Aircraft Maintenance Prediction Using Deep Learning

Mama Mohammed Kebede

A Thesis Submitted to the Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

June, 2023

Adama, Ethiopia

Aircraft Maintenance Prediction Using Deep Learning

Mama Mohammed Kebede

Advisor: Getinet Yilma, PhD

A Thesis Submitted to the Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies
Adama Science and Technology University

June, 2023
Adama, Ethiopia

APPROVAL PAGE

I, the advisors of the thesis entitled "**Aircraft Maintenance Prediction using Deep Learning**" and developed by Mama Mohammed Kebede, hereby certify that the recommendation and suggestions made by the board of examiners are appropriately incorporated into the final version of the thesis.

Dr. Getinet Yilma		
Major Advisor/Supervisor	Signature	Date

We, the undersigned, members of the Board of Examiners of the thesis by Mama Mohammed Kebede have read and evaluated the thesis entitled "**Aircraft Maintenance Prediction using Deep Learning**" and examined the candidate during the open defense. This is, therefore, to certify that the thesis is accepted for partial fulfillment of the requirement of the degree of Master's of Science in Computer Science and Engineering.

Chair Person	Signature	Date
Internal Examiner	Signature	Date
External Examiner	Signature	Date

Finally, approval and acceptance of the thesis is contingent upon submission of its final copy to the Office of Postgraduate Studies (OPGS) through the Department Graduate Council (DGC) and School Graduate Committee (SGC).

Department Head	Signature	Date
School Dean	Signature	Date
Office of Postgraduate Studies, Dean	Signature	Date

DECLARATION

I hereby declare that this Master's Thesis entitled "**Aircraft Maintenance Prediction Using Deep Learning**" is my original work. That is, it has not been submitted for the award of any academic degree, diploma, or certificate in any other university. All sources of materials that are used for this thesis have been duly acknowledged through citation.

Mama Mohammed	_____	_____
Name of Student	Signature	Date

RECOMMENDATION OF ADVISORS/SUPERVISORS

I, the advisor of this thesis, hereby certify that I have read the revised version of the thesis entitled "**Aircraft Maintenance Prediction Using Deep Learning**" prepared under my guidance by Mama Mohammed Kebede submitted in partial fulfillment of the requirements for the degree of Master's of Science in Computer Science and Engineering. Therefore, I recommend the submission of a revised version of the thesis to the department following the applicable procedures.

Dr. Getinet Yilma	_____	_____
Major Advisor/Supervisor	Signature	Date
_____	_____	_____
Co-Advisor/Co-Supervisor	Signature	Date

ACKNOWLEDGEMENT

I would like to start by expressing my sincere gratitude to **ALLAH**, the Most Merciful and Compassionate, for giving me the fortitude, direction, and blessings along my path of finishing my thesis.

I am incredibly grateful to my advisor, Dr. Getinet Yilma, for his invaluable guidance, unwavering encouragement, and profound knowledge. He has provided positive encouragement to finish this thesis. It has been a great pleasure and honor to have him as my advisor.

I extend my heartfelt appreciation to all the individuals who have contributed to the success of my thesis. Their contributions, both big and small, have made a significant impact on the outcome of my research. I am grateful to Arefat Hyredin, Getacher Ashebir for their assistance, guidance, and support in various stages of my work. Their expertise, insightful suggestions, and constructive criticism have played a vital role in shaping my ideas and refining my research.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ACMS	- Aircraft Condition Monitoring System
ASIAS	- Aviation Safety Information Analysis and Sharing
AUC	- Area Under Curve
CNN	- Convolutional Neural Network
DBN	-Deep Belief Network
FAA	- Federal Aviation Administration
GRU	- Gated Recurrent Unit
LSTM	- Long Short-Term Memory
MHSA	-MultiHead Self Attention
MTS	- Multivariate Time Series
NGAFID	- National General Aviation Flight Information Database
PR	- Precision-Recall
RUL	- Remaining Useful Life
ReLU	- Rectified Linear Unit
RNN	- Recurrent Neural Network
ROC	- Receiver Operating Characteristic
TCN	- Temporal Convolutional Network
IoTs	- Internet of Things

ABSTRACT

Now days, advancement in Machine Learning and Artificial Intelligence many sectors are using them to enhance their service. For Aviation industry one of the highest expenses is that the cost for aircraft maintenance so now days, it is witnessed that a growing interest in implementing predictive maintenance techniques to improve aircraft reliability, reduce unscheduled downtime, and enhance safety.

