

**Prediction of Remaining Useful Life (RUL) of Commercial
Jet Engine using Deep Neural Network**
A PROJECT REPORT

Submitted by

Jeyaraman S (810020106034)

Kakini M (810020106036)

Kiruthiga J (810020106042)

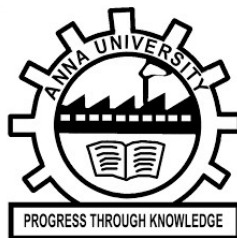
in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

In

ELECTRONICS AND COMMUNICATION ENGINEERING



UNIVERSITY COLLEGE OF ENGINEERING, BIT CAMPUS,

TIRUCHIRAPPALLI - 620024

ANNA UNIVERSITY: CHENNAI - 600 025

MAY 2024

BONAFIDE CERTIFICATE

Certified that this project report "**Prediction of Remaining Useful Life (RUL) of Commercial Jet Engine using Deep Neural Network**" is the work of "**Jeyaraman S (810020106034), Kakini M (810020106036), Kiruthiga J (810020106042)**" who carried out the project work under my supervision.

SIGNATURE

Dr. P. RAMADEVI,
Associate Professor,
HEAD OF THE DEPARTMENT

Department of Electronics and
Communication Engineering,
University College of Engineering,
BIT Campus,
Tiruchirappalli - 620024

SIGNATURE

Mrs. M. AZHAGU SUBHA,
Teaching Fellow,
SUPERVISOR

Department of Electronics and
Communication Engineering,
University College of
Engineering, BIT Campus,
Tiruchirappalli – 620024

Submitted for “EC8611 – Project Work” in B.E of Electronics and
Communication Engineering Jan – May 2024 Examination held on

Internal Examiner

External Examiner

DECLARATION

We hereby declare that the work entitled " **Prediction of Remaining Useful Life (RUL) of Commercial Jet Engine using Deep Neural Network** " submitted in partial fulfillment of the requirement for the award of the degree in B.E., University College of Engineering, BIT Campus, Anna University, Tiruchirappalli is a record of my work carried out by me during the academic year 2023- 2024 under the supervision of **Mrs. M. AZHAGU SUBHA** Teaching Fellow, Department of Electronics and Communication Engineering, University College of Engineering, Anna University, BIT Campus, Tiruchirappalli. The extent and source of information are derived from the existing literature and have been indicated through the dissertation at the appropriate places. The matter embodied in this work is original and has not been submitted for the award of any other degree or diploma, either in this or any other university.

JEYARAMAN S (810020106034)

KAKINI M (810020106036)

KIRUTHIGA J (810020106042)

I certify that the declaration made by the above candidates is true.

SIGNATURE OF THE SUPERVISOR

ACKNOWLEDGEMENT

Our first and the foremost gratitude to almighty and our parents who showered the blessing and gave us the knowledge and strength to perform our work.

We wish to express our heartily and sincerely thanks to our beloved **Dean Dr. T. SENTHILKUMAR**, University College of Engineering- BIT campus for generously providing us with excellent facilities throughout the course of study and especially for this project.

We owe our deep sense and profound gratitude to Head of the ECE Department **Dr. P. RAMADEVI**, University College of Engineering- BIT campus for her through put provoking, painstaking guidance and supervision at every stage of our study.

We thankfully acknowledge the abiding interest of our project guide **Mrs. M. AZHAGU SUBHA**, Teaching Fellow, Department of ECE University College of Engineering- BIT campus who took this project upon her in all status and worked very closely with all members of our project.

We also express our hearty thanks to the project coordinators of ECE Department for their encouragement and support to complete the project work successfully.

JEYARAMAN S (810020106034)

KAKINI M (810020106036)

KIRUTHIGA J (810020106042)

Abstract

Remaining Useful Life prediction of commercial jet engines can be achieved using LSTMDNN, where LSTM (Long- Short Term Memory) is a type of RNNs Recurrent Neural Network (RNN) that can detain long-term dependencies in sequential data. LSTMs are capable to process and analyse sequential data, such as time series, text, and speech for Natural Language Processing (NLP), speech recognition, and time series forecasting. So, LSTM techniques can be applied in the Indian commercial jet named Airbus A320 to predict the remaining useful life from its lifetime of 60,000 flight hours with a speed of 830 km/h. In traditional methods for prediction of remaining useful life of commercial jets uses several conventional approaches such as Markov chains, Fault Tree Analysis, and Analytic Hierarchy Process. However, these methods commonly suffer from high false positive rates (FPR) and false negative rates (FNR), which can be critical and pose a risk to the aircraft. Another traditional method, namely Prognostics and Health Management (PHM) systems is a complex model that demands maintenance, susceptibility to false positives and negatives, and cost implications. The LSTM-DNN methodology emphasizes simplicity, reduces model complexity and minimize false predictions. To validate the proposed method experimentally, we utilize the C-MAPSS aero engine dataset, which comprises data gathered from 21 sensors, including temperature, pressure, and speed sensors, alongside 18 sensors specifically employed for calculating the Remaining Useful Life (RUL). The Temperature sensors such as IAT, EGT, ECT, Oil Temperature, and Turbocharger Temperature Sensors are integrated, each with specific ranges ensuring accurate data acquisition. For instance, IAT operates typically between -40°C to 125°C , EGT in the range of 500°C to $1,200^{\circ}\text{C}$ or higher, and ECT between -40°C to 150°C . Pressure sensors, including MAP, Boost Pressure, EBP, Fuel Rail Pressure and Oil Pressure Sensors, play a pivotal role. MAP sensors measure pressures from about 15 kPa

to 250 kPa, while Boost Pressure sensors typically range from 100 kPa to 300 kPa or higher. Speed sensors, such as Turbocharger Speed, Wheel Speed, and Transmission Speed Sensors, monitor rotational speeds crucial for engine performance. Such speed sensor values might range from a few 1000 RPM to 10000 of RPM. The Quality of the trained LSTM model is assessed by calculating the false positives and false negatives. Evaluation metrics such as precision, recall, F1 score, and accuracy of remaining useful life (RUL) prediction to assess the model quantitatively. This technique can be applied to similar commercial jets, including the Airbus A380, Boeing 737, Airbus A300, Boeing 777, Airbus A321, Antonov, and Beechcraft 1300.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	v
	LIST OF TABLES	ix
	LIST OF FIGURES	x
	LIST OF SYMBOLS(ACRONYMS)	xi
1	INTRODUCTION	1
	1.1 Overview	1
	1.2 Importance	2
	1.3 Existing methods	2
	1.4 Limitations	3
2	LITERATURE SURVEY	4
	2.1 Conclusion of Literature Survey	14
3	SYSTEM ANALYSIS	16
	3.1 Existing Approach	16
	3.2 Limitations of Existing System	17
	3.3 Proposed System	18
	3.4 Steps in Proposed Approach	18
	3.5 Advantages of Proposed System	18
	3.6 Data Flow Diagram	18

	3.7 Module Diagram	21
	3.7.1 Dataset Collection	21
	3.7.2 Data Description	22
	3.7.3 Data Preprocessing	25
	3.7.4 Data Exploration	25
	3.7.5 Data Visualization	26
	3.8 Feature Engineering	30
	3.9 Model Building	32
	3.9.1 Existing Approach	32
	3.9.1.1 Performance metrics	36
	3.9.2 Proposed Approach	39
4	RESULTS AND DISCUSSIONS	42
5	CONCLUSION AND FUTURE WORKS	46
	REFERENCES	48
	APPENDICES	51

LIST OF TABLES

TABLE NO.	TITLE	PAGE NO
3.1	Data description of turbofan engine sensors	24
3.2	Sensor trends summary	30
4.1	Value of LSTM	42
4.2	All Features Without Historical Data	43
4.3	Data Without Useless Sensors, Without Historical Data	43
4.4	Data Without Useless Sensors, With Historical Data	44

LIST OF FIGURES

Fig.NO.	Title	Page no
1.1	Illustration of Commercial Jet Engine	1
3.1	DFD level 0	19
3.2	DFD level 1	21
3.3	Module Diagram	24
3.4	Time vs Cycle plot	25
3.5-3.25	Sensor 1 - 21	26-29
3.26	Selected Sensors	29
3.27	Lag Feature	31
3.28	Poly Feature	32
3.29	Architecture of Random Forest	33
3.30	Architecture of Support Vector Regression	34
3.31	Architecture of Linear Regression	35
3.32	All Features Without Historical Data	36
3.33	Data Without Useless Sensors, Without Historical Data	37
3.34	Data Without Useless Sensors, With Historical Data	38
3.35	Deep Neural Network	39
3.36 (a)	LSTM Model	40
3.36 (b)	LSTM Model	41
4.1	Value of LSTM	45

LIST OF ACRONYMS

NASA	National Aeronautics and Space Administration
LSTM	Long Short-Term Memory
CNN	Convolutional Neural Network
DFD	Data Flow Diagram
DL	Deep Learning
RUL	Remaining Useful Life
Psia	Pounds Per Square Inch
Rpm	Revaluations Per Unit
C-MAPSS	Commercial Modular Aero-System Propulsion Simulation
LR	Linear Regression
SVR	Support Vector Regressor
RFR	Random Forest Regression

CHAPTER 1

INTRODUCTION

1.1 Overview

The rapid advancement of deep learning techniques has played a significant role in the evolution of engine health management technology. Effective maintenance of flight-critical components including gas Commercial Jet engines plays a significant role in the aircraft industry. Predicting the remaining useful life of the Commercial Jet Engine is an important research area to avoid downtime and failures. Predictive Maintenance is the process of predicting malfunctions using data from equipment monitoring and process performance measurements. The conventional approach to monitoring a Commercial Jet engine's performance is unreliable and more prone to failure. Therefore, this project focuses on designing intelligent decision support for monitoring the Commercial Jet engine performance using a deep learning LSTM algorithm.

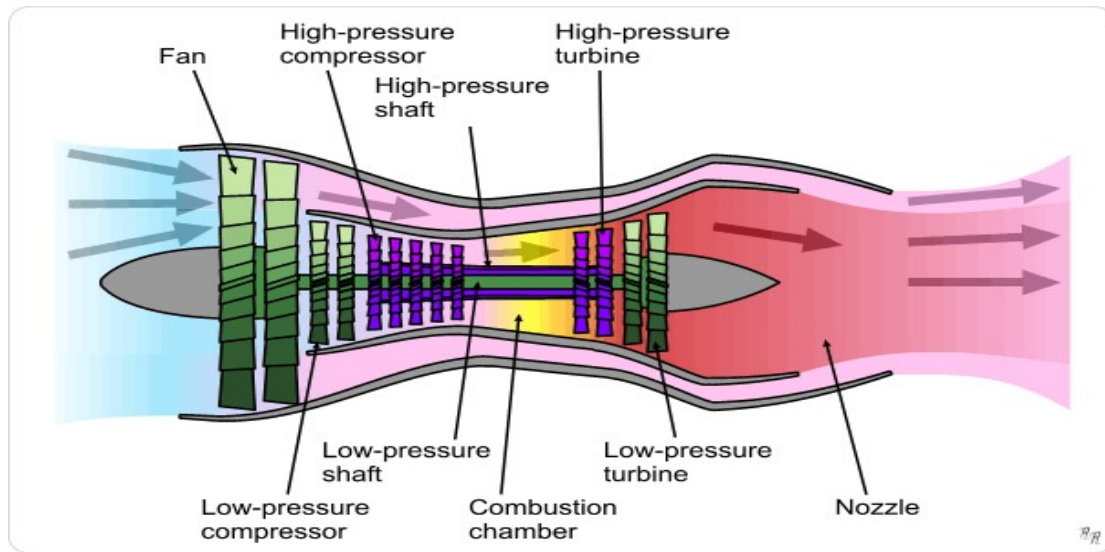


Fig 1.1 Illustration of Commercial Jet Engine

During their lifetime, aircraft components are susceptible to degradation, which directly affects their reliability and performance. This deep learning project will be directed to provide a framework for predicting the aircraft's remaining useful life (RUL) based on the entire life cycle data in order to provide the necessary maintenance behavior.

1.2 Importance

Predicting Remaining Useful Life (RUL) is a critical problem in the maintenance and operation of complicated equipment such as the Commercial Jet engine. Based on available information about a system's current status and prior performance, RUL prediction estimates the remaining time until it fails.

Cost savings: By predicting RUL, operators can more effectively schedule maintenance activities, avoiding the need for costly unplanned repairs and downtime. It can also help to extend the engine's useful life, avoiding the need for costly replacements.

Safety: Accurate RUL prediction can help avert catastrophic failures, which could jeopardize the aircraft's and its occupants' safety.

Efficiency: Operators can optimize engine performance and reduce fuel usage by forecasting RUL, resulting in cost savings and lower emissions.

Reliability: RUL prediction can help increase engine reliability by recognizing possible faults before they cause breakdowns.

1.3 Existing methods

The Existing method of monitoring the performance of a Commercial Jet Engine is unreliable and more prone to failure.

1.4 Limitations

They may not capture the underlying physics and mechanics that cause engine degradation. They require a huge amount of high-quality training data to perform accurately.

They may not generalize well to new or unknown operating circumstances or rare and complex failure modes.

CHAPTER 2

Literature Survey

[1] The paper titled "**Aircraft Engine Performance Monitoring and Diagnostics Based on Deep Convolutional Neural Networks**" was published in **2021**. The paper proposes a method to monitor and diagnose the performance of aircraft engines using deep convolutional neural networks (CNNs). The proposed method utilizes sensor data collected from aircraft engines to train a CNN model. The model is designed to learn the complex relationships between the sensor data and the performance of the engine. The trained model is then used to detect engine faults and predict engine performance. The paper includes experimental results that demonstrate the effectiveness of the proposed method in detecting engine faults and predicting engine performance. The results show that the proposed method outperforms traditional methods for engine performance monitoring and diagnostics in terms of accuracy. One advantage of the proposed method is that it can handle large amounts of data and learn complex relationships between the sensor data and engine performance. Additionally, the proposed method can provide early detection of engine faults, which can lead to improved maintenance and reliability of aircraft engines. One potential disadvantage of the proposed method is that it requires a significant amount of labeled data to train the CNN model effectively. Additionally, the complexity of the CNN model can make it computationally expensive, which can limit the practicality of the proposed method in real-time applications.

Overall, the paper provides a promising approach for monitoring and diagnosing the performance of aircraft engines using deep CNNs. The proposed method has the potential to improve the maintenance and reliability of aircraft engines, leading to reduced costs and increased safety.

[2] The paper titled "**Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture**" was published in **2019**. The paper proposes a method to predict the remaining useful life (RUL) of turbofan engines using a semi-supervised deep architecture. The proposed method utilizes sensor data collected from turbofan engines to train a deep architecture model. The model is trained in a semi-supervised manner, where only a small portion of the training data is labeled. The remaining unlabeled data is used to improve the model's ability to generalize to unseen data. The paper includes experimental results that demonstrate the effectiveness of the proposed method in predicting the RUL of turbofan engines. The results show that the proposed method outperforms several traditional machine learning approaches and achieves high accuracy in predicting the RUL of turbofan engines. Overall, the paper provides a promising approach for predicting the RUL of turbofan engines using a semi-supervised deep architecture. The proposed method has the potential to improve the maintenance and reliability of turbofan engines, leading to reduced costs and increased safety.

The paper titled "Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture" does not report the error rate or accuracy score for the proposed method. However, the paper presents experimental results that demonstrate the effectiveness of the proposed method in predicting the remaining useful life (RUL) of turbofan engines.

