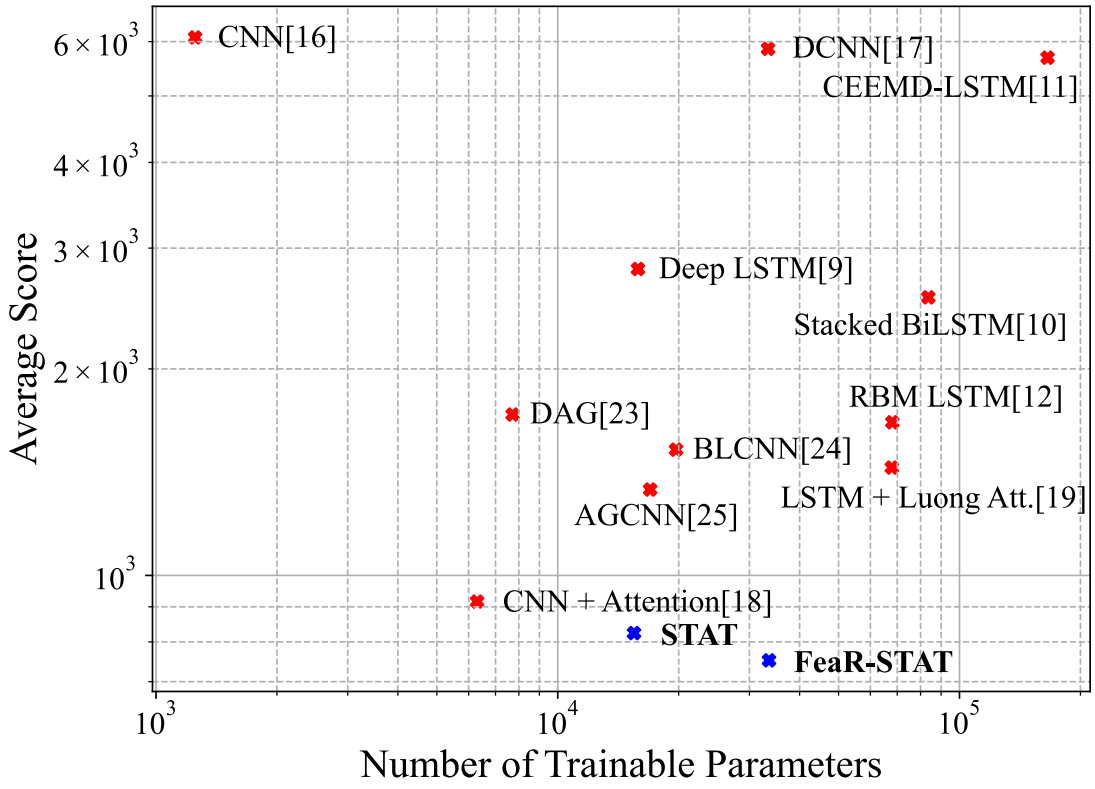


Figure IV.17: Average score of the proposed architectures on the C-MAPSS dataset compared to existing work in literature with respect to the number of trainable parameters.