This thesis focuses on the application of deep learning methodologies for aircraft predictive maintenance, specifically exploring the effectiveness of the Temporal Convolutional Network with Multi-Head Self-Attention (TCN-MHSA) model in comparison to the Convolutional Networks (CNN) and other deep learning architectures. Our proposed study utilizes a benchmark dataset obtained from the National General Aviation Flight Information Database (NGAFI) for training and evaluation. Since the full data set is very huge and very expensive to train a model, we selected flights which are with 3 days before and after maintenance which includes the 5 maintenance issues.

We proposed TCN-MHSA model which is introduced as a novel architecture capable of capturing both temporal and spatial dependencies in the data, utilizing a combination of convolutional layers, multi-head self-attention mechanisms.

To evaluate the performance of the TCN-MHSA model, comprehensive experiments are conducted and compared against the CNN-MHSA and CNN-LSTM architectures. The evaluation metrics include area under the Receiver Operating Characteristic curve (AUC-ROC), area under the Precision-Recall curve (AUC-PR) and along with Accuracy. Our results indicate that the TCN-MHSA model outperforms both the CNN-MHSA and CNN-LSTM architectures in terms of predictive maintenance performance. The TCN-MHSA model achieves higher AUC-PR, AUC-ROC and Accuracy across the 5 maintenance categories.

***Keywords**: Predictive maintenance, Deep learning, aircraft maintenance, TCN-MHSA model, CNN, LSTM, NGAFI dataset.*

CHAPTER ONE

1. INTRODUCTION

1.1. Background of the Study

In recent years, the global business practices are being changed by the advancement in technology. It is hard to say there is a business sector which is not affect by the advancement of technology. Advancement in technologies such as Artificial Intelligence or Machine learning have transformed the lives of many people. Artificial Intelligence (AI) is not a recent innovation, but rather it has grown exponentially in the recent years (Makhija & Chacko, 2021).

One sector that continues to enhance its offerings as technology develops is the aviation sector. Bigger aviation industries employ modern AI algorithms in their modern planes and services for their client's safety. These modern airplanes come with several sensors that generate a large number of readings for tracking the health of aircraft systems and parts. The engine of a Boeing 787, for instance, is continuously checked for roughly 1000 characteristics, accumulating 20 terabytes of data per flying hour (Ning et al., 2021). Such data are gathered by the Aircraft Condition Monitoring System (ACMS) for offline post-flight study of the aircraft.

Maintenance costs can make up a substantial portion of an airline's expenses, with historical estimates indicating that they typically range from 10% to 15% of the overall expenditure incurred by airlines (Verhagen & De Boer, 2018). Nowadays, aircraft systems are able to detect faults in a real-time to ensure the safety of the airplane(Hermawan et al., 2020). But it is necessary to make a foreseen prediction to let the user know how the airplane will perform in the future.

One way to do this is to anticipate how long the parts of the airplane will last so that the user has time to repair or replace it. With the growing demand and complexity of modern aircraft, it is important to find efficient ways of predicting maintenance issues before they become critical or lead to failure.

Predictive maintenance is a technique which is applied over various set of fields like in a factory they would use predictive maintenance system to predict how long and which component or machinery is going to need a maintenance. The use of predictive maintenance techniques can lower costs, lengthen the performance and lifespan of machines, minimize risk, and improve safety (Adhikari et al., n.d.; Yang et al., 2022; Xu et al., 2019).

Predictive maintenance is an emerging technique that utilizes machine learning or AI algorithms to predict when equipment will fail, allowing maintenance teams to intervene before the equipment breaks down. Predictive maintenance offers a comprehensive approach to evaluating the current and future health (diagnostics and prognostics) of the entire system and generates operational, maintenance, and logistical planning (Xu et al., 2019).

Recently advanced deep learning techniques have been used to characterize complex patterns in larger datasets, providing accurate predictions for almost any kind of data-driven process. Many studies are being held to develop an accurate algorithm to estimate the remaining useful life of many parts of machineries like automobile, aircraft and many factories equipment. Big data analytics with a deep learning approach has attracted an increasing attention in these studies.