The proposed method outperforms several traditional machine learning approaches in terms of RUL prediction accuracy. The authors also highlight the advantages of using a semi-supervised deep architecture for RUL prediction, which allows the model to learn from labeled and unlabeled data and improves its ability to generalize to unseen data. One potential disadvantage of the proposed method is that it requires a large amount of sensor data to train the deep architecture model effectively. Additionally, the process of collecting labeled data for training can be time-consuming and expensive, which can limit the

practicality of the proposed method in real-world settings. Overall, the paper provides a promising approach for predicting the RUL of turbofan engines using a semi-supervised deep architecture. While there are potential disadvantages to the proposed method, its high accuracy in RUL prediction could lead to improved maintenance and reliability of turbofan engines.

[3] The paper titled "**Prediction of Remaining Useful Lifetime (RUL) of turbofan engine using machine learning**" was published in 2019. The paper proposes a method to predict the remaining useful lifetime (RUL) of turbofan engines using machine learning.

The proposed method utilizes sensor data collected from turbofan engines to train a machine-learning model. The model is trained using several machine-learning algorithms, including artificial neural networks (ANN), support vector regression (SVR), and decision tree regression (DTR). The trained model is then used to predict the RUL of the engine.

The paper includes experimental results that demonstrate the effectiveness of the proposed method in predicting the RUL of turbofan engines. The results show that the machine learning algorithms used in the proposed method outperform traditional methods for RUL prediction, such as linear regression and exponential smoothing.

One advantage of the proposed method is that it can handle large amounts of data and learn complex relationships between the sensor data and the RUL of the engine. Additionally, the proposed method can provide early detection of engine failure, which can lead to improved maintenance and reliability of turbofan engines.

One potential disadvantage of the proposed method is that it requires a significant amount of labeled data to train the machine-learning model effectively. Additionally, the complexity of the machine learning algorithms used

in the proposed method can make it computationally expensive, which can limit the practicality of the proposed method in real-time application.

Overall, the paper provides a promising approach for predicting the RUL of turbofan engines using machine learning. The proposed method has the potential to improve the maintenance and reliability of turbofan engines, leading to reduced costs and increased safety.

[4] The paper titled "**Health state estimation and remaining useful life prediction of power devices subject to noisy and aperiodic condition monitoring**" addresses the challenge of predicting the health and remaining lifespan of power devices under noisy and irregular monitoring conditions. Power devices are critical components in various systems, and accurately predicting their health and remaining useful life (RUL) is crucial for maintenance and reliability. However, traditional predictive maintenance methods struggle in noisy and aperiodic monitoring environments. The paper proposes novel techniques to overcome these challenges. It presents advanced algorithms for health state estimation in the presence of noise. These algorithms incorporate sophisticated signal processing and filtering methods to extract meaningful information from noisy data.

Additionally, the paper introduces innovative approaches for predicting RUL under aperiodic monitoring conditions. These methods leverage machine learning and statistical models to forecast the remaining lifespan of power devices. By considering the irregularity of monitoring intervals, the proposed techniques provide more accurate predictions. The paper discusses the importance of adapting to real-world conditions where monitoring may not occur at regular intervals. It highlights the limitations of traditional models in such scenarios. The authors argue that their proposed methods offer significant improvements in accuracy and reliability. Experimental results demonstrate the effectiveness of the proposed techniques. Real-world data sets are used to validate

the performance of the algorithms. The paper provides insights into the behavior of power devices under different monitoring conditions. It discusses the implications of noisy and aperiodic monitoring on maintenance decisions. By accurately estimating health states, maintenance activities can be optimized. Predicting RUL allows for proactive maintenance scheduling, reducing downtime and costs. The proposed techniques contribute to the advancement of predictive maintenance in industrial applications. They offer practical solutions to challenges faced by engineers and maintenance professionals. The paper emphasizes the need for robust and adaptable predictive maintenance strategies. It contributes to the growing body of research in predictive maintenance and reliability engineering. The authors explore various aspects of health state estimation and RUL prediction in their analysis. They consider factors such as sensor noise, measurement errors, and operational variability. The proposed methods are designed to be scalable and applicable to different types of power devices. The paper discusses the potential impact of their research on industry practices. It suggests avenues for future research, including the integration of advanced sensor technologies. The authors highlight the broader implications of their work for the field of condition monitoring.

Overall, the paper offers valuable insights and practical solutions for predicting the health and remaining lifespan of power devices in challenging monitoring environments. It contributes to the optimization of maintenance strategies and the improvement of system reliability. By addressing the limitations of traditional methods, the proposed techniques pave the way for more effective predictive maintenance practices.

[5] The paper titled "**Transfer Learning for Remaining Useful Life Prediction of Multi-Condition Bearings Based on Bidirectional-GRU Network**" by Jiangang Lu, published in 2021. This paper proposes using a Bidirectional-Gated Recurrent Unit (GRU) network for predicting the remaining useful life of multi-

condition bearings. In this paper, "transfer learning" involves using knowledge gained from one domain or task to improve learning in another domain or task. Here, the authors likely transfer knowledge from pre-trained models or datasets to enhance the performance of the GRU network in predicting the remaining useful life of bearings under different conditions.

The Bidirectional-GRU network is a type of recurrent neural network (RNN) that can capture temporal dependencies in data from both past and future time steps, making it suitable for time series prediction tasks like remaining useful life estimation. The bidirectional aspect allows the network to learn from both past and future contexts simultaneously, potentially improving prediction accuracy. Overall, this paper proposes a novel approach to predict the remaining useful life of bearings under various conditions, leveraging transfer learning and bidirectional GRU networks for improved performance.

[6] The paper "**Remaining Useful Life Estimation in Prognostics Using Deep Convolution Neural Network**" by Li X, Ding Q Sun J-Q. It was published in 2017, focuses on predicting the remaining useful life (RUL) of machinery, which is crucial for maintenance planning and cost reduction in various industries. The authors propose a method that leverages deep convolutional neural networks (CNNs) to estimate RUL accurately. First, the authors preprocess the Time-series sensor data collected from the machinery, extracting features relevant to its health condition and degradation patterns. These features are then fed into a deep CNN architecture.

The CNN is designed to automatically learn hierarchical representations of the input data, capturing intricate patterns and correlations indicative of machinery health and remaining lifespan. The network architecture typically includes multiple convolutional layers followed by pooling layers for feature extraction and dimensionality reduction. To train the CNN, the authors use historical sensor data along with corresponding RUL labels. They employ

techniques such as transfer learning and data augmentation to improve model generalization and robustness.

Once trained, the CNN is used to predict the RUL of new instances of machinery by processing their sensor data. The predicted RUL provides valuable information for scheduling maintenance activities, preventing unexpected failures, and optimizing equipment usage. The paper evaluates the proposed method on real-world datasets from various industrial applications, demonstrating its effectiveness in accurately estimating RUL across different types of machinery and operating conditions. The results show significant improvements over traditional prognostics methods, highlighting the potential of deep learning for predictive maintenance.

Overall, the paper provides a comprehensive framework for RUL estimation using deep CNNs, offering insights into the application of advanced machine learning techniques in industrial prognostics and maintenance management.

[7] The paper titled **"Switching State Space Degradation Model with Recursive Filter/ Smoother for Prognostics of Remaining Useful Life"** by Peng Y, Wang Y, Zi Y. It was published in 2011. This paper proposes a method for prognostics, specifically for estimating the remaining useful life (RUL) of machinery. The authors introduce a switching state space degradation model, which allows for the representation of multiple degradation modes that a system may undergo. This model can capture changes in degradation behavior over time, making it more robust for real-world applications where degradation processes may be complex and dynamic.

The key component of this approach is the use of recursive filter/smothers, which are algorithms used to estimate the state of a system based on observed measurements. These algorithms iteratively update their estimates as new measurements become available, providing real-time insights into the system's health condition. By incorporating these recursive filter/smothers into the

switching state space degradation model, the authors are able to not only estimate the current state of the system but also predict its future degradation trajectory and remaining useful life. This is particularly valuable for proactive maintenance planning and decision-making. The paper provides a detailed explanation of the mathematical formulation of the switching state space degradation model and the recursive filter/smoothers, along with algorithms for implementation. Additionally, the authors demonstrate the effectiveness of their approach through experimental validation using real-world degradation data from various mechanical systems.

Overall, the paper offers a novel and practical method for prognostics by combining advanced modeling techniques with recursive filtering, contributing to the field of predictive maintenance and remaining useful life estimation.

[8] The paper titled "**Remaining useful life prognosis of turbofan engines based on deep feature extraction and fusion**" proposes a method for predicting the remaining useful life (RUL) of turbofan engines using deep feature extraction and fusion techniques.

The proposed method utilizes sensor data collected from turbofan engines and extracts deep features from the data using a convolutional neural network (CNN). The extracted features are then fused using a feature fusion layer to obtain a comprehensive representation of the engine's health status. The fused features are fed into a long short-term memory (LSTM) network to predict the RUL of the engine.

The paper includes experimental results that demonstrate the effectiveness of the proposed method in predicting the RUL of turbofan engines. The results show that the proposed method outperforms traditional methods for RUL prediction, such as linear regression and Cox proportional hazards model.

One advantage of the proposed method is that it can extract high-level features from raw sensor data, which can capture complex relationships between

the sensor data and the engine's health status. Additionally, the proposed method can handle multiple sources of data, such as vibration signals and temperature readings, to improve the accuracy of RUL prediction.

One potential disadvantage of the proposed method is that it requires a significant amount of labeled data to train the CNN and LSTM effectively. Additionally, the proposed method can be computationally expensive, which can limit the practicality of the proposed method in real-time applications.

Overall, the paper provides a promising approach for predicting the RUL of turbofan engines using deep feature extraction and fusion techniques. The proposed method has the potential to improve the maintenance and reliability of turbofan engines, leading to reduced costs and increased safety.

[9] The paper "**Remaining Useful Life Prognosis for Turbofan Engine Using Explainable Deep Neural Networks with Dimensionality Reduction**" proposes a method for predicting the remaining useful life (RUL) of turbofan engines using an explainable deep neural network (xDNN) with dimensionality reduction.

The proposed method first uses principal component analysis (PCA) for dimensionality reduction to reduce the complexity of the input sensor data. The reduced data is then fed into an xDNN model, which is a type of deep neural network that uses a decision tree-like structure to provide an explanation for the predictions made by the network.

The xDNN model is trained on a labeled dataset of sensor data and corresponding RUL values, and the trained model is used to predict the RUL of a given engine based on its sensor data. The xDNN model can also provide an explanation for its predictions by identifying the most important features that contributed to the prediction.

Experimental results reported in the paper show that the proposed method outperforms traditional methods for RUL prediction, such as linear regression

and random forest. The paper also shows that the xDNN model provides a useful explanation for its predictions, which can help engineers understand the factors that contribute to engine degradation.

One advantage of the proposed method is its ability to handle high-dimensional sensor data while reducing the complexity of the input using PCA. Additionally, the xDNN model provides an explanation for its predictions, which can help engineers understand the underlying factors that contribute to engine degradation and make informed decisions about maintenance and repair.

One potential disadvantage of the proposed method is that it requires a significant amount of labeled data to train the xDNN effectively. Additionally, the use of PCA for dimensionality reduction may result in some loss of information, which can impact the accuracy of RUL prediction.

Overall, the paper provides a promising approach for predicting the RUL of turbofan engines using an explainable deep neural network with dimensionality reduction. The proposed method has the potential to improve the maintenance and reliability of turbofan engines, leading to reduced costs and increased safety.

[10] The paper titled “**Remaining useful life (RUL) prediction for bearings based on gated recurrent units (GRUs)**” was published in 2023. This paper presents a novel approach for predicting the remaining useful life (RUL) of bearings using gated recurrent units (GRUs). The RUL prediction of bearings is crucial for effective maintenance scheduling and cost reduction in various industries. Traditional methods often struggle to capture the complex temporal patterns present in sensor data collected from bearings. In this study, we propose the use of GRUs, a type of recurrent neural network (RNN), to overcome these challenges. GRUs are specifically designed to capture long-term dependencies in sequential data, making them well-suited for RUL prediction tasks.

This paper provides an overview of the importance of predicting RUL for bearings and its implications for maintenance scheduling and cost reduction. It

highlights the limitations of traditional methods and introduces GRUs as a promising solution. This discusses previous approaches to RUL prediction, including statistical methods, machine learning algorithms, and RNNs. It outlines the advantages of RNNs for handling sequential data and introduces GRUs as an improvement over standard RNNs due to their ability to address the vanishing gradient problem and capture long-term dependencies.

This paper details the methodology used for RUL prediction with GRUs. It describes the preprocessing of sensor data, feature extraction, and the architecture of the GRU model. Hyperparameters and training procedures are also discussed. The experimental results present the performance of the proposed method on a dataset of bearing sensor data. The accuracy of RUL predictions is compared with baseline methods, demonstrating the effectiveness of GRUs. The paper analyzes the results, highlighting the strengths and limitations of the proposed approach. It also discusses potential improvements and future research directions.

In the conclusion, the paper summarizes the key findings and emphasizes the significance of GRUs for RUL prediction in bearings. It concludes by suggesting avenues for further research and development in this area.

2.1 Conclusion of Literature Survey

They may not capture the underlying physics and mechanics that cause engine degradation. They require a huge amount of high-quality training data to perform accurately.

They may not generalize well to new or unknown operating circumstances or rare and complex failure modes.

The Existing traditional methods for predicting the remaining useful life (RUL) of Commercial Jet engines have a few limitations: They struggle to handle complex relationships in the data, which can lead to less accurate predictions. They often need engineers to manually choose which features are important, which can be time-consuming and may miss important details. may have trouble

dealing with uncertainty in the data, which can make their predictions less reliable. They might not easily adapt to changes in operating conditions or new types of engines, requiring updates or modifications. While they're easy to understand, sometimes this simplicity comes at the cost of accuracy, especially in complex situations.

CHAPTER 3

SYSTEM ANALYSIS

3.1 Existing Approach

Before the rise of deep learning techniques, traditional methods for predicting the remaining useful life (RUL) of Jet Engines relied heavily on Statistical and Machine learning models. Some common approaches included:

- **Regression Models**
- **Prognostic Models**
- **Survival Analysis**
- **Expert Systems**

1.Regression Models: Linear regression and its variants were commonly used to model the relationship between various features extracted from engine sensor data and the remaining useful life. These models typically assumed a linear relationship between the input features and the RUL.

2.Prognostic Models: Prognostic models were developed based on physics-based understanding of engine degradation mechanisms. These models often involved complex mathematical formulations and required detailed knowledge of the underlying physics of engine operation. They aimed to simulate the degradation process and predict the RUL based on the observed degradation patterns.

3.Survival Analysis: Survival analysis techniques, such as Kaplan-Meier estimation and Cox proportional hazards models, were applied to analyze time-to-failure data obtained from engine sensor measurements. These methods accounted for censoring and provided estimates of the probability of failure at different time points.

4.Expert Systems: Expert systems incorporated domain knowledge and heuristics provided by domain experts to predict the RUL of jet engines. These systems typically involved rule-based reasoning and decision-making based on predefined rules and thresholds.

The existing approaches for predicting the remaining useful life (RUL) of jet engines is their ability to provide understandable and reliable predictions. These methods use knowledge about how jet engines work and incorporate expert insights, making them trustworthy tools for decision-making in industries like aviation.

3.2 Limitations of existing system

The Existing traditional methods for predicting the remaining useful life (RUL) of Commercial Jet engines have a few limitations:

- **Difficulty with Complexity:** They struggle to handle complex relationships in the data, which can lead to less accurate predictions.
- **Manual Work Required:** They often need engineers to manually choose which features are important, which can be time-consuming and may miss important details.
- **Handling Uncertainty:** They may have trouble dealing with uncertainty in the data, which can make their predictions less reliable.
- **Adaptability Issues:** They might not easily adapt to changes in operating conditions or new types of engines, requiring updates or modifications.
- **Trade-off Between Interpretability and Accuracy:** While they're easy to understand, sometimes this simplicity comes at the cost of accuracy, especially in complex situations.