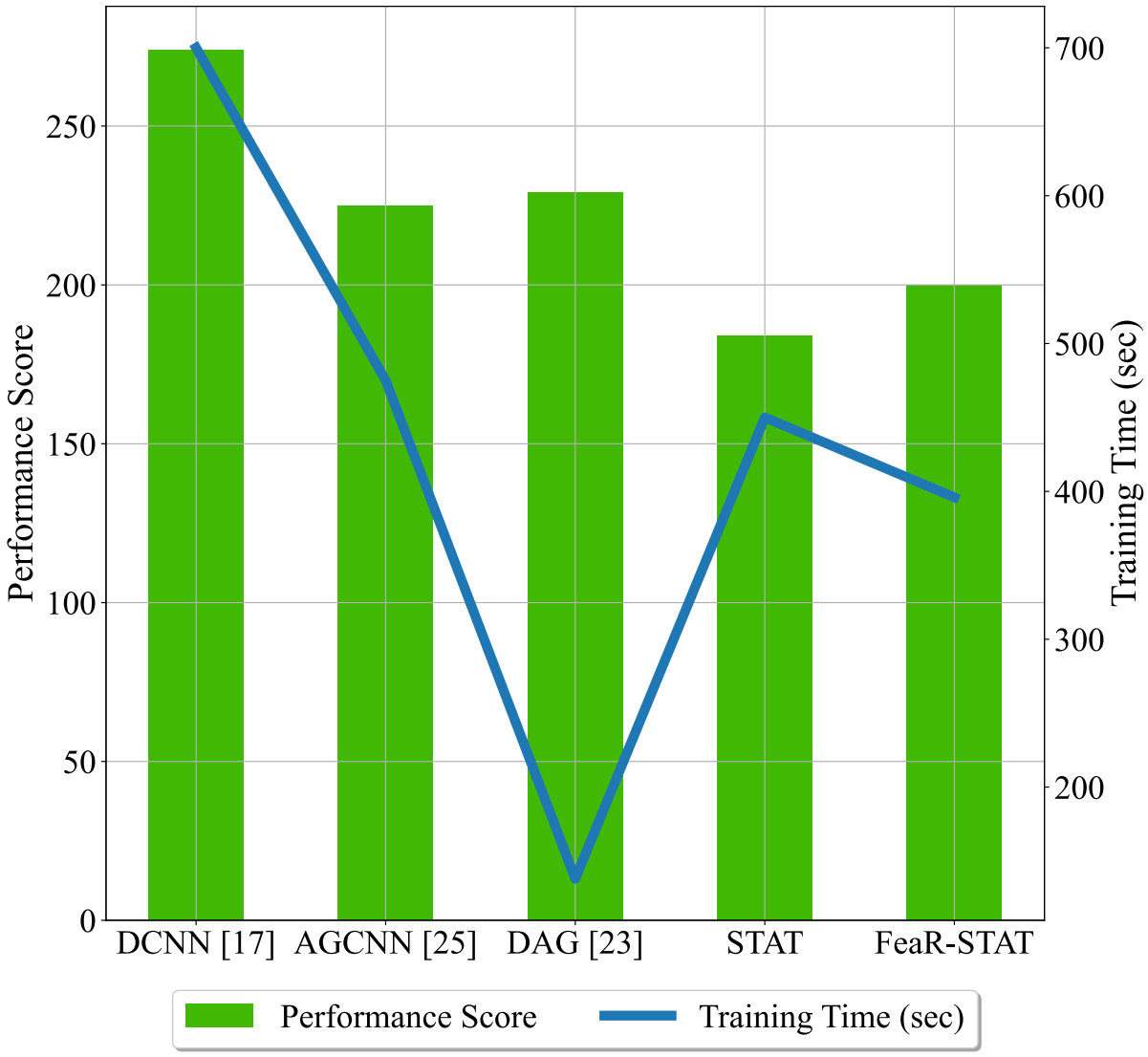


Figure IV.18: Performance Score vs Training Time comparison between proposed STAT and FeaR-STAT architectures and current literature for C-MAPSS sub-dataset FD001.

Table IV.10: Comparison of Model Performance *(Scores)* with Related Literature

Model Description	**Score**			
	FD001	**FD002**	**FD003**	**FD004**
Deep LSTM [9]	338	4450	852	5550
Stacked BiLSTM [10]	295	4130	317	5430
Handcrafted Features + LSTM [15]	322	–	–	5649
CEEMD+DLSTM [11]	262	6953	452	15069
RBM+LSTM [12]	231	3366	251	2840
CNN [16]	1287	13570	1596	7886
CNN+Attention [18]	<u>198</u>	1144	251	2072
HI CNN-LSTM-NN [21]	303	3440	1420	4630
FCLCNN [22]	204	–	234	–
BLCNN [23]	302	1558	381	3859
DAG [24]	229	2730	535	3370
DCNN [17]	274	10412	284	12466
AGCNN (Self-Att.) [25]	225	1492	227	3392
LSTM + Luong Att. [19]	320	2102	223	3100
Transformer Encoder + Gated CNN[29]	–	–	–	–
Proposed STAT	**184**	<u>1098</u>	**189**	<u>1824</u>
Proposed FeaR-STAT	200	**1023**	<u>199</u>	**1587**