Deep learning can be used to automate aircraft maintenance visual inspection (Bouarfa et al., 2020) and also Remaining-Useful-Life (RUL) prognostics for aircraft components and systems in one of the focuses of recent studies in the area. There have been few studies in the developing RUL prognostics for aircraft components and system such as the RUL prognostics of aircraft cooling unit's, gear brakes and electro-mechanical actuators and batteries using different deep learning algorithms (Lee & Mitici, 2023).

Due to lack of public data set for few studies are being conducted in this field even if there a high potential of developing a better predictive system which could lower the cost for maintenance and save lots amount expenses for the aviation industry. This study's primary objective is to build a predictive model by employ a variety of deep-learning techniques to achieve a better model.

1.2. Motivation of the Study

One of the biggest expenses for airlines is maintenance of their aircraft. Therefore, lowering this expense would increase the airlines' profitability. The aviation industry is highly reliant on the availability of aircraft, and reducing downtime is a key priority for airlines and other aircraft operators. Studies on Predictive maintenance could help to identify potential equipment failures before they occur, allowing maintenance teams to intervene before a failure occurs, reducing downtime and increasing aircraft availability.

In addition to reducing downtime, such study can also help optimize maintenance schedules and reduce maintenance costs. By analyzing large amounts of data from aircraft sensors and other monitoring devices, deep learning algorithms can identify patterns that indicate when maintenance is required, allowing maintenance teams to optimize maintenance schedules and minimize the number of unnecessary maintenance interventions. This can lead to substantial cost savings for airlines and other aircraft operators. So, conducting research on how to reduce the cost of aircraft maintenance is necessary.

Such studies have been made on predictive maintenance on other sectors like factories but due to lack of public dataset of the aviation industry not much studies are made. Since the availability of real flight data set many of the study made on simulation dataset might not perform well. By the dataset obtained from NGAFID which is a real dataset meaning the dataset is not a dataset from a simulation aircraft but rather from a real aircraft. The motivation of this study is to experiment and come up with a better deep learning model which would work for a real flight scenario.

1.3. Statement of the Problem

Airlines perform scheduled maintenance, or which is commonly known as "*conventional maintenance*" send an aircraft in for inspection after a certain number of flights or following a crew report. There are some shortcomings to this traditional maintenance technique. The main disadvantage is that it is reactive, which means that it wouldn't be brought in for inspection unless the plane had completed its number of flights or there was some other report concerning it. Aircraft that are in good functioning order are called in for inspection, which costs the corporation money that it could have made by flying the aircraft as well as money for performing the examination.

Therefore, the development of a predictive maintenance system would be necessary to address the issues stated above. Due to a lack of publicly available data, very few studies have been conducted in this field. Many of studies made on this area use datasets which are generated with simulations or in laboratory settings. This dataset does not include or the mostly on focus on the aircraft data than other factor like weather condition, seasons, pilots, and flight patterns. So, it is difficult to say that the model trained with simulation would perform well on a real fight dataset.

1.4. Research Questions

The research will try to address the following questions.

RQ1. What are the various methods that can be employed for aircraft maintenance prediction?

RQ2. Which features and parameters are most effective in predicting maintenance needs in an aircraft?

RQ3. What is the most effective deep learning model or combination of models that improves the accuracy of aircraft maintenance prediction?

1.5. Objective

1.5.1. General Objective

The general objective of the study is to develop a predictive maintenance model using deep learning.

1.5.2. Specific Objective

The following specific objectives are addressed to achieve the general objective.

- To collect and arrange all data necessary for developing the model.
- Extract flight data features that mainly focus on potential failure target variables.
- To design a pre-processing technique to enhance the quality of the training data.
- To design the proposed model.
- To train the proposed model with the data sets.
- Evaluate the performance of the proposed method on benchmark flight dataset.

1.6. Scope and Limitation of the Study

1.6.1. Scope of the Study

This study mainly focuses on developing a better a maintenance predictive model using deep learning for aircraft. So, the main goal or focus of this study is to develop an accurate predictive system using deep learning approach.