These limitations have prompted the exploration of deep learning techniques, which can handle complex data better and may offer more accurate predictions for RUL.

3.3 Proposed System

The proposed methodology will accurately calculate the Remaining Useful Life and anticipate the engine's performance effectively. The main focus will be the prediction of the RUL of the Commercial Jet engine considering the failure and more precisely capturing low RUL values to avoid putting the machine at risk.

3.4 Steps in the proposed approach

Data Collection

Data preprocessing

Feature engineering

Model Building

3.5 Advantages of The Proposed System

- Detection is perfect
- Avoid difficulties in preprocessing
- Best performance
- Faster than the existing approaches

3.6 Data Flow Diagram

Level 0

The Fig 3.1 we've described illustrates the process of using deep learning algorithms to generate results. Here's a breakdown:

User Input:

This is the initial data or information provided by the user. It could be anything from text, images, or numerical data.

Deep Learning Algorithms:

These are complex mathematical models designed to learn patterns and features from data. Deep learning algorithms typically involve neural networks with many layers, hence the term "deep." These algorithms analyze the input data and extract relevant features.

Results:

This is the output generated by the deep learning algorithms. The results could vary depending on the task. For example, if the input is an image, the result might be a classification label indicating what the image depicts. If the input is text, the result could be sentiment analysis, translation, or text generation.

So, the block diagram shows how data flows from the user input through deep learning algorithms to produce meaningful results.



Fig 3.1 DFD level 0

Level 1

From the Fig 3.2 outlines a process for predicting Remaining Useful Life (RUL) using machine learning techniques. Here's a breakdown:

Data Collection (Input):

The process starts with collecting data relevant to the system being analyzed. This data could include sensor readings, operational parameters, or historical records. It serves as the foundation for the RUL prediction.

Data Preprocessing:

Before the data can be used for training the model, it needs to be preprocessed. This involves cleaning the data, handling missing values, and transforming it into a suitable format for analysis.

Feature Engineering:

This step involves selecting, extracting, or creating features from the raw data that are relevant and informative for predicting the Remaining Useful Life. Feature engineering helps in improving the model's performance by providing it with meaningful inputs.

Model Training:

Once the data is preprocessed and features are engineered, the next step is to train a machine learning model. This model learns from the input data to make predictions about the Remaining Useful Life of the system.

RUL Prediction:

After the model is trained, it can be used to predict the Remaining Useful Life of the system based on new input data. The model takes into account the features extracted during preprocessing and the patterns learned during training to make these predictions.

Result:

The final output of the process is the prediction of the Remaining Useful Life. This prediction can be used for various purposes such as maintenance scheduling, resource allocation, or decision-making related to the system being analyzed.

Fig 3.2, the block diagram illustrates a structured approach for predicting the Remaining Useful Life of a system, starting from data collection through to model training and prediction, with an emphasis on data preprocessing and feature engineering to enhance prediction accuracy.

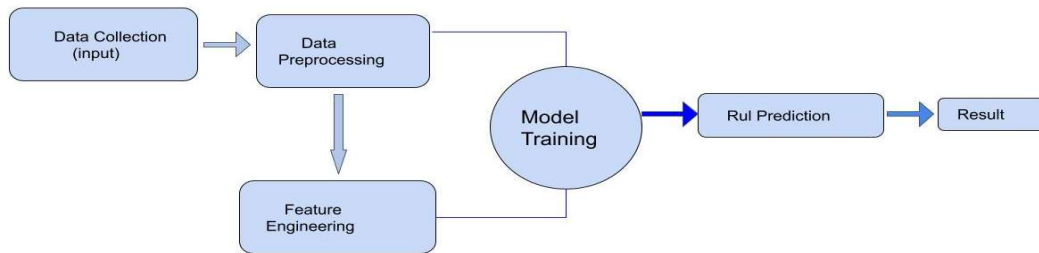


Fig 3.2 DFD level 1

3.7 Module Diagram

The steps followed in the proposed approach are

- Data Set Extraction of Commercial Aircraft
- Data Processing
- Feature Selection
- Training The Model (LSTM)
- Testing And Rul Prediction
- Performance Evaluation

3.7.1 Dataset Collection

Engine degradation simulation was carried out using C-MAPSS. Four different were sets simulated under different combinations of operational conditions and fault modes. Records several sensor channels to characterize fault evolution. The data set was provided by the Prognostics Coe at NASA Ames.

3.7.2 Dataset Description

The dataset FD001 contains a time series of 21 sensors and 3 settings of 100 units (Commercial Jet engine).

Each engine works normally at the beginning of each time series and fails at the end of the time series. Each row is a snapshot of the data taken during a single operation cycle.

The training dataset consists of 20631 rows and 26 columns and the testing dataset consists of 13096 rows and 26 columns which is an average of 30% of total values.

This Module diagram Fig.3.3 outlines the process of developing a predictive maintenance model for commercial aircraft using LSTM (Long Short-Term Memory) neural networks. Here's a breakdown:

Data set extraction of Commercial Aircraft:

This step involves gathering relevant data about commercial aircraft, which could include information about various components, operational parameters, maintenance records, etc.

Data processing:

Once the data is collected, it needs to be preprocessed to clean it, handle missing values, normalize or standardize features, and prepare it for modeling. The dataset FD001 contains a time series of 21 sensors and 3 settings of 100 units (Commercial Jet engine).

Each engine works normally at the beginning of each time series and fails at the end of the time series. Each row is a snapshot of the data taken during a single operation cycle.

The training dataset consists of 20631 rows and 26 columns and the testing dataset consists of 13096 rows and 26 columns which is an average of 30% of total values.

Feature Selection:

Not all features may be relevant or useful for predicting the Remaining Useful Life (RUL) of aircraft components. Feature selection involves identifying and selecting the most important features that contribute to the predictive accuracy of the model.

Training the model (LSTM):

LSTM, a type of recurrent neural network (RNN), is used for modeling the data. It's particularly effective for sequences and time-series data, making it suitable for predicting the RUL of aircraft components based on historical data.

Testing and RUL prediction:

Once the LSTM model is trained, it's tested using a separate set of data to evaluate its performance. Then, the model is used to predict the Remaining Useful Life (RUL) of aircraft components based on their current condition and historical data.

Performance Evaluation:

Finally, the performance of the LSTM model is evaluated using various metrics such as accuracy, precision, recall, F1-score, etc. This step helps assess how well the model performs in predicting the RUL of aircraft components and identifies areas for improvement if necessary.



Fig 3.3 Module Diagram

Sensor number	Sensor Description	Units
1	Fan inlet temperature	°R
2	LPC outlet temperature	°R
3	HPC outlet temperature	°R
4	LPT outlet temperature	°R
5	Fan inlet pressure	psia
6	Bypass-duct pressure	psia
7	HPC outlet pressure	psia
8	Physical fan speed	rpm
9	Physical core speed	rpm
10	Engine pressure ratio P50/P2	-
11	HPC outlet static pressure	psia
12	Ratio of fuel flow to Ps30	pps/psia
13	Corrected fan speed	rpm
14	Corrected core speed	rpm
15	Bypass ratio	-
16	Burner fuel–air ratio	-
17	Bleed enthalpy	-
18	Required fan speed	rpm
19	Required fan conversion speed	rpm
20	High-pressure turbines cool air flow	lb/s
21	Low-pressure turbines cool air flow	lb/s

Table 3.1 Data description of Commercial Jet engine sensors

The **Table 3.1** likely lists 21 different sensors used in the project aimed at predicting the Remaining useful life of Commercial Jets. These sensors could monitor various aspects of the aircraft's condition such as engine performance, structural integrity, fuel efficiency etc. data from these sensors is likely

analyzed to predict how much longer each aircraft can remain in service before needing significant maintenance or replacement.

3.7.3 Data Preprocessing

Data Exploration

Data visualization

Feature Engineering

3.7.4 Data Exploration

- When inspecting the max time cycles, we can see
- The engine which failed the earliest did so after 128 cycles,
- Whereas the engine which operated the longest broke down after 362 cycles.
- The average engine breaks between 199 and 206 cycles.
- RUL corresponds to the remaining time cycles for each unit before it fails.

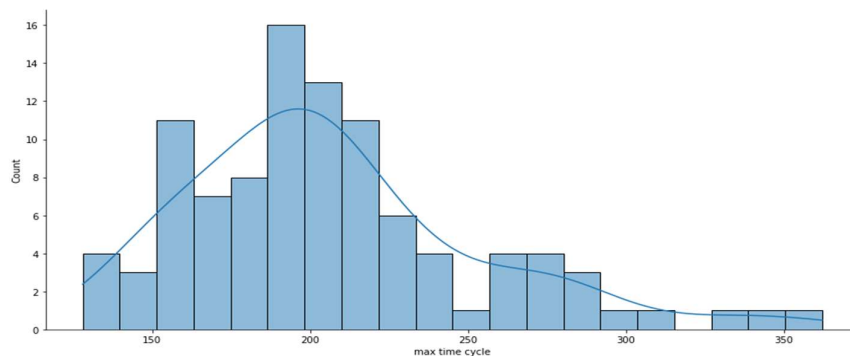


fig 3.4 Time Cycle vs count

From the Fig 3.4, we notice that most of the time, the maximum time cycles that an engine can achieve is between 190 and 210 before HPC failure