namely shared temporal and split-feature attention blocks, which are specifically designed for the analysis of multivariate time series data. We leverage these blocks by proposing shared temporal and feature-represented shared temporal attention transformer used to detect degradation patterns for RUL estimation. We apply both transformer architectures to the C-MAPSS benchmark dataset where both architectures exhibit superior performance compared to the existing state-of-the-art approaches.

In future work, we will further elaborate on the positional encoding technique for the proposed transformer types to mitigate the actual drawbacks for time series analysis. Furthermore, we will test both architectures in other application domains involving multivariate time series analysis including condition monitoring and forecasting and apply them to other dataset as well.

Table IV.11: Comparison of Model Performance *(RMSE)* with related attention literature

Model Description	**RMSE**			
	FD001	**FD002**	**FD003**	**FD004**
DAG [24]	11.96	20.34	12.46	22.43
CNN+Attention [18]	<u>11.48</u>	17.25	12.31	20.58
AGCNN (Self-Attention) [25]	12.42	19.43	13.39	21.50
LSTM+Luong Attention [19]	13.95	17.65	12.72	20.21
Transformer Encoder + Gated CNN[29]	**11.27**	22.81	11.42	24.86
Proposed STAT	12.1	**15.2**	**10.6**	<u>15.54</u>
Proposed FeaR-STAT	12.01	<u>15.5</u>	<u>10.9</u>	**15.03**

Bibliography

[1] K. L. Tsui, N. Chen, Q. Zhou, Y. Hai, and W. Wang, "Prognostics and health management: A review on data driven approaches," *Mathematical Problems in Engineering*, vol. 2015, no. 6, pp. 1–17, 2015.

[2] A. K. Jardine, D. Lin, and D. Banjevic, "A review on machinery diagnostics and prognostics implementing condition-based maintenance," *Mechanical Systems and Signal Processing*, vol. 20, no. 7, pp. 1483–1510, 2006.

[3] F. Ahmadzadeh and J. Lundberg, "Remaining useful life estimation: review," *International Journal of System Assurance Engineering and Management*, vol. 5, no. 4, pp. 461–474, 2014.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[5] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, pp. 396–404, Morgan Kaufmann, 1990.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[9] S. Zheng, K. Ristovski, A. Farahat, and C. Gupta, "Long short-term memory network for remaining useful life estimation," in *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*, pp. 88–95, IEEE, 2017.

[10] J. Wang, G. Wen, S. Yang, and Y. Liu, "Remaining useful life estimation in prognostics using deep bidirectional lstm neural network," in *2018 Prognostics and System Health Management Conference (PHM-Chongqing)*, pp. 1037–1042, 2018.

[11] S. Zhao, Y. Zhang, S. Wang, B. Zhou, and C. Cheng, "A recurrent neural network approach for remaining useful life prediction utilizing a novel trend features construction method," *Measurement*, vol. 146, pp. 279–288, 2019.

[12] A. Listou Ellefsen, E. Bjørlykhaug, V. Æsøy, S. Ushakov, and H. Zhang, "Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture," *Reliability Engineering & System Safety*, vol. 183, pp. 240–251, 2019.

[13] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *3rd International Conference on Learning Representations*, 2015.

[14] Thang Luong, Hieu Pham, and Christopher D. Manning, "Effective approaches to attention-based neural machine translation," in *EMNLP*, 2015.

[15] Z. Chen, M. Wu, R. Zhao, F. Guretno, R. Yan, and X. Li, "Machine Remaining Useful Life Prediction via an Attention-Based Deep Learning Approach," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 3, pp. 2521–2531, 2021.