The scope of the thesis work within the given time and resource includes: -

- On this study the model will be trained only on the NGAFID Dataset.
- It uses only a Deep Learning approach to extract features.
- It only works for predicting common 5 types of aircraft maintenance issues which are listed below: -
    - Intake Gasket Leak/Damage
    - Rocker Cover Leak/Loose/Damage
    - Baffle Crack/Damage/Loose/miss
    - Intake Tube/bolt/seal/boot lose or damage
    - Baffle Plug needs Repair/Replace

1.6.2. Limitation of the Study

Due to lack of public dataset of the aviation industry this study uses a dataset restricted to general aviation which may not be directly transferable to commercial aviation, which is one drawback of the study. Commercial aviation may have quite different operating conditions, maintenance procedures, and equipment than general aviation. As a result, given the scope of operations and the complexity of the equipment in commercial aviation, the predictive maintenance algorithms created using a general aviation dataset would not work as well there. This constraint emphasizes the need for additional study using commercial aviation-specific datasets.

1.7. Significance of the Study

1.7.1. Significance of the Study

The aviation sector will be significantly impacted by this study. Reduced downtime, lower maintenance costs, improved safety, more precise maintenance interventions, and a more sustainable aviation industry are some of the potential advantages. Airline companies and other aircraft operators place a high focus on minimizing downtime because the aviation sector depends heavily on the availability of aircraft. This study suggested predictive maintenance model can assist in identifying probable equipment problems before they happen, allowing maintenance personnel to take action before a failure occurs, minimizing downtime and boosting aircraft availability.

This study can save maintenance costs and decrease downtime in addition to helping to improve maintenance schedules. Deep learning algorithms can analyze massive amounts of data from aircraft sensors and other monitoring devices to find patterns that indicate when maintenance is necessary. This enables maintenance teams to optimize maintenance schedules and reduce the number of pointless maintenance interventions. For airlines and other aircraft operators, this may result in significant financial savings.

Another important advantage of this study is that it will increase safety. Maintenance teams can handle possible safety issues by addressing them before they become major problems by anticipating equipment breakdowns before they happen. This may lower the possibility of mishaps and raise industrywide safety standards, resulting in a more resilient aviation sector.

1.8. Organization of the Study

This research work is divided into seven chapters, which are briefly discussed below.

- ***Chapter One:*** This chapter provides an outline of the research work as well as justifications for how and what will be accomplished.
- ***Chapter Two:*** This chapter explains the theoretical background, literature review and related works regarding Predictive maintenance.
- ***Chapter Three:*** This chapter covers research methodologies such as data collection, data preprocessing, design tools, development frameworks and platforms, and evaluation methods.
- ***Chapter Four:*** Using different block diagrams, pseudo-codes, and flow charts of the implementation with corresponding mathematical descriptions, this chapter aims to provide comprehensive information on how the proposed solution attempts to address the problem.
- ***Chapter Five:*** This chapter explains how to implement the proposed solution to the problem into action. It also contains a code snippet for the proposed solutions as well as the environment setup.
- ***Chapter Six:*** The outcomes of the proposed solution are discussed in this chapter, along with a comparison of the results utilizing figures, graphs, and tables.
- ***Chapter Seven:*** The research is summarized in this chapter. The conclusion of the work is given, as well as suggested future works.

CHAPTER TWO

2. LITERATURE REVIEW AND RELATED WORKS

2.1. Introduction

The term "predictive maintenance," which is also referred to as "condition-based maintenance," refers to a maintenance approach that has been used in the industrial sector since the 1990s. However, despite the fact that its history has not been formally recorded, predictive maintenance is actually much older. Automated methods using cutting-edge signal processing techniques based on pattern recognition have evolved from automated methods using the oldest and most powerful predictive maintenance techniques, visual inspection (Hashemian, 2011).

Predictive maintenance is maintenance strategy in which proactive maintenance is carried out in advance of expected equipment failures and data analysis is utilized to forecast when failures are likely to occur. Numerous sectors, especially those that depend on machinery or equipment to function properly, can benefit greatly from this strategy.

Predictive maintenance is now a data-driven strategy that makes use of analytics and data to foresee equipment failures. This method involves gathering data from equipment or sensors, which is then examined either instantly or on a regular basis. Potential indicators of impending failures can be discovered by looking for patterns and anomalies in the data. Organizations can predict when a component is likely to fail using proactive analysis, enabling them to perform preventive maintenance and avoid expensive disruptions.

Now a days, Predictive maintenance is a common practice in a variety of machinery-dependent industries. It is used to maximize operational effectiveness and increase equipment lifespan. Businesses can proactively spot potential faults or failures before they happen by actively monitoring the performance and condition of machines in real-time. This strategy aids in decreasing maintenance expenses, increasing productivity, and minimizing downtime. As a result, predictive maintenance has been widely embraced by various industries as an effective method for ensuring the durability and consistent operation of machinery.