3.7.5 Data Visualization

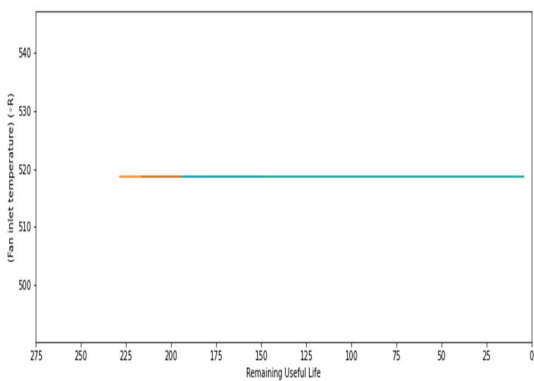


Fig3.5 Rul vs Sensor 1

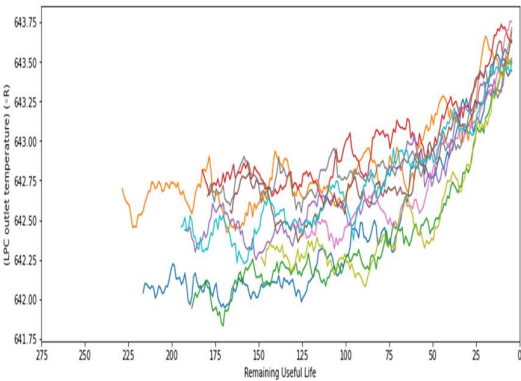


Fig3.6 Rul vs Sensor 2

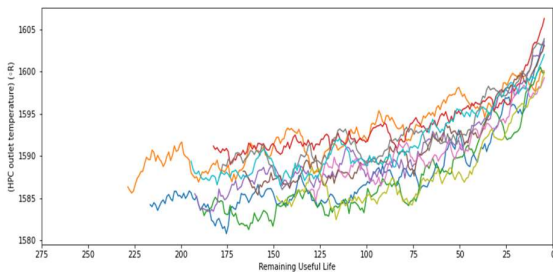


Fig3.7 Rul vs Sensor 3

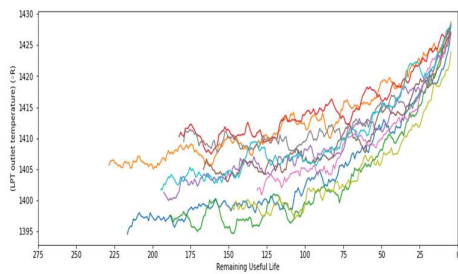


Fig3.8 Rul vs Sensor 4

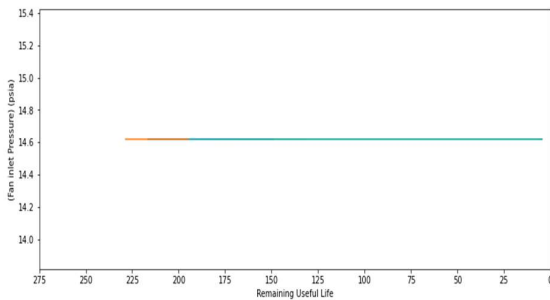


Fig3.9 Rul vs Sensor 5

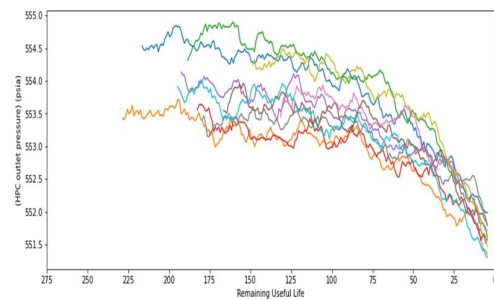


Fig3.10 Rul vs Sensor 7

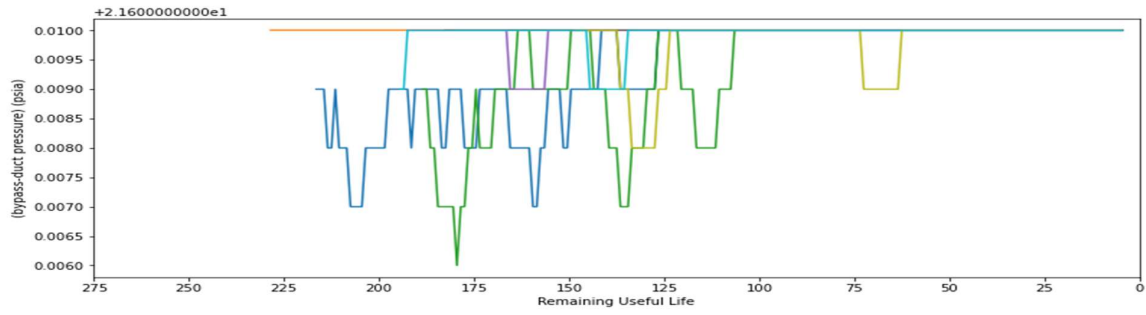


Fig3.11 Rul vs Sensor 6

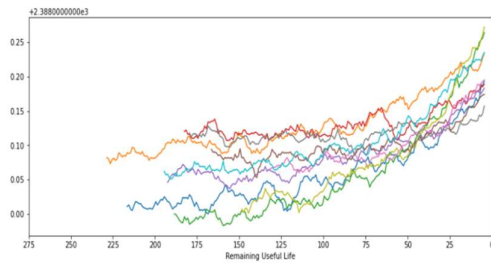


Fig3.12 Rul vs Sensor 8

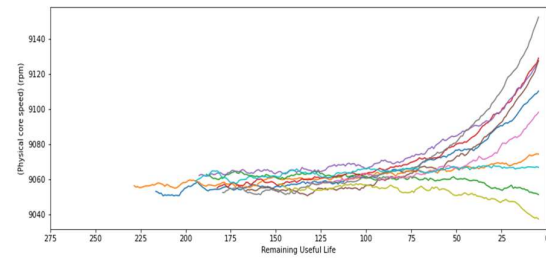


Fig3.13 Rul vs Sensor 9

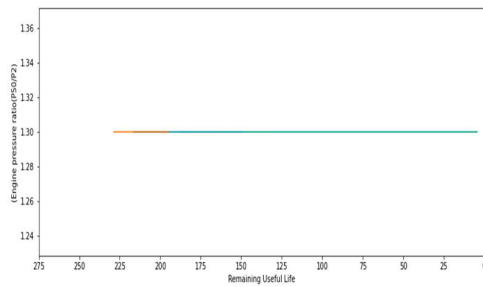


Fig3.14 Rul vs Sensor 10

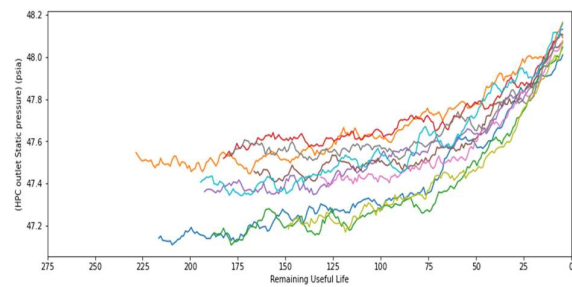


Fig3.15 Rul vs Sensor 11

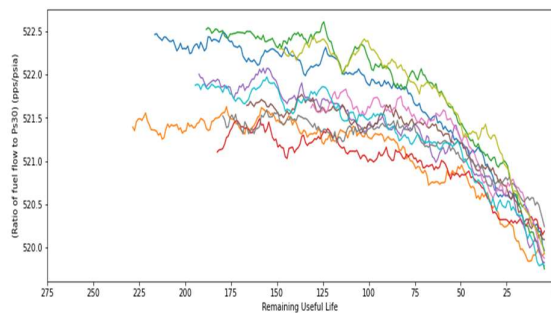


Fig3.16 Rul vs Sensor 12

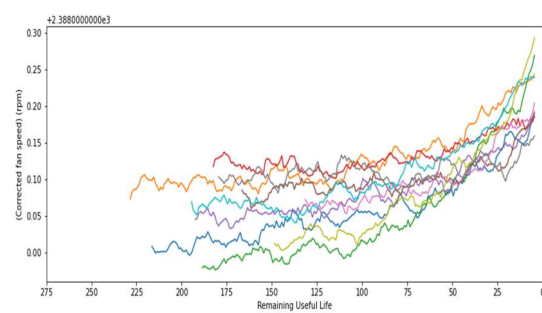


Fig3.17 Rul vs Sensor 13

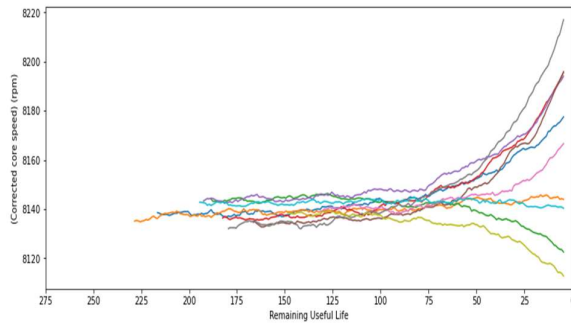


Fig3.18 Rul vs Sensor 14

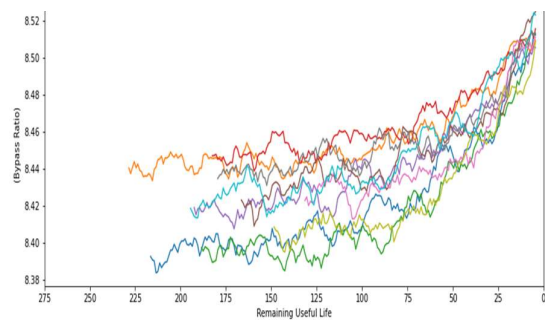


Fig3.19 Rul vs Sensor 15

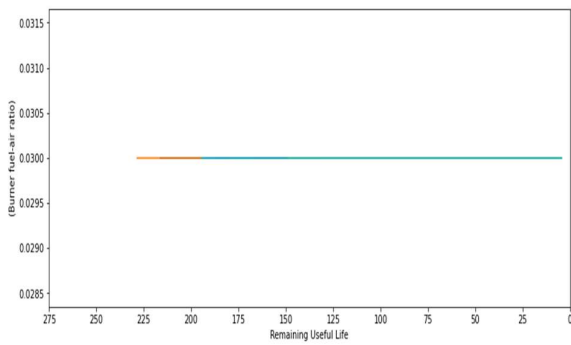


Fig3.20 Rul vs Sensor 16

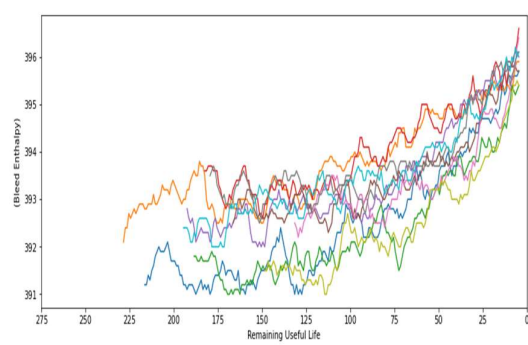


Fig3.21 Rul vs Sensor17

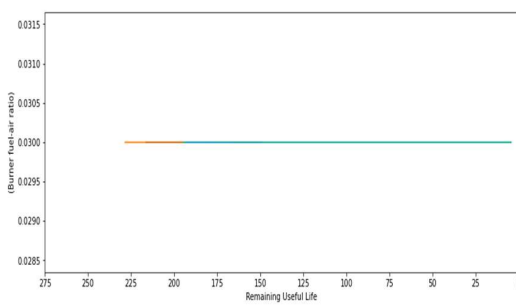


Fig3.22 Rul vs Sensor18

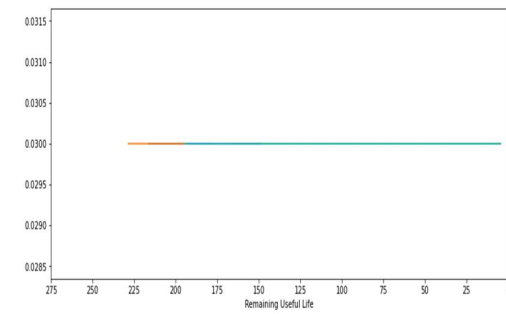


Fig3.23 Rul vs Sensor19

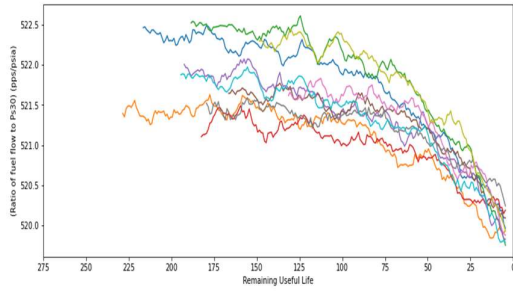


Fig3.24 Rul vs Sensor20

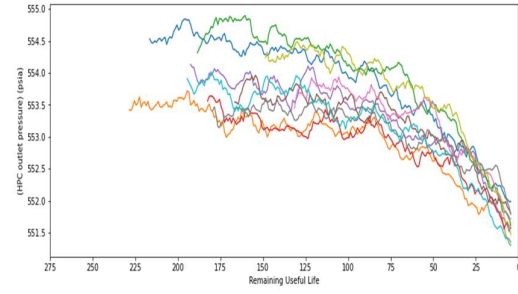


Fig3.25 Rul vs Sensor21

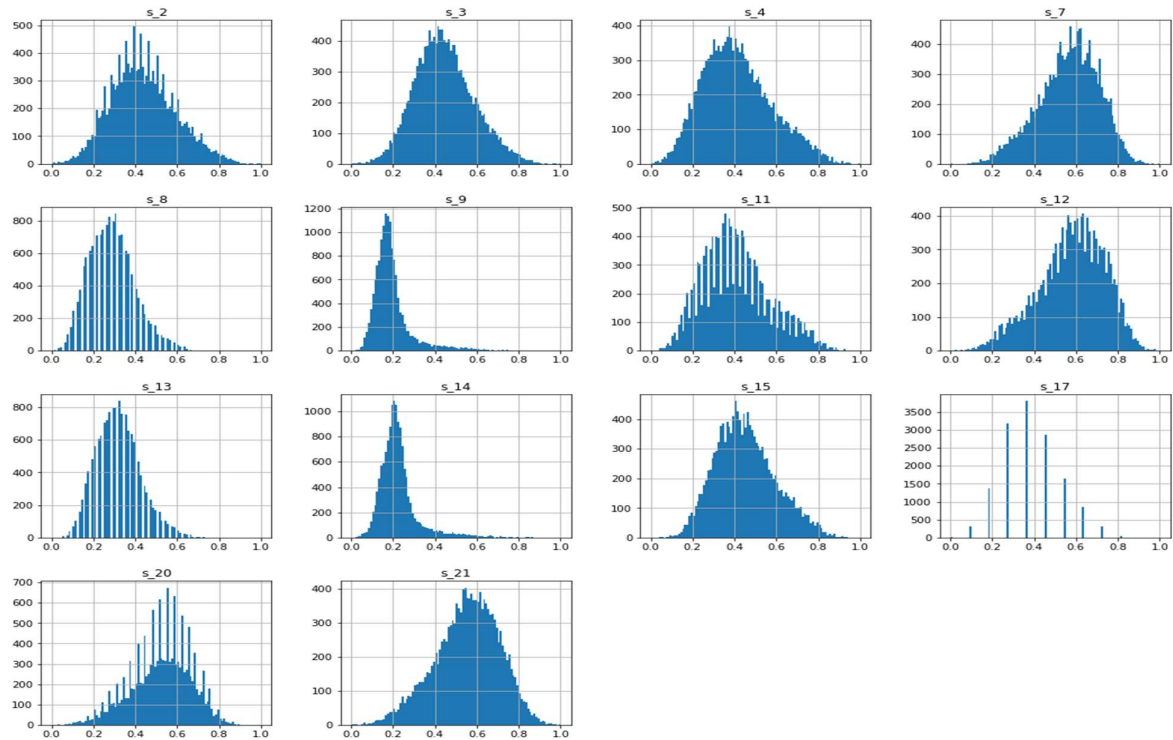


Fig3.26 Selected sensors

The plots for the remaining selected features are for sensors 2,3,4,7,8,9,11,12,13,14,15,17,20,21

Trend	Sensor number
Increasing	[2, 3, 4, 8, 11, 13, 14, 15, 17]
Decreasing	[7, 12, 20, 21]
Irregular	[9, 14]
Unchanged	[1, 5, 6, 10, 16, 18, 19]

Table 3.2 Sensor trends summary

After Visualizing the sensor measurements, we can conclude in the **Table 3.2**

- Sensors 1,10,18,19 are similar hence they hold no information
- The Sensors 5,16 show flat lines, and excluded
- Sensors 2,3,4,8,11,13,15,17 shows rising trends
- The sensor 6 shows the peak downwards
- The sensor 7,12,20,21 shows decreasing trend
- The sensor 9 and 14 shows the similar pattern
- Based on our Exploratory Data Analysis we can determine sensors 1, 5, 6, 10, 16, 18 and 19 hold no information related to RUL as the sensor values remain constant throughout time.so, the sensors will be taken for prediction.

3.8 Feature Engineering

Lag Feature

Lag feature is created by shifting the original time series forward or backward in time by a certain number of time steps. For example, a lag feature of one day for a time series of daily sales data would represent the sales from the previous day. Lag features are commonly used in time series forecasting and

prediction tasks, as they can capture the temporal dependencies between past and future values of the time series.

By including lag features in a Deep learning model, it can better capture patterns and trends in the time series data. An engine's Remaining useful life can be estimated based on changes in sensor measurements. For instance, if a certain sensor measurement steadily rises or falls over time, this can be a sign of a degradation tendency that could eventually cause failure.

```
x_train_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20631 entries, 0 to 20630
Data columns (total 42 columns):
#   Column              Non-Null Count  Dtype
---  -
0   s_2                  20631 non-null  float64
1   s_3                  20631 non-null  float64
2   s_4                  20631 non-null  float64
3   s_7                  20631 non-null  float64
4   s_8                  20631 non-null  float64
5   s_9                  20631 non-null  float64
6   s_11                 20631 non-null  float64
7   s_12                 20631 non-null  float64
8   s_13                 20631 non-null  float64
9   s_14                 20631 non-null  float64
10  s_15                 20631 non-null  float64
11  s_17                 20631 non-null  int64
12  s_20                 20631 non-null  float64
13  s_21                 20631 non-null  float64
14  s_2_lag1             20630 non-null  float64
15  s_2_lag2             20629 non-null  float64
16  s_3_lag1             20630 non-null  float64
17  s_3_lag2             20629 non-null  float64
18  s_4_lag1             20630 non-null  float64
19  s_4_lag2             20629 non-null  float64
20  s_7_lag1             20630 non-null  float64
21  s_7_lag2             20629 non-null  float64
22  s_8_lag1             20630 non-null  float64
--  -
```

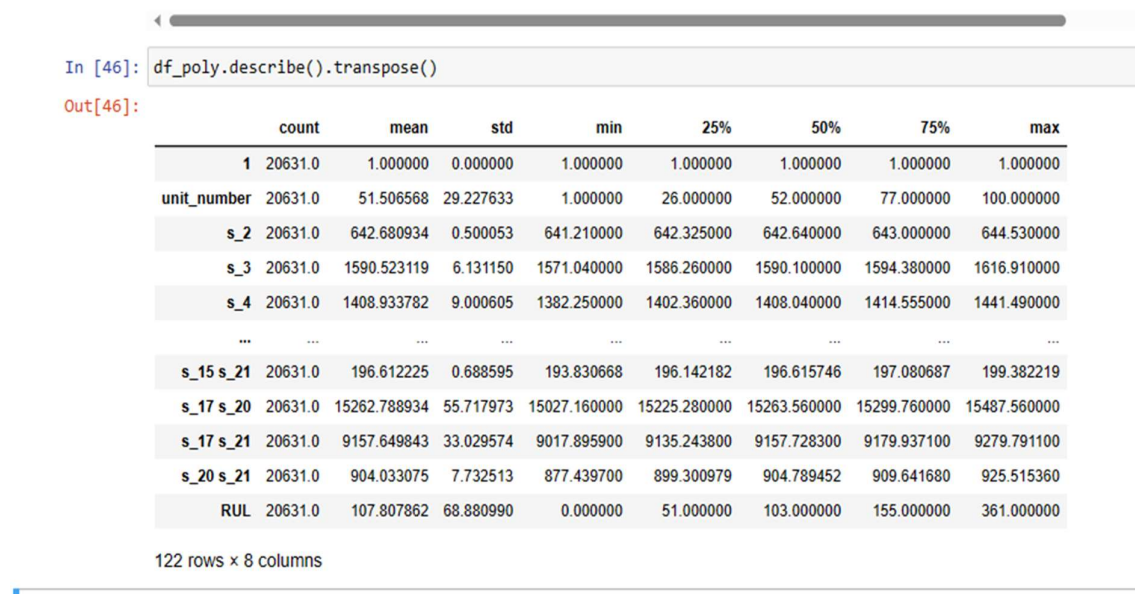
Fig 3.27 lag feature

The column called s_1_lag1 that represents the sensor measurements for s1 shifted by one cycle (i.e., the previous cycle's measurement). We have used the groupby() method to group the data by unit (i.e., engine number) and applied the shift() method to shift the s1 data by one row. Finally, we have removed the rows with missing values using the dropna() method.

Polynomial Feature

Polynomial feature transformation can be applied to a wide range of machine learning algorithms, including linear regression, logistic regression, and support vector machines. It can help improve the model's accuracy by capturing complex relationships between the input features and the output variable.