[16] G. Sateesh Babu, P. Zhao, and X.-L. Li, "Deep convolutional neural network based regression approach for estimation of remaining useful life," in *Database Systems for Advanced Applications*, vol. 9642 of *Lecture Notes in Computer Science*, pp. 214–228, Cham: Springer International Publishing, 2016.

[17] X. Li, Q. Ding, and J.-Q. Sun, "Remaining useful life estimation in prognostics using deep convolution neural networks," *Reliability Engineering & System Safety*, vol. 172, pp. 1–11, 2018.

[18] W. M. Tan and T. H. Teo, "Remaining useful life prediction using temporal convolution with attention," *AI*, vol. 2, no. 1, pp. 48–70, 2021.

[19] P. Costa, A. Akcay, Y. Zhang, and U. Kaymak, "Attention and Long Short-Term Memory Network for Remaining Useful Lifetime Predictions of Turbofan Engine Degradation," *International Journal of Prognostics and Health Management*, vol. 10, p. 12, 2020.

[20] H. Zhang, Q. Zhang, S. Shao, T. Niu, and X. Yang, "Attention-based lstm network for rotatory machine remaining useful life prediction," *IEEE Access*, vol. 8, pp. 132188–132199, 2020.

[21] Z. Kong, Y. Cui, Z. Xia, and H. Lv, "Convolution and long short-term memory hybrid deep neural networks for remaining useful life prognostics," *Applied Sciences*, vol. 9, no. 19, 2019.

[22] C. Peng, Y. Chen, Q. Chen, Z. Tang, L. Li, and W. Gui, "A remaining useful life prognosis of turbofan engine using temporal and spatial feature fusion," *Sensors (Basel, Switzerland)*, vol. 21, no. 2, 2021.

[23] H. Liu, Z. Liu, W. Jia, and X. Lin, "A novel deep learning-based encoder-decoder model for remaining useful life prediction," in *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2019.

[24] 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.

[25] H. Liu, Z. Liu, W. Jia, and X. Lin, "Remaining Useful Life Prediction Using a Novel Feature-Attention-Based End-to-End Approach," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 2, pp. 1197–1207, 2021.

[26] N. Kitaev, L. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," in *International Conference on Learning Representations*, 2020.

[27] J. Pfeiffer, A. Rücklé, C. Poth, A. Kamath, I. Vulić, S. Ruder, K. Cho, and I. Gurevych, "Adapterhub: A framework for adapting transformers," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, (Online), pp. 46–54, Association for Computational Linguistics, 2020.

[28] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[29] Y. Mo, Q. Wu, X. Li, and B. Huang, "Remaining useful life estimation via transformer encoder enhanced by a gated convolutional unit," *Journal of Intelligent Manufacturing*, 2021.

[30] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[31] Y. Huang and Y. Yu, "An internal covariate shift bounding algorithm for deep neural networks by unitizing layers' outputs," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8462–8470, 2020.

[32] D. K. Frederick, J. A. DeCastro, and J. S. Litt, "User's guide for the commercial modular aero-propulsion system simulation (c-mapss)," 2007.

[33] 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*, pp. 1–9, IEEE, 06/10/2008 - 09/10/2008.

[34] Santanu Chatterjee and Jonathan Litt, "Online model parameter estimation of jet engine degradation for autonomous propulsion control," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2013.

[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: The international journal of advanced manufacturing technology, 101(1-4), 87-103," *The International Journal of Advanced Manufacturing Technology*, vol. 101, no. 1-4, pp. 87–103, 2019.

[36] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

[37] X. Glorot and Y. Bengio, "Understanding the Difficulty of Training Deep Feedforward Neural Networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, vol. 9 of *Proceedings of Machine Learning Research*, pp. 249–256, PMLR, 13–15 May 2010.

Complete List of Publications

Journal Papers

1. **G. S. Chadha**, S.R.B. Shah, A. Schwung, and S. X. Ding, "Shared Temporal Attention Transformer for Remaining Useful Lifetime Estimation," *IEEE Access*, vol. 10, pp. 74244–74258, 2022.