Advancements in technology, particularly in the field of machine learning, have significantly enhanced the implementation of predictive maintenance in various industries. The availability of powerful computing systems and sophisticated algorithms has enabled organizations to extract meaningful insights from vast amounts of data collected from their equipment.

Machine learning algorithms play a crucial role in predictive maintenance by automatically analyzing and learning from historical data patterns. These algorithms can identify hidden correlations, detect anomalies, and predict potential equipment failures with a high degree of accuracy. As more data is collected and analyzed over time, the algorithms improve their predictive capabilities, leading to even more precise maintenance recommendations.

Furthermore, the integration of predictive maintenance systems with Internet of Things (IoT) devices and sensors has revolutionized the monitoring process(Kanawaday & Sane, 2017). Real-time data collection from connected devices enables continuous monitoring of equipment health and performance. This allows for immediate identification of any deviations from normal operating conditions, triggering alerts and proactive maintenance actions.

The utilization of these advanced technologies and algorithms has not only improved the effectiveness of predictive maintenance but also reduced costs and downtime. By accurately predicting maintenance needs, organizations can schedule repairs or replacements during planned maintenance windows, avoiding unexpected breakdowns and minimizing production disruptions.

Moreover, the benefits of predictive maintenance extend beyond cost savings and increased equipment lifespan. By proactively addressing potential failures, industries can improve safety by preventing accidents that could result from equipment malfunctions. Predictive maintenance also promotes sustainability by optimizing energy consumption and reducing environmental impact.

The high cost of aircraft maintenance is well known in the aviation industry. These expenses are a result of how challenging and tightly controlled aircraft maintenance is, which entails routine inspections, fixes, component replacements, and adherence to predetermined safety standards.

In recent years, researchers have been actively developing and refining machine learning algorithms specifically tailored for the aviation industry, with the goal of reducing maintenance costs. These advancements in algorithmic techniques hold immense promise for revolutionizing aircraft maintenance practices and optimizing resource utilization(Serradilla et al., 2022).

One area of focus is the development of algorithms that can effectively analyze large volumes of data collected from various aircraft systems and sensors. These algorithms are designed to detect patterns, identify correlations, and recognize anomalies that could indicate potential maintenance issues. By efficiently processing and interpreting this data, machine learning algorithms can generate accurate predictions and insights regarding the condition and health of critical components.

Deep learning algorithms are as powerful tools for developing predictive maintenance models compared to traditional machine learning algorithms. they have the ability to handle complex and high-dimensional data, deep learning techniques have revolutionized the field of predictive maintenance, enabling more accurate predictions and proactive maintenance actions.

Predictive maintenance is mostly a multivariate time series problem. Where multivariate time series analysis is a critical technique employed in predictive maintenance. It involves tracking multiple variables simultaneously over a period to identify patterns and trends that may indicate the likelihood of equipment breakdown or failure. Deep learning models, such as CNN and LSTM, excel in handling multivariate time series data due to their ability to capture temporal dependencies and relationships among variables.

2.2. Multivariate Time Series (MTS)

When multiple variable or observations are recorded over a continuous sequence of time interval it is known as Multivariate time series. In multivariate time series, each observation consists of measurements of several different variables at a specific point in time(Wan et al., 2022).

Two frequent tasks in machine learning when applied to multivariate time series are forecasting and classification. Forecasting entails predicting on the future values of one or more time series

variables. In contrast, classification entails determining a category for each time step in the time series.

Numerous machine learning and deep neural network models for MTS classification and forecasting have been developed as a result of deep neural networks' remarkable success across a wide range of applications (Hsieh et al., 2020).

Predictive maintenance uses multivariate time series analysis in many different ways. For instance, patterns in sensor data from machinery, such as temperature, vibration, or pressure readings, can be found. It could be possible to predict when a specific machine is likely to fail by tracking these characteristics over time. This would enable maintenance teams to do preventative maintenance before the failure actually happens.

The detection of patterns in the maintenance data itself is another application of multivariate time series analysis in predictive maintenance. It may be able to predict when specific types of maintenance are likely to be required by examining maintenance data over time and planning maintenance schedules appropriately.