Polynomial feature transformation can also be applied in RUL (Remaining Useful Life) prediction for Commercial Jet engines to capture the non-linear relationships between the sensor measurements and the RUL. In this context, polynomial features represent the higher-order interactions between the sensor measurements, which can be used to predict the RUL more accurately.



```
In [46]: df_poly.describe().transpose()
```

Out[46]:

	count	mean	std	min	25%	50%	75%	max
1	20631.0	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000
unit_number	20631.0	51.506568	29.227633	1.000000	26.000000	52.000000	77.000000	100.000000
s_2	20631.0	642.680934	0.500053	641.210000	642.325000	642.640000	643.000000	644.530000
s_3	20631.0	1590.523119	6.131150	1571.040000	1586.260000	1590.100000	1594.380000	1616.910000
s_4	20631.0	1408.933782	9.000605	1382.250000	1402.360000	1408.040000	1414.555000	1441.490000
...
s_15 s_21	20631.0	196.612225	0.688595	193.830668	196.142182	196.615746	197.080687	199.382219
s_17 s_20	20631.0	15262.788934	55.717973	15027.160000	15225.280000	15263.560000	15299.760000	15487.560000
s_17 s_21	20631.0	9157.649843	33.029574	9017.895900	9135.243800	9157.728300	9179.937100	9279.791100
s_20 s_21	20631.0	904.033075	7.732513	877.439700	899.300979	904.789452	909.641680	925.515360
RUL	20631.0	107.807862	68.880990	0.000000	51.000000	103.000000	155.000000	361.000000

122 rows x 8 columns

Fig 3.28 polynomial feature

3.9 Model Building

3.9.1 Existing approach

In the existing system the algorithms like Random Forest Regression, Support Vector Regression, Linear Regression etc. There are used all algorithms All results give high error rate.

Random Forest Regression

Random Forest Regression is a supervised learning algorithm that uses ensemble learning methods for regression. Ensemble learning method is a technique that combines predictions from multiple machine learning algorithms to make a more accurate prediction than a single model.

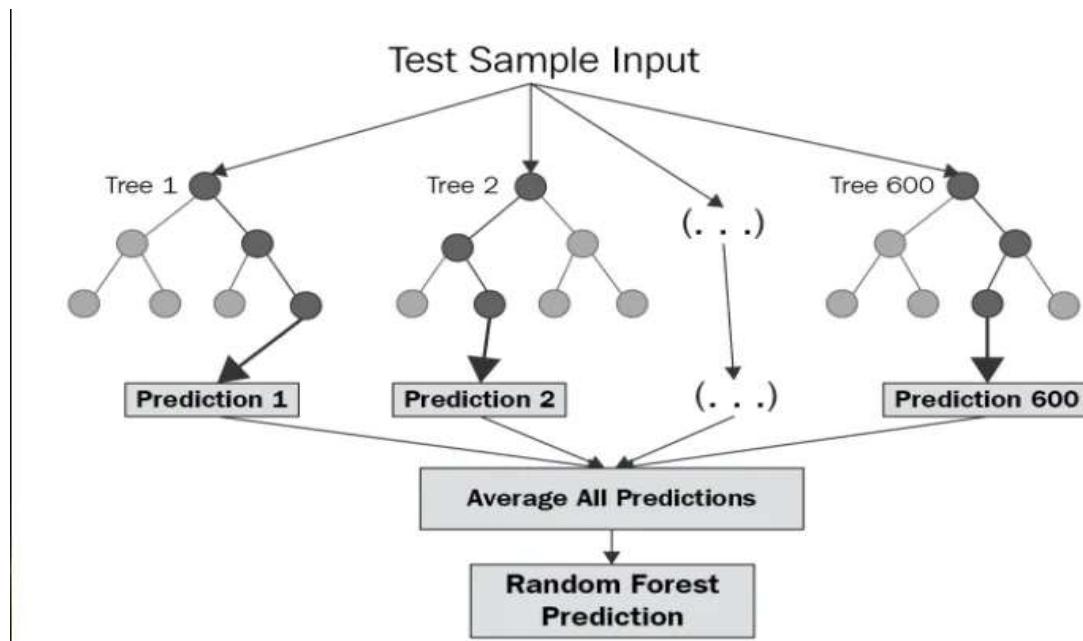


Fig 3.29 Architecture of random forest

1. This Random Forest is a simple supervised algorithm.
2. Each node in the tree represents a decision based on the values of one of the sensor inputs. The decision tree uses these sensor values to predict the RUL of the turbofan engine at each time point.

Support Vector Regressor

Support Vector Regression (SVR) is a type of machine learning algorithm used for regression analysis. The goal of SVR is to find a function that

approximates the relationship between the input variables and a continuous target variable, while minimizing the prediction error.

SVR can handle non-linear relationships between the input variables and the target variable by using a kernel function to map the data to a higher-dimensional space. This makes it a powerful tool for regression tasks where there may be complex relationships between the input variables and the target variable.

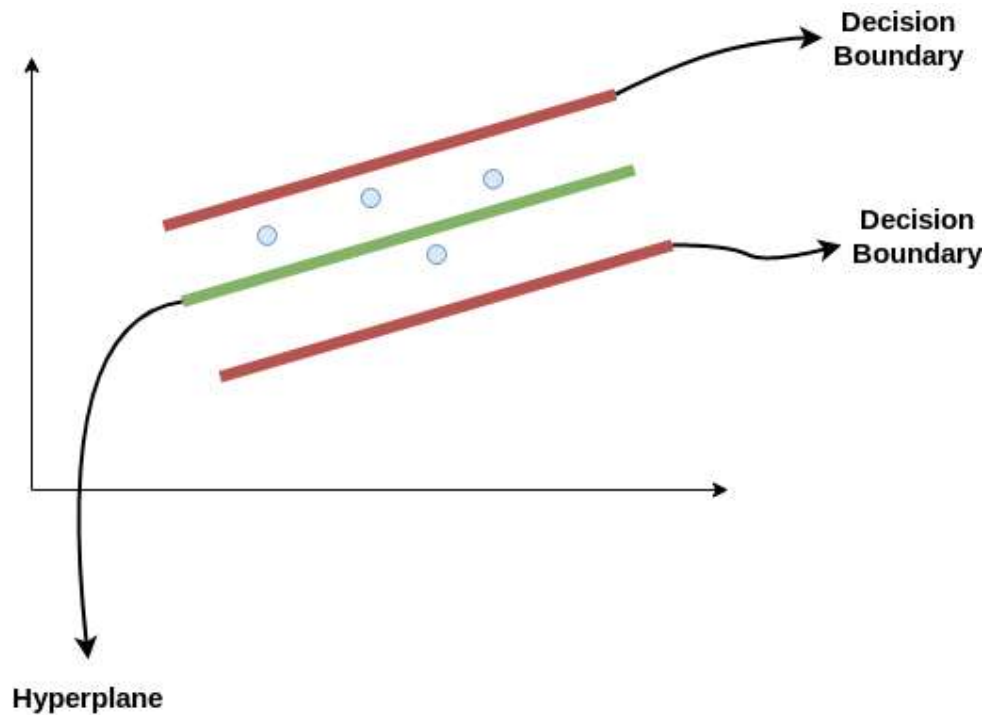


Fig 3.30 Architecture of support vector regression

1. Support Vector Regression is a supervised learning algorithm that is used to predict discrete values
2. Support Vector Regression uses the same principle as the SVMs. The basic idea behind SVR is to find the best-fit line. In SVR, the best-fit line is the hyperplane that has the maximum number of points.

Linear Regression

Linear regression is a type of statistical analysis used to predict the relationship between two variables. It assumes a linear relationship between the independent variable and the dependent variable, and aims to find the best-fitting line that describes the relationship. The line is determined by minimizing the sum of the squared differences between the predicted values and the actual values.

Linear regression is a quiet and the simplest statistical regression method used for predictive analysis in machine learning. Linear regression shows the linear relationship between the independent(predictor) variable i.e. X-axis and the dependent(output) variable i.e. Y-axis, called linear regression. If there is a single input variable X (independent variable), such linear regression is called simple linear regression.

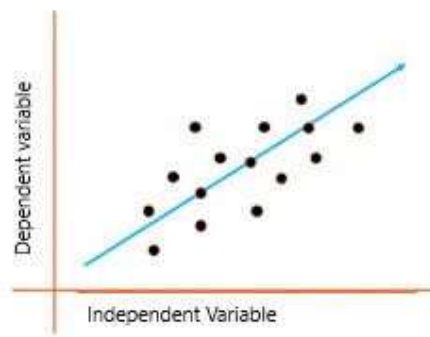


Fig 3.31 Architecture of Linear regression

1. Linear regression analysis is used to predict the value of a variable based on the value of another variable. The variable you want to predict is called the dependent variable.
2. The variable you are using to predict the other variable's value is called the independent variable

3.9.1.1 Performance metrics

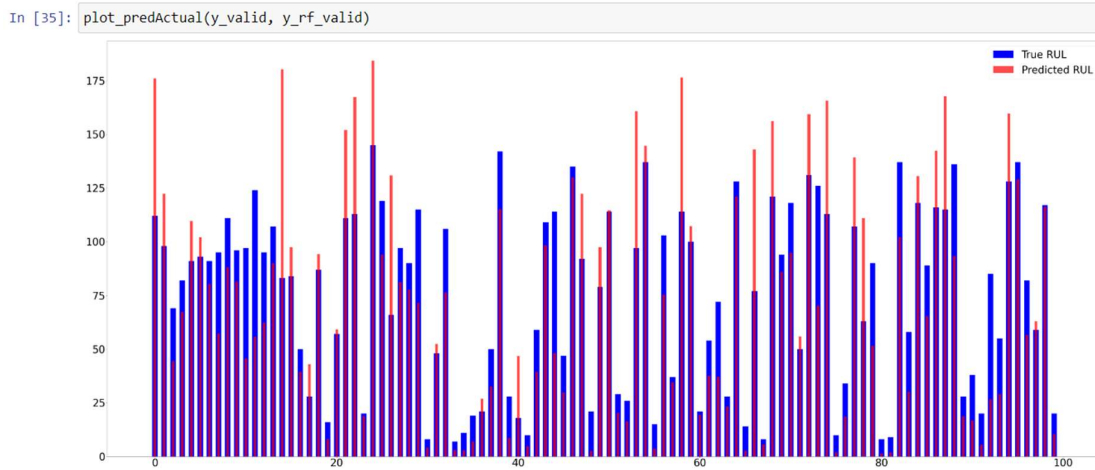


Fig 3.32 All Features Without Historical Data

Linear Regression:

train set RMSE:56.19869318681088, R2:0.3439361345919162

test set RMSE:54.264009074691856, R2:0.35713062175493715

valid set RMSE:34.84962228223509, R2:0.2967064952591344

Support Vector Regressor:

train set RMSE:43.56448464126648, R2:0.605761670162996

test set RMSE:47.26718449996112, R2:0.5122262167433573

valid set RMSE:34.84962228223509, R2:0.2967064952591344

Random forest Regression:

train set RMSE:15.408303170220156, R2:0.9506822432050425

test set RMSE:44.36383009860364, R2:0.5703082128383993

valid set RMSE:30.653618024631285, R2:0.45586841183607596

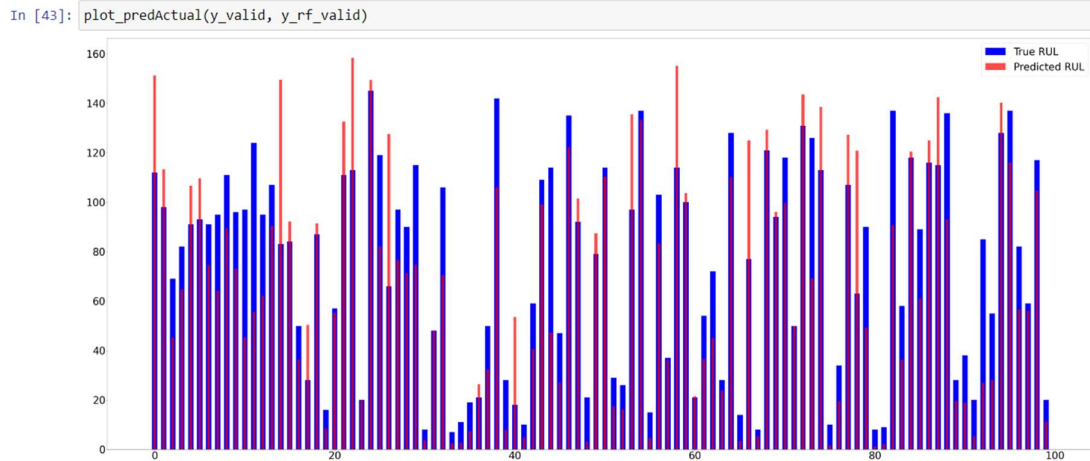


Fig 3.33 Data Without Useless Sensors, Without Historical Data

Linear Regression:

train set RMSE:47.86312955545103, R2:0.3792662465007206

test set RMSE:55.6261002397046, R2:0.3244519972933424

valid set RMSE:34.21070108960351, R2:0.3222579927041398

Support Vector Regressor:

train set RMSE:32.91543148706822, R2:0.7064363470171213

test set RMSE:48.75516379159361, R2:0.48103242301470384

valid set RMSE:25.947912225366203, R2:0.6101071274546097

Random forest Regression:

train set RMSE:12.036862499525814, R2:0.9607418547981479

test set RMSE:45.81838291793876, R2:0.5416697719315231

valid set RMSE:26.548403567823055, R2:0.5918523714819

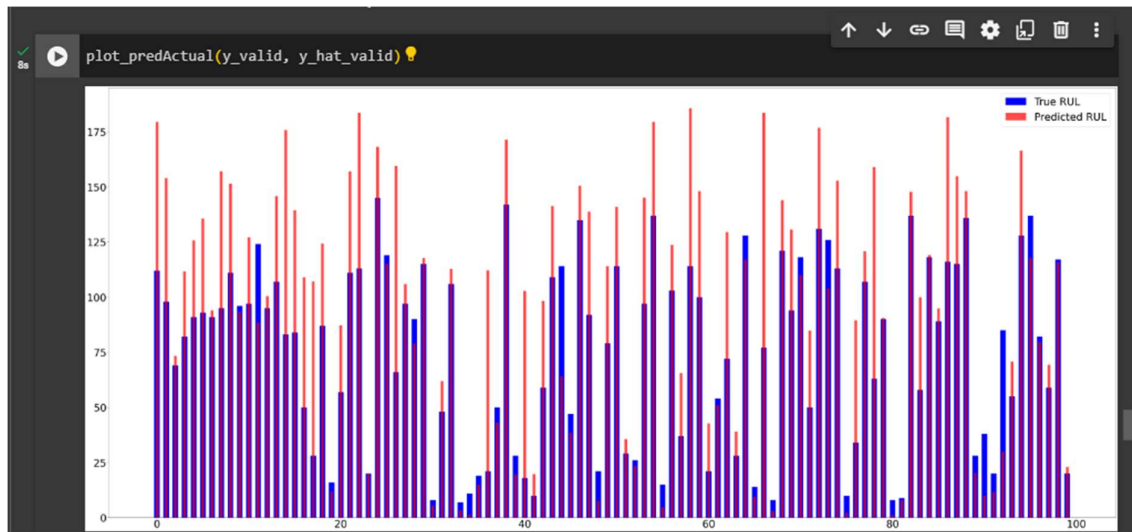


Fig 3.34 Data Without Useless Sensors, With Historical Data

Linear Regression:

train set RMSE:36.12278318538548, R2:0.6461026809508146

test set RMSE:36.24759043148782, R2:0.6359939567063242

valid set RMSE:33.12521026826295, R2:0.3645845897590453

Support Vector Regressor:

train set RMSE:32.313524381002445, R2:0.7168063437968639

test set RMSE:32.7061504393512, R2:0.7036471078045101

valid set RMSE:34.076921735663475, R2:0.3275481860480176

Random forest Regression:

train set RMSE:9.785392549750972, R2:0.974030034737145

test set RMSE:26.696381891435006, R2:0.802550916397019

valid set RMSE:34.099712755388424, R2:0.3266483989294847

3.9.2 Proposed Approach

DEEP NEURAL NETWORK(DNN)