2. J. Pöppelbaum, **G. S. Chadha**, and A. Schwung, "Contrastive learning based self-supervised time-series analysis," *Applied Soft Computing*, vol. 10-11, pp. 108397, 2022.

3. S.R.B. Shah, **G. S. Chadha**, A. Schwung, and S. X. Ding, "A Sequence-to-Sequence Approach for Remaining Useful Lifetime Estimation Using Attention-augmented Bidirectional LSTM," *Intelligent Systems with Applications*, vol. 10-11, pp. 200049, 2021.

4. M.S.A Hameed, **G. S. Chadha**, A. Schwung, and S. X. Ding, "Gradient Monitored Reinforcement Learning," *IEEE transactions on neural networks and learning systems*, 2021.

5. **G. S. Chadha**, U. Panara, A. Schwung, and S. X. Ding, "Generalized dilation convolutional neural networks for remaining useful lifetime estimation," *Neurocomputing*, vol. 452, pp. 182–199, 2021.

6. **G. S. Chadha**, I. Islam, A. Schwung, and S. X. Ding, "Deep Convolutional Clustering-Based Time Series Anomaly Detection," *Sensors*, vol. 21(16), 5488, 2021.

7. **G. S. Chadha**, A. Panambilly, A. Schwung, and S. X. Ding, "Bidirectional deep recurrent neural networks for process fault classification," *ISA transactions*, vol. 106, pp. 330–342, 2020.

Conference Papers

1. **G. S. Chadha**, J.N. Reimann and A. Schwung, "Generalized Dilation Structures in Convolutional Neural Networks," *In Proceedings of the 10th International Conference on Pattern Recognition Applications and Methods - Volume 1: ICPRAM*, 2021, pp 79-88.

2. **G. S. Chadha**, J. Kim, A. Schwung, and S. X. Ding, "Permutation Learning in Convolutional Neural Networks for Time-Series Analysis," in *Lecture Notes in Computer Science, Artificial Neural Networks and Machine Learning – ICANN 2020, Cham: Springer International Publishing*, 2020, pp. 220–231.

3. **G. S. Chadha**, S. M. Nazmus Sakeib, and A. Schwung, "Remaining Useful Lifetime Estimation with Sobolev Training," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2020, pp. 1447–1450.

4. **G. S. Chadha**, M. Krishnamoorthy, and A. Schwung, "Time Series based Fault Detection in Industrial Processes using Convolutional Neural Networks," in *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, 2019, pp. 173–178.

5. **G. S. Chadha**, A. Rabbani, and A. Schwung, "Comparison of Semi-supervised Deep Neural Networks for Anomaly Detection in Industrial Processes," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, 2019, pp. 214–219.

6. **G. S. Chadha**, E. Meydani, and A. Schwung, "Regularizing Neural Networks with Gradient Monitoring," in *Proceedings of the International Neural Networks Society, Recent Advances in Big Data and Deep Learning (INNSBDDL 2019), Cham: Springer International Publishing*, 2019, pp. 196–205.

7. P. Rehlaender, M. Schroeer, **G. S. Chadha**, and A. Schwung, "Traffic Sign Detection Using R-CNN," in *Proceedings of the International Neural Networks Society, Recent Advances in Big Data and Deep Learning (INNSBDDL 2019), Cham: Springer International Publishing*, 2019, pp. 226–235.

8. **G. S. Chadha**, F. Westbrink, T. Schütte, and A. Schwung, "Optimal dosing of bulk material using mass-flow estimation and DEM simulation," *in 2018 IEEE International Conference on Industrial Technology (ICIT)*, 2018, pp. 256–261.

9. F. Westbrink, **G. S. Chadha**, and A. Schwung, "Integrated IPC for data-driven fault detection," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, 2018, pp. 277–282.

10. **G. S. Chadha** and A. Schwung, "Comparison of deep neural network architectures for fault detection in Tennessee Eastman process," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2017, pp. 1–8.

DOI: 10.17185/duepublico/82327
URN: urn:nbn:de:hbz:465-20240821-133721-0