2.3. Machine Learning and Deep Learning Method for Predictive Maintenance

Many research studies have presented a wide range of methodologies utilizing both traditional machine learning algorithms and deep learning techniques for predictive maintenance. Classic machine learning algorithms such as Random Forest and support vector machines which have been explored (Gohel et al., 2020; Paolanti et al., 2018). Additionally, deep learning algorithms like Recurrent Neural Network, Long Short-Term Memory, Gated Recurrent Unit, and Convolutional Neural Network have also been investigated (Dong et al., 2021; Hermawan et al., 2020; Khan et al., 2021; Ning et al., 2021; Serradilla et al., 2022; Xu et al., 2019) for predictive maintenance.

2.3.1. Recurrent Neural Network (RNN)

One popular type of deep learning model for MTS analysis is the recurrent neural network (RNN). Deep learning architectures known as recurrent neural networks (RNNs) are made to handle sequential data. RNNs can capture temporal dependencies in the data, making them particularly well-suited for processing temporal data, such as time series.

The foundation of RNNs is the notion of keeping a "memory" of previous inputs as they process a stream of data. Each time step updates the memory, which is kept in a hidden state. The network can detect temporal dependencies in the data because the hidden state at each time step is a function of the current input and the previous hidden state. Figure 2-1 below describes the general form of Recurrent Neural Networks.

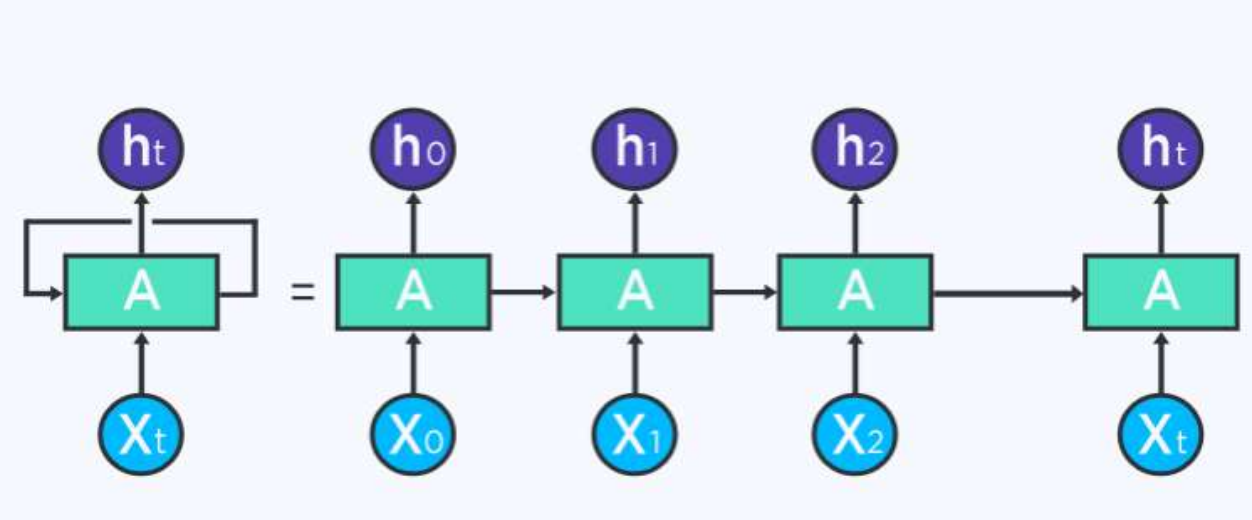

Figure 2-1 Generic Recurrent Neural Network Representation[1]

The Mathematical RNN is can be represented as:

$$h^t = f(h^{t-1}, x^t, \theta)$$

The equation states that the current hidden state **h(t)** is a function **f** of the previous hidden state **h(t-1)** and the current input **x(t)**. The **theta** are the parameters of the function f. The network typically learns to use h(t) as a kind of lossy summary of the task-relevant aspects of the past sequence of inputs up to **t**.

[1] https://builtin.com/data-science/recurrent-neural-networks-and-lstm

RNNs have been successfully applied to a wide range of tasks, such as speech recognition, language translation, image captioning, and music generation. RNNs have also been used for predictive modeling tasks, such as time series forecasting and anomaly detection.

One of the key advantages of RNNs is their ability to handle sequential data of varying lengths. This makes them particularly useful for tasks such as speech recognition, where the length of the audio signal can vary, or language modeling, where the length of the input text can vary.

One of the challenges with RNNs is the vanishing gradient problem, where the gradients become very small as they are propagated back through time. This can make it difficult