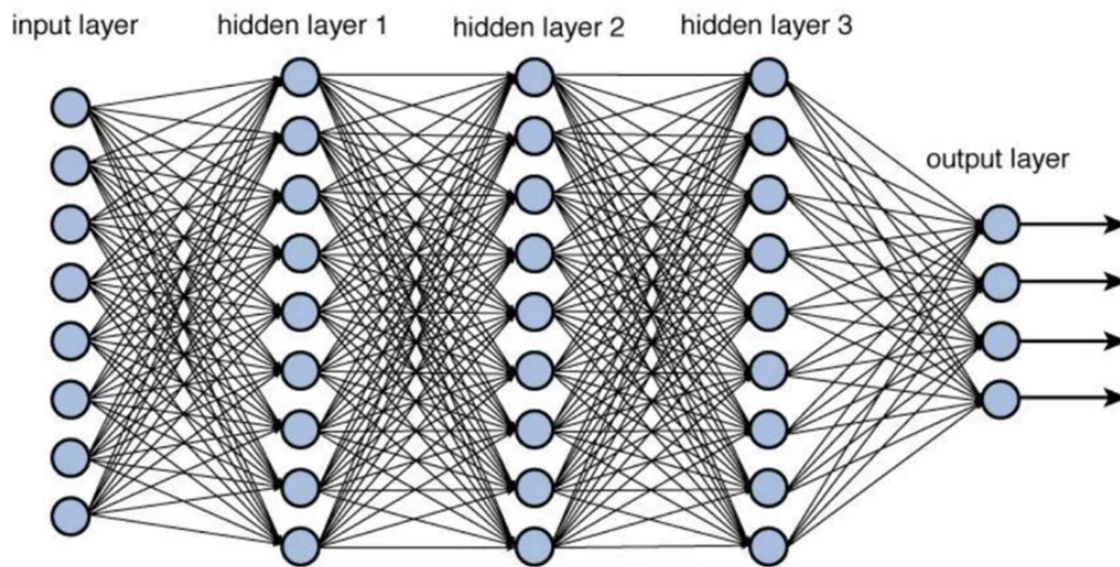


Fig 3.35 Deep Neural Network

A deep neural network (DNN) is a type of artificial neural network (ANN) that has multiple layers between the input and output layers. These layers help the network learn complex patterns in data by progressively extracting higher-level features. Each layer consists of interconnected nodes (neurons) that apply transformations to the input data. Deep neural networks are used in various tasks like image recognition, natural language processing, and voice recognition.

Long Short-Term Memory (LSTM)

LSTM is a special type of RNN algorithm that evolved to overcome the demerits of the RNN algorithm i.e., short-term memory of sequence. RNN isn't able to predict the solution for big scenarios and context. LSTM can remember the essential context for the long term and can forget unnecessary things using forget gates and add a gate, used more familiarly than RNN due to its memory capability. Forget gate is designed using the sigmoid function which is used to remove the unwanted memory in the LSTM cell. Add gate is designed by using tanh and sigmoid function. In NLP, a sequence of words or sentences passed over

a period of time, the output of each cell may take or regret and pass to the next cell depending upon the problems and scenarios.

- LSTM stands for long short-term memory networks, used in the field of Deep Learning. It is a variety of recurrent neural networks (RNNs) that are capable of learning long-term dependencies, especially in sequence prediction problems.
- LSTM in RUL prediction, we can capture the temporal dependencies between the sensor measurements and RUL, and also handle the non-linearity and complex relationships between the features. LSTM can provide accurate and reliable RUL predictions, which can help in proactive maintenance scheduling and reduce the risk of equipment failure.

```
In [140]: history = model.fit(X_train, y_train, epochs = 30 ,
                             validation_data = (X_test, y_test),
                             shuffle = False,
                             batch_size = 96, verbose = 2)
```

```
Epoch 1/30
148/148 - 11s - loss: 801854.7500 - mse: 6981.9897 - val_loss: 89445.8281 - val_mse: 4255.9482
Epoch 2/30
148/148 - 7s - loss: 45684.4570 - mse: 3434.9690 - val_loss: 18407.4160 - val_mse: 2765.6233
Epoch 3/30
148/148 - 8s - loss: 13442.4131 - mse: 2166.0005 - val_loss: 5543.7827 - val_mse: 1583.7644
Epoch 4/30
148/148 - 7s - loss: 5699.4121 - mse: 1489.6654 - val_loss: 2851.4175 - val_mse: 1198.7019
Epoch 5/30
148/148 - 7s - loss: 3458.8037 - mse: 1168.1829 - val_loss: 1713.9327 - val_mse: 915.4308
Epoch 6/30
148/148 - 7s - loss: 2573.4087 - mse: 990.9606 - val_loss: 1677.5808 - val_mse: 888.9062
Epoch 7/30
148/148 - 8s - loss: 2255.0129 - mse: 901.5261 - val_loss: 1085.8528 - val_mse: 686.0731
Epoch 8/30
148/148 - 7s - loss: 1800.3473 - mse: 796.4572 - val_loss: 858.9575 - val_mse: 609.5947
Epoch 9/30
148/148 - 7s - loss: 1595.1573 - mse: 727.0687 - val_loss: 711.9587 - val_mse: 540.1740
Epoch 10/30
148/148 - 7s - loss: 1496.0670 - mse: 693.9235 - val_loss: 800.8646 - val_mse: 559.7991
Epoch 11/30
148/148 - 7s - loss: 1648.5452 - mse: 702.5109 - val_loss: 706.2751 - val_mse: 519.8767
Epoch 12/30
148/148 - 7s - loss: 1587.1489 - mse: 683.3690 - val_loss: 680.8571 - val_mse: 481.5082
Epoch 13/30
148/148 - 7s - loss: 1402.3809 - mse: 650.5212 - val_loss: 591.4016 - val_mse: 433.5052
Epoch 14/30
148/148 - 7s - loss: 1508.9955 - mse: 654.6163 - val_loss: 593.3486 - val_mse: 439.8290
Epoch 15/30
148/148 - 7s - loss: 1415.7455 - mse: 635.5909 - val_loss: 578.6046 - val_mse: 446.1237
Epoch 16/30
```

Fig 3.36 (a) LSTM Model

```

148/148 - 8s - loss: 1068.0944 - mse: 519.0379 - val_loss: 469.9640 - val_mse: 380.4148
Epoch 27/30
148/148 - 7s - loss: 930.6558 - mse: 491.4902 - val_loss: 399.2488 - val_mse: 315.2323
Epoch 28/30
148/148 - 7s - loss: 932.4613 - mse: 484.8512 - val_loss: 514.0067 - val_mse: 400.4417
Epoch 29/30
148/148 - 7s - loss: 1008.5410 - mse: 506.5692 - val_loss: 547.5371 - val_mse: 413.0472
Epoch 30/30
148/148 - 7s - loss: 919.5463 - mse: 475.7612 - val_loss: 381.6017 - val_mse: 300.8080

In [141]: RUL_predicted = model.predict(test_processed)

MSE = mean_squared_error(RUL_FD001, RUL_predicted)
MAE = np.abs(RUL_FD001 - RUL_predicted).values.mean()
std_AE = np.abs(RUL_FD001 - RUL_predicted).values.std()
print("MSE: ", MSE.round(2))
print("RMSE: ", np.sqrt(MSE).round(2))
print("MAE: ", MAE.round(2))
print("std_AE: ", std_AE.round(2))
print("s_score: ", s_score(RUL_FD001, RUL_predicted).round(2))

MSE: 206.32
RMSE: 14.36
MAE: 10.45
std_AE: 9.85
s_score: 377.37

In [142]: #True RUL vs predicted RUL
fig, axes = plt.subplots(1, figsize = (7,5))
fig.tight_layout()
axes.plot(RUL_FD001, label = "True RUL", color = "red")
axes.plot(RUL_predicted, label = "Predicted RUL")
axes.set_title("True RUL vs predicted RUL")
axes.legend()
plt.show()

```

Fig 3.36 (b) LSTM Model

CHAPTER 4

RESULTS AND DISCUSSION

The goal of the project was to enhance the accuracy of the deep learning model by utilizing feature selection techniques. These techniques aimed to decrease the training time of the model while still retaining a significant number of effective features. Our proposed approach utilized the LSTM deep learning algorithm, which led to a notable increase in accuracy.

We utilized the LSTM deep learning algorithm, which is a type of recurrent neural network that is specifically designed to handle sequential data. LSTM is capable of learning long-term dependencies in data, making it ideal for predicting time-series data. By using LSTM, we were able to build a highly accurate model that could effectively predict outcomes based on a sequence of input data.

The application of LSTM contributed to the reduced the RMSE of the model, indicating the effectiveness of using deep learning algorithms in predictive modeling. Overall, our proposed approach offers a promising solution for enhancing the RUL prediction of deep learning models while reducing the duration of training.

4.1 Value of LSTM

	MSE	RMSE	MAE	Std_AE	S_score
LSTM	206.32	14.36	10.45	9.85	377.37

TABLE 4.1 Value of LSTM

	Train		Test		Valid	
	RMSE	R2	RMSE	R2	RMSE	R2
L R	56.1986931	0.34393613	54.2640090	0.35713062	34.8496222	0.29670649
	8681088	45919162	74691856	175493715	8223509	52591344
S V R	43.5644846	0.60576167	47.2671844	0.51222621	34.8496222	0.29670649
	4126648	0162996	9996112	67433573	8223509	52591344
R F R	15.4083031	0.95068224	44.3638300	0.57030821	30.6536180	0.45586841
	70220156	32050425	9860364	28383993	24631285	183607596

TABLE 4.2 All Features Without Historical Data

	Train		Test		Valid	
	RMSE	R2	RMSE	R2	RMSE	R2
L R	47.8631295	0.37926624	55.626100	0.324451997	34.2107010	0.32225799
	5545103	65007206	2397046	2933424	8960351	27041398
S V R	32.9154314	0.70643634	48.755163	0.481032423	25.9479122	0.61010712
	8706822	70171213	79159361	01470384	25366203	74546097
R F R	12.0368624	0.96074185	45.818382	0.541669771	26.5484035	0.59185237
	99525814	47981479	91793876	9315231	67823055	14819

TABLE 4.3 Data Without Useless Sensors, Without Historical Data

	Train		Test		Valid	
	RMSE	R2	RMSE	R2	RMSE	R2
L R	36.1227831	0.64610268	36.2475904	0.63599395	33.1252102	0.36458458
	8538548	09508146	3148782	67063242	6826295	97590453
S V R	32.3135243	0.71680634	32.7061504	0.70364710	34.0769217	0.32754818
	81002445	37968639	393512	78045101	35663475	60480176
R F R	9.78539254	0.97403003	26.6963818	0.80255091	34.0997127	0.32664839
	9750972	4737145	91435006	6397019	55388424	89294847

TABLE 4.4 Data Without Useless Sensors, With Historical Data

The conclusion of the above **Table 4.1,4.2,4.3,4.4** The Root Mean Square Error (RMSE) produced by the proposed LSTM (Long Short-Term Memory) method is lower compared to the RMSE of existing methods. Lower RMSE indicates that the proposed LSTM method provides more accurate predictions or fits the data better than existing methods.

Fig 4.1 The graph illustrates the relationship between the true Remaining Useful Life (RUL) values and the predicted RUL values. In predictive maintenance or reliability engineering, RUL refers to the estimated time until a machine or system will fail. By comparing true RUL values with predicted RUL values on the graph, you can assess the accuracy of a predictive model.

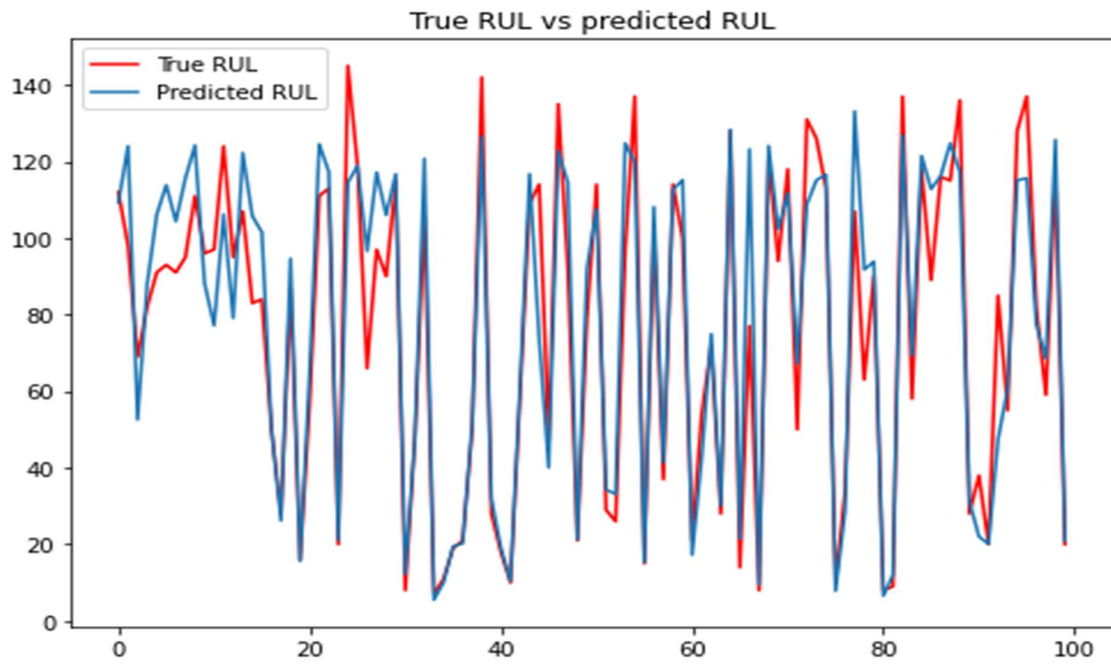


Fig 4.1 LSTM Model Graph

CHAPTER 5

CONCLUSION AND FUTURE WORKS

In conclusion, the project aimed to revolutionize engine health management in the aviation industry by leveraging deep learning techniques, particularly the Long Short-Term Memory (LSTM) algorithm, to predict the Remaining Useful Life (RUL) of Commercial Jet engines. Predicting the RUL of critical components like Jet engines is crucial for cost savings, safety, efficiency, and reliability in the aircraft industry. By accurately estimating the time until failure, operators can schedule maintenance activities more effectively, optimize engine performance, and prevent catastrophic failures.

Traditional methods such as regression models, prognostic models, survival analysis, and expert systems have limitations in capturing complex relationships and providing reliable predictions. Deep learning techniques offer a promising solution to overcome these challenges by handling non-linearity, temporal dependencies, and complex data patterns. LSTM, a type of recurrent neural network, excels in learning long-term dependencies in sequential data, making it well-suited for time-series prediction tasks like RUL estimation. Its ability to capture temporal dependencies and handle complex relationships between features contributes to more accurate and reliable predictions compared to traditional approaches.

The project proposed a methodology that involved data collection, preprocessing, feature engineering, and LSTM model building. By utilizing LSTM and feature selection techniques, the accuracy of the RUL prediction model was significantly enhanced, as evidenced by the reduced Root Mean Square Error (RMSE) and improved prediction performance.

The successful application of LSTM in RUL prediction opens up opportunities for proactive maintenance scheduling, cost reduction, and enhanced safety in the aviation industry. Future research could focus on further optimizing

deep learning models, exploring additional features, and integrating real-time data for more dynamic predictions.

REFERENCE

- [1] Que, Z., Jin, X. & Xu, Z. Remaining useful life prediction for bearings based on a gated recurrent unit. *IEEE Trans. Instrum. Meas.* **15**(10), 10–28 (2021).
- [2] Zhao, S. *et al.* Health state estimation and remaining useful life prediction of power devices subject to noisy and aperiodic condition monitoring. *IEEE Trans. Instrum. Meas.* **14**(6), 102–125 (2021).
- [3] Cao, Y. *et al.* Transfer learning for remaining useful life prediction of multi-conditions bearings based on bidirectional-GRU network. *Measurement* **178**(5), 109–127 (2021).
- [4] Li X, Ding Q, Sun J-Q. Remaining useful life estimation in prognostics using deep convolution neural networks. *ReliabEng Syst Saf*2018; 172:1–11.
- [5] Peng Y, Wang Y, Zi Y. Switching state-space degradation model with recursive filter/smoothing for prognostics of remaining useful life. *IEEE Trans Ind Inf* 2018. <https://doi.org/10.1109/TII.2018.2810284>. 1–1
- [7] Kalgren PW, Byington CS, Roemer MJ, Watson MJ. Defining phm, a lexical evolution of maintenance and logistics. 2006 IEEE Autotestcon. 2006. p. 353–8. <https://doi.org/10.1109/AUTEST.2006.283685>.
- [8] Peng Y, Wang Y, Zi Y. Switching state-space degradation model with recursive filter/smoothing for prognostics of remaining useful life. *IEEE Trans Ind Inf* 2018. <https://doi.org/10.1109/TII.2018.2810284>. 1–1

- [9] Zhao G, Zhang G, Ge Q, Liu X. Research advances in fault diagnosis and prognostic based on deep learning. Prognostics and System Health Management Conference (PHM-Chengdu). IEEE; 2016. p. 1–6. <https://doi.org/10.1109/PHM.2016.7819786>.
- [10] Chen XW, Lin X. Big data deep learning: challenges and perspectives. IEEE Access 2014; 2:514–25. <https://doi.org/10.1109/ACCESS.2014.2325029>.
- [11] Sateesh Babu G, Zhao P, Li X-L. Deep convolutional neural network-based regression approach for estimation of remaining useful life. Cham: Springer International Publishing; 2016. p. 214–28. https://doi.org/10.1007/978-3-319-32025-0_14. ISBN 978-3-319-32025-0
- [12] Zheng S, Ristovski K, Farahat A, Gupta C. Long short-term memory network for remaining useful life estimation. 2017 IEEE International Conference on Prognostics and Health Management (ICPHM). IEEE; 2017. p. 88–95.
- [13] Li X, Ding Q, Sun J-Q. Remaining useful life estimation in prognostics using deep convolution neural networks. ReliabEng Syst Saf2018; 172:1–11.
- [14] Erhan D, Manzagol P-A, Bengio Y, Bengio S, Vincent P. The difficulty of training deep architectures and the effect of unsupervised pre-training. Artificial Intelligence and Statistics. 2009. p. 153–60.

- [15] Glorot X, Bordes A, Bengio Y. Deep sparse rectifier neural networks. In: Gordon G, Dunson D, Dudík M, editors. Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics; vol. 15 of Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR; 2011. p. 315–23.
- [16] Kingma D.P., Ba J... Adam: a method for stochastic optimization. arXiv: 1412.6980 2014;
- [17] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 2014;15(1):1929–58.
- [18] Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. Proceedings of the 13th International Conference on Artificial Intelligence and Statistics. 2010. p. 249–56.
- [19] He K, Zhang X, Ren S, Sun J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. Proceedings of the IEEE international conference on computer vision. 2015. p. 1026–34.

APPENDICES

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
import sklearn
from sklearn.metrics import mean_squared_error, r2_score
import os
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
import random
import warnings
np.random.seed(34)
warnings.filterwarnings('ignore')
index_names = ['unit_number', 'time_cycles']
setting_names = ['setting_1', 'setting_2', 'setting_3']
sensor_names = ['s_{}'.format(i+1) for i in range(0,21)]
col_names = index_names + setting_names + sensor_names
dftrain =
pd.read_csv('Data/CMaps/train_FD001.txt',sep='s+',header=None,index_col=False,names=col_names)
dfvalid =
pd.read_csv('Data/CMaps/test_FD001.txt',sep='s+',header=None,index_col=False,names=col_names)
y_valid =
pd.read_csv('Data/CMaps/RUL_FD001.txt',sep='s+',header=None,index_col=False,names=['RUL'])
dfvalid.shape
train = dftrain.copy()
valid = dfvalid.copy()
train
print('Shape of the train dataset : ',train.shape)
```

```

print('Shape of the validation dataset : ',valid.shape)
print('Percentage of the validation dataset : ',len(valid)/(len(valid)+len(train)))
#Cheking the presence of Nan values
print('Total None values in the train dataset : ',train.isna().sum())
print("Total no of rows: ",int(train.loc[:,['unit_number']].describe().loc['count']))
da=train.loc[:,['unit_number','time_cycles']].groupby('unit_number').max()
da.describe().loc[['count','min','max','50%','mean']]
train.loc[:,['s_1']].describe().transpose()
max_time_cycles=train[index_names].groupby('unit_number').max()
plt.figure(figsize=(20,50))
ax=max_time_cycles['time_cycles'].plot(kind='barh',width=0.8,
stacked=True,align='center')
plt.title('Turbofan Engines LifeTime',fontweight='bold',size=30)
plt.xlabel('Time cycle',fontweight='bold',size=20)
plt.xticks(size=15)
plt.ylabel('unit',fontweight='bold',size=20)
plt.yticks(size=15)
plt.grid(True)
plt.tight_layout()
plt.show()
def add_RUL_column(df):
train_grouped_by_unit = df.groupby(by='unit_number')
max_time_cycles = train_grouped_by_unit['time_cycles'].max()
merged = df.merge(max_time_cycles.to_frame(name='max_time_cycle'),
left_on='unit_number',right_index=True)
merged["RUL"] = merged["max_time_cycle"] - merged['time_cycles']
merged = merged.drop("max_time_cycle", axis=1)
return merged
train = add_RUL_column(train)
train[['unit_number','RUL']]
#Rul analysis
maxrul_u = train.groupby('unit_number').max().reset_index()
maxrul_u.head()
# Compute the correlation matrix
corr = train.corr()
mask = np.triu(np.ones_like(corr, dtype=bool))
f, ax = plt.subplots(figsize=(10, 10))

```

```

cmap = sns.diverging_palette(230, 10, as_cmap=True)
# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0, square=True,
linewidths=.5, cbar_kws={"shrink": .5})
Sensor_dictionary={}
dict_list=[ "(Fan inlet temperature) (°R)",
"(LPC outlet temperature) (°R)",
"(HPC outlet temperature) (°R)",
"(LPT outlet temperature) (°R)",
"(Fan inlet Pressure) (psia)",
"(bypass-duct pressure) (psia)",
"(HPC outlet pressure) (psia)",
"(Physical fan speed) (rpm)",
"(Physical core speed) (rpm)",
"(Engine pressure ratio(P50/P2))",
"(HPC outlet Static pressure) (psia)",
"(Ratio of fuel flow to Ps30) (pps/psia)",
"(Corrected fan speed) (rpm)",
"(Corrected core speed) (rpm)",
"(Bypass Ratio) ",
"(Burner fuel-air ratio)",
"(Bleed Enthalpy)",
"(Required fan speed)",
"(Required fan conversion speed)",
"(High-pressure turbines Cool air flow)",
"(Low-pressure turbines Cool air flow)" ]
i=1
for x in dict_list :
Sensor_dictionary['s_'+str(i)]=x
i+=1
Sensor_dictionary
def plot_signal(df, Sensor_dic, signal_name):
plt.figure(figsize=(13,5))
for i in df['unit_number'].unique():
if (i % 10 == 0): #For a better visualisation, we plot the sensors signals of 20
units only
plt.plot('RUL', signal_name, data=df[df['unit_number']==i].rolling(10).mean())

```



```

plt.xlim(250, 0) # reverse the x-axis so RUL counts down to zero
plt.xticks(np.arange(0, 300, 25))
plt.ylabel(Sensor_dic[signal_name])
plt.xlabel('Remaining Useful Life')
plt.show()
for i in range(1,22):
try:
plot_signal(train, Sensor_dictionary,'s_'+str(i))
except:
pass
for x in sensor_names:
plt.figure(figsize=(13,7))
plt.boxplot(train[x])
plt.title(x)
plt.show()
train.loc[:, 's_1:'].describe().transpose()
from sklearn.model_selection import train_test_split
drop_labels = index_names+setting_names
X_train=train.drop(columns=drop_labels).copy()
X_train, X_test, y_train, y_test=train_test_split(X_train,X_train['RUL'],
test_size=0.3, random_state=42)
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
#Dropping the target variable
X_train.drop(columns=['RUL'], inplace=True)
X_test.drop(columns=['RUL'], inplace=True)
#Scaling X_train and X_test
X_train_s=scaler.fit_transform(X_train)
X_test_s=scaler.fit_transform(X_test)
#Conserve only the last occurrence of each unit to match the length of y_valid
X_valid =
valid.groupby('unit_number').last().reset_index().drop(columns=drop_labels)
#scaling X_valid
X_valid_s=scaler.fit_transform(X_valid)
print(X_valid_s.shape)
print(y_valid.shape)

```

```

sensor_names=['s_{}'.format(i) for i in range(1,22) if i not in
[1,5,6,10,16,18,19]]
pd.DataFrame(X_train_s,columns=['s_{}'.format(i) for i in
range(1,22)])[sensor_names].hist(bins=100, figsize=(18,16))
class Linear_Regression() :
def __init__( self, lr=0.01, iterations=150 ) :
self.lr = lr
self.iterations = iterations
def fit( self, X, Y ) :
self.l, self.p = X.shape
# weight initializer
self.W = np.zeros( self.p )
self.b = 0
self.X = X
self.Y = Y
# gradientlearning
for i in range( self.iterations ) :
self.weight_updater()
return self
def weight_updater( self ) :
Y_pred = self.predict( self.X )
# gradients
dW = - ( 2 * ( self.X.T ).dot( self.Y - Y_pred ) ) / self.l
db = - 2 * np.sum( self.Y - Y_pred ) / self.l
# new weights assigned
self.b = self.b - self.lr * db
self.W = self.W - self.lr * dW
return self
def predict( self, X ) :
# Y_pr=X.W+b
return X.dot( self.W ) + self.b
from sklearn.svm import SVR
import tensorflow as tf
regressor = SVR(kernel='rbf')
rf = RandomForestRegressor(max_features="sqrt", random_state=42)
#R2 score & RMSE & MAER
def evaluate(y_true, y_hat, label='test'):

```

```

mse = mean_squared_error(y_true, y_hat)
rmse = np.sqrt(mse)
variance = r2_score(y_true, y_hat)
print('{} set RMSE: {}, R2: {}'.format(label, rmse, variance))
#plot real data and the predicted one to make some comparison
def plot_predActual(y_test, y_test_hat):
indices = np.arange(len(y_test_hat))
wth= 0.6
plt.figure(figsize=(70,30))
true_values = [int(x) for x in y_test.values]
predicted_values = list(y_test_hat)
plt.bar(indices, true_values, width=wth,color='b', label='True RUL')
plt.bar([i for i in indices], predicted_values, width=0.5*wth, color='r',
alpha=0.7, label='Predicted RUL')
plt.legend(prop={'size': 40})
plt.tick_params(labelsize=40)
plt.show()
#all features, without historical data
#linear regression
lr=Linear_Regression() #Instantiation
lr.fit(X=X_train_s, Y=y_train) #Fitting
y_lr_train = lr.predict(X_train_s) #Prediction on train data
evaluate(y_train,y_lr_train, label='train')
y_lr_test = lr.predict(X_test_s) #Prediction on test data
evaluate(y_test, y_lr_test, label='test')
y_lr_valid= lr.predict(X_valid_s) #Prediction on validation data
evaluate(y_valid, y_lr_valid, label='valid')
#support vector regression
regressor.fit(X_train_s, y_train) # fitting
y_svr_train = regressor.predict(X_train_s) #Prediction on train data
evaluate(y_train,y_svr_train, label='train')
y_svr_test = regressor.predict(X_test_s) #Prediction on test data
evaluate(y_test, y_svr_test, label='test')
y_svr_valid= lr.predict(X_valid_s) #Prediction on validation data
evaluate(y_valid, y_svr_valid, label='valid')
#random forest regression
# gridsearch goes here

```

```

rf.fit(X_train_s, y_train)
# predict and evaluate
y_rf_train = rf.predict(X_train_s)
evaluate(y_train, y_rf_train, label='train')
y_rf_test = rf.predict(X_test_s)
evaluate(y_test, y_rf_test, label='test')
y_rf_valid = rf.predict(X_valid_s)
evaluate(y_valid, y_rf_valid, label='valid')
plot_predActual(y_valid, y_rf_valid)
#Data without useless sensors, without historical data
print('maximum of y_train : ', y_train.max())
print('maximum of y_test : ', y_test.max())
print('maximum of y_valid : ', y_valid.max())
plt.hist(y_test)
drop_labels2=['s_1', 's_5', 's_6', 's_10', 's_16', 's_18', 's_19']
X_train_2=X_train.drop(columns=drop_labels2, axis=1) # drop the constant
columns from the train dataset
X_test_2=X_test.drop(columns=drop_labels2, axis=1) # drop the constant
columns from the test dataset
X_train_2_s=scaler.fit_transform(X_train_2) #scaling X_train_2
X_test_2_s=scaler.fit_transform(X_test_2) #scaling X_test_2
y_train_clip=y_train.clip(upper=195) #Clipping y_train to have 195 as the
maximum value
X_valid_2=X_valid.drop(columns=drop_labels2, axis=1) # drop the constant
columns from the validation dataset
X_valid_2_s=scaler.fit_transform(X_valid_2) #scaling X_valid_2
#linear regression
lr=Linear_Regression()
lr.fit(X=X_train_2_s, Y=y_train_clip)
y_lr_train = lr.predict(X_train_2_s)
evaluate(y_train_clip, y_lr_train, label='train')
y_lr_test = lr.predict(X_test_2_s)
evaluate(y_test, y_lr_test, label='test')
y_lr_valid = lr.predict(X_valid_2_s)
evaluate(y_valid, y_lr_valid, label='valid')
#support vector regression
regressor.fit(X_train_2_s, y_train)

```

```

y_svr_train = regressor.predict(X_train_2_s)
evaluate(y_train_clip,y_svr_train, label='train')
y_svr_test = regressor.predict(X_test_2_s)
evaluate(y_test, y_svr_test, label='test')
y_svr_valid = regressor.predict(X_valid_2_s)
evaluate(y_valid, y_svr_valid, label='valid')
#random froest regression
rf.fit(X_train_2_s, y_train_clip)
# predict and evaluate
y_rf_train = rf.predict(X_train_2_s)
evaluate(y_train_clip,y_rf_train, label='train')
y_rf_test = rf.predict(X_test_2_s)
evaluate(y_test, y_rf_test, label='test')
y_rf_valid = rf.predict(X_valid_2_s)
evaluate(y_valid, y_rf_valid, label='valid')
!pip install xgboost
import xgboost
xgb = xgboost.XGBRegressor(n_estimators=110, learning_rate=0.02,
gamma=0, subsample=0.8,colsample_bytree=0.5, max_depth=3)
xgb.fit(X_train_2_s, y_train_clip)
y_xgb_train = xgb.predict(X_train_2_s)
evaluate(y_train_clip,y_xgb_train, label='train')
y_xgb_test = xgb.predict(X_test_2_s)
evaluate(y_test, y_xgb_test, label='test')
y_xgb_valid = xgb.predict(X_valid_2_s)
evaluate(y_valid, y_xgb_valid, label='valid')
plot_predActual(y_valid, y_rf_valid)
#Data without useless sensors, with historical data
df=train.copy()
for x in X_train_2.columns:
df[x+'_rm']=0
df.columns
drop_labels2=['s_1', 's_5','s_6','s_10', 's_16', 's_18', 's_19']
df=df.drop(columns=setting_names+drop_labels2+['RUL'], axis=1)
X_valid_3=valid.drop(columns=index_names+setting_names+drop_labels2,
axis=1)
def update_rolling_mean(data, mask):

```

```

for x, group in mask.groupby("unit_number"):
    for x in X_train_2.columns:
        data.loc[group.index[10:], x+"_rm"] = data.loc[group.index,
        x].rolling(10).mean()[10:]
        data.loc[group.index[:10], x+"_rm"] = data.loc[group.index[:10], x]
    update_rolling_mean(df, df)
    update_rolling_mean(X_valid_3, valid)
    X_valid_3=X_valid_3.fillna(0)
    df.iloc[-1,-14:]=df.iloc[-2,-14:]
    X_valid_3.iloc[-1,-14:]=X_valid_3.iloc[-2,-14:]
    df.iloc[-1,-14:]=df.iloc[-2,-14:]
    X_valid_3.iloc[-1,-14:]=X_valid_3.iloc[-2,-14:]
    train_tm=df
    train_tm=train_tm.drop(columns=index_names, axis=1)
    X_train_tm, X_test_tm, y_train_tm,
    y_test_tm=train_test_split(train_tm,train['RUL'].clip(upper=195),
    test_size=0.35, random_state=42)
    X_train_tm_s=scaler.fit_transform(X_train_tm)
    X_test_tm_s=scaler.fit_transform(X_test_tm)
    X_val3=pd.concat([valid['unit_number'],X_valid_3],axis=1)
    X_val3 =
    X_val3.groupby('unit_number').last().reset_index().drop(columns=['unit_numbe
    r'])
    X_val3_s=scaler.fit_transform(X_val3)
    #linear regression
    lr=Linear_Regression()
    lr.fit(X_train_tm_s, y_train_tm)
    y_lr_train = lr.predict(X_train_tm_s)
    evaluate(y_train_tm,y_lr_train, label='train')
    y_lr_test = lr.predict(X_test_tm_s)
    evaluate(y_test_tm, y_lr_test, label='test')
    y_lr_valid = lr.predict(X_val3_s)
    evaluate(y_val3, y_lr_valid, label='valid')
    #support vector regression
    regressor.fit(X_train_tm_s, y_train_tm)
    y_svr_train = regressor.predict(X_train_tm_s)
    evaluate(y_train_tm,y_svr_train, label='train')

```

```

y_svr_test = regressor.predict(X_test_tm_s)
evaluate(y_test_tm, y_svr_test, label='test')
y_svr_valid = regressor.predict(X_valid_s)
evaluate(y_valid, y_svr_valid, label='valid')
#random forest regression
rf = RandomForestRegressor(max_features="sqrt", random_state=42)
rf.fit(X_train_tm_s, y_train_tm)
# predict and evaluate
y_hat_train = rf.predict(X_train_tm_s)
evaluate(y_train_tm, y_hat_train, label='train')
y_hat_test = rf.predict(X_test_tm_s)
evaluate(y_test_tm, y_hat_test, label='test')
y_hat_valid = rf.predict(X_valid_s)
evaluate(y_valid, y_hat_valid, label='valid')
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit
from sklearn.ensemble import RandomForestRegressor
estimator = RandomForestRegressor()
param_grid = {
'n_estimators': [50,90,120],
'max_depth' : [8,9,10],
}
grid = GridSearchCV(estimator, param_grid, n_jobs=-1, cv=3)
grid.fit(X_train_tm_s, y_train_tm)
print(grid.best_score_ , grid.best_params_)
#max_features=5,
rf=RandomForestRegressor(n_estimators=90, max_depth=10, n_jobs=-1,
random_state=42)
rf.fit(X_train_tm_s, y_train_tm)
# predict and evaluate
y_hat_train = rf.predict(X_train_tm_s)
evaluate(y_train_tm, y_hat_train, label='train')
y_hat_test = rf.predict(X_test_tm_s)
evaluate(y_test_tm, y_hat_test, label='test')
y_hat_valid = rf.predict(X_valid_s)
evaluate(y_valid, y_hat_valid, label='valid')
#The best Rmse score out the above trained models for validation set is 34.49

```

```

# Params found using Bayesian Optimisation
xgb = xgboost.XGBRegressor(n_estimators=50,
max_depth=6,
learning_rate=0.1,
reg_lambda=0.02,
gamma=0.4,
random_state=42)
xgb.fit(X_train_tm_s, y_train_tm)
# predict and evaluate
y_hat_train = xgb.predict(X_train_tm_s)
evaluate(y_train_tm, y_hat_train, 'train')
y_hat_test = xgb.predict(X_test_tm_s)
evaluate(y_test_tm, y_hat_test)
y_hat_valid = xgb.predict(X_valid_s)
evaluate(y_valid, y_hat_valid, label='valid')
random_state = 1
def LSTM_preprocessing(data, feature_to_split, target, window_size = 30,
feature_to_drop=[], Ceil_RUL=None, shift = 1):
"""
A preprocessing function for generating LSTM 3D data input, where output first
dimension is the number of batches to be made from data, second dimension is
window size (timesteps),
and the third dimension is the number of features.
input:
- data: array, data to process in windows (DataFrame)
- feature_to_split: the name of the single feature which elements are used for
selecting rows to split in windows (list/str/int)
- target: the column of data which is the target of prediction (series)
- window_size: the size of the window we want for the LSTM (int)
- feature_to_drop: self explanatory, could be the feature_to_split (list)
- Ceil_RUL: if int, the max RUL we want, will change all greater RULs to it
(int)
- shift: distance between a window and another. Data will repeat if shift <
window_size (int)
output: an 3D ndarray which is divided in windows
"""
assert feature_to_split in data.columns , f"feature_to_split not in data features"

```



```

assert type(feature_to_drop) == list , f'feature_to_drop must be a list'
num_split = np.unique(data[feature_to_split])
num_features = data.shape[1]-len(feature_to_drop)
processed_data = np.zeros([0, window_size,num_features])
processed_target = np.zeros(0)
for i in num_split:
    data_temp = data[data[feature_to_split] == i].drop(feature_to_drop, axis=1)
    assert len(data_temp) - window_size>0, f'Window size greater than data at unit
    number: {i}'
    n_batches = (len(data_temp) - window_size)//shift + 1
    singular_output_data = np.zeros([n_batches, window_size,num_features])
    singular_processed_target = np.zeros([n_batches])
    for n_batch in range(len(data_temp) - window_size,-1,-shift):
        n_batches-=1
        singular_output_data[n_batches] = data_temp[n_batch:n_batch+window_size]
        singular_processed_target[n_batches] = target[data[feature_to_split] ==
        i].iloc[n_batch+window_size-1]
        processed_data = np.append(processed_data, singular_output_data, axis=0)
        processed_target = np.append(processed_target, singular_processed_target,
        axis=0)
    if Ceil_RUL is not None:
        processed_target[processed_target>Ceil_RUL] = Ceil_RUL
    return processed_data, processed_target
def test_preprocessing(data, feature_to_split, feature_to_drop, window_size =
30):
    """

```

A preprocessing function for generating LSTM 3D data input for test, where output first dimension is the number of batches made from data, second dimension is window size (timesteps), and the third dimension is the number of features.

This function will take only the last window data from every feature_to_split in order to make predictions.

input:

- data: array, data to process in windows (DataFrame)
- feature_to_split: the single feature which is used as index, list/str/int
- feature_to_drop: self explanatory (list)

output: a 3D array which contains the last window of data for each

feature_to_split

"""

```
assert(type(feature_to_drop) == list), f"feature_to_drop must be a list"
```

```
num_split = np.unique(data[feature_to_split])
```

```
num_features = data.shape[1]-len(feature_to_drop)
```

```
processed_data = np.zeros([0, window_size,num_features])
```

```
for i in num_split:
```

```
    data_temp = data[data[feature_to_split] == i].drop(feature_to_drop, axis=1)
```

```
    singular_output_data = np.zeros([1, window_size,num_features])
```

```
    singular_output_data[0] = data_temp[-window_size:]
```

```
    processed_data = np.append(processed_data, singular_output_data, axis=0)
```

```
return processed_data
```

```
train_FD001 = dftrain.copy()
```

```
test_FD001 = dfvalid.copy()
```

```
RUL_FD001 = pd.read_csv("Data/CMaps/RUL_FD001.txt", header=None)
```

```
columns =
```

```
['unit_number','time_(cycles)','operational_setting_1','operational_setting_2','operational_setting_3','T2_Total_temperature_at_fan_inlet_(°R)','T24_Total_temperature_at_LPC_outlet_(°R)','T30_Total_temperature_at_HPC_outlet_(°R)','T50_Total_temperature_at_LPT_outlet_(°R)','P2_Pressure_at_fan_inlet_(psia)','P15_Total_pressure_in_bypass-duct_(psia)','P30_Total_pressure_at_HPC_outlet_(psia)','Nf_Physical_fan_speed_(rpm)',
```

```
'Nc_Physical_core_speed_(rpm)','epr_Engine_pressure_ratio_(P50/P2)','Ps30_Static_pressure_at_HPC_outlet_(psia)','phi_Ratio_of_fuel_flow_to_Ps30_(pps/psi)','NRf_Corrected_fan_speed_(rpm)','NRc_Corrected_core_speed_(rpm)','BPR_Bypass_Ratio','farB_Burner_fuel-air_ratio','htBleed_Bleed_Enthalpy','Nf_dmd_Demanded_fan_speed_(rpm)','PCNfR_dmd_Demanded_corrected_fan_speed_(rpm)','W31_HPT_coolant_bleed_(lbm/s)','W32_LPT_coolant_bleed_(lbm/s)']
```

```
train_FD001.columns = columns
```

```
test_FD001.columns = columns
```

```
nu_column = ['operational_setting_3',
```

```
'T2_Total_temperature_at_fan_inlet_(°R)','P2_Pressure_at_fan_inlet_(psia)',
```

```
"P15_Total_pressure_in_bypass-duct_(psia)",
```

```

'epr_Engine_pressure_ratio_(P50/P2)', 'farB_Burner_fuel-air_ratio',
'Nf_dmd_Demanded_fan_speed_(rpm)',
'PCNfR_dmd_Demanded_corrected_fan_speed_(rpm)']
train_FD001.drop(columns=nu_column, inplace=True)
test_FD001.drop(columns=nu_column, inplace=True)
train_FD001['RUL'] =
train_FD001.groupby('unit_number')['time_(cycles)'].apply(lambda x: x.max()-
x).values
train_FD001.drop("NRc_Corrected_core_speed_(rpm)", axis=1, inplace=True)
test_FD001.drop("NRc_Corrected_core_speed_(rpm)", axis=1, inplace=True)
#Standard scale the data, except from categorical numerical columns such as
"unit_number", which is re-added after standard scaling. "time_(cycles)" is
dropped.
scaler = StandardScaler()
train_scaled = pd.DataFrame(np.c_[train_FD001[["unit_number"]],
scaler.fit_transform(train_FD001.drop(["unit_number", "time_(cycles)",
"RUL"], axis=1))])
test_scaled = pd.DataFrame(np.c_[test_FD001[["unit_number"]],
scaler.transform(test_FD001.drop(["unit_number", "time_(cycles)"], axis=1))])
pd.DataFrame(train_scaled).head()
print("Minimum window size possible for train: ",
train_FD001[["unit_number",
"time_(cycles)"]].groupby("unit_number").max().values.min())
print("Minimum window size possible for test: ", test_FD001[["unit_number",
"time_(cycles)"]].groupby("unit_number").max().values.min())
window_size = 30
#Column 0 is "unit_number", it used to split the data by engine then dropped, as
we do not want this feature for prediction
X, y = LSTM_preprocessing(train_scaled, 0, train_FD001['RUL'],
window_size, [0], Ceil_RUL=150, shift = 1)
#Manually shuffle the dataset to ensure reproducibility, setting shuffle=false
later on the model
np.random.seed(1)
shuffled_index = np.random.permutation(len(X))
X, y = X[shuffled_index], y[shuffled_index]
test_processed = test_preprocessing(test_scaled, 0, [0],
window_size=window_size)

```

```

print("X shape: ", X.shape)
print("y shape: ", y.shape)
print("test_processed shape: ", test_processed.shape)
# print("RUL_FD001 shape: ", RUL_FD001.shape)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state = 1)
#X_test, y_test will be used as validation data
print("X_train shape: ", X_train.shape)
print("X_test shape: ", X_test.shape)
print("y_train shape: ", y_train.shape)
print("y_test shape: ", y_test.shape)
def s_score(RUL_true, RUL_predicted, a1=10, a2=13):
    """
    Input: 1D np.array
    Late prediction (negative diff) are more penalized than early prediction
    The lower the score the better
    """
    diff = np.array(RUL_true) - np.array(RUL_predicted)
    s = sum(np.exp(-diff[diff<0]/a1)-1) + sum(np.exp(diff[diff>=0]/a2)-1)
    return s
def s_score_model(RUL_true, RUL_predicted, a1=10, a2=13):
    """
    Input: 1D tf.tensor
    Late prediction (negative diff) are more penalized than early prediction
    """
    diff = tf.subtract(RUL_true, RUL_predicted)
    s = tf.reduce_sum(tf.where(diff < 0, tf.exp(-diff/a1)-1, tf.exp(diff/a2)-1), axis=0)
    return s
initializer = tf.initializers.GlorotNormal(seed=1)
model = Sequential([
layers.LSTM(128, input_shape = (X.shape[1], X.shape[2]),
return_sequences=True, activation = "tanh", bias_initializer="ones",
kernel_initializer=initializer,
kernel_regularizer=tf.keras.regularizers.L1L2(l1=1e-5, l2=1e-4),
),
layers.LSTM(64, activation = "tanh", bias_initializer="ones",
kernel_initializer=initializer,

```

```

kernel_regularizer=tf.keras.regularizers.L1L2(l1=1e-4, l2=1e-3),
),
layers.Dropout(0.5, seed=random_state),
layers.Dense(32, activation = "relu", kernel_initializer=initializer,
),
layers.Dense(8, activation = "relu", kernel_initializer=initializer,
),
layers.Dense(1)
])
opt = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(loss = s_score_model, metrics=["mse"], optimizer = opt)
history = model.fit(X_train, y_train, epochs =30 ,
validation_data = (X_test, y_test),
shuffle = False,
batch_size = 96, verbose = 2)
RUL_predicted = model.predict(test_processed)
MSE = mean_squared_error(RUL_FD001, RUL_predicted)
MAE = np.abs(RUL_FD001 - RUL_predicted).values.mean()
std_AE = np.abs(RUL_FD001 - RUL_predicted).values.std()
print("MSE: ", MSE.round(2))
print("RMSE: ", np.sqrt(MSE).round(2))
print("MAE: ", MAE.round(2))
print("std_AE: ", std_AE.round(2))
print("s_score: ", s_score(RUL_FD001, RUL_predicted).round(2))
#True RUL vs predicted RUL
fig, axes = plt.subplots(1, figsize = (7,5))
fig.tight_layout()
axes.plot(RUL_FD001, label = "True RUL", color = "red")
axes.plot(RUL_predicted, label = "Predicted RUL")
axes.set_title("True RUL vs predicted RUL")
axes.legend()
plt.show()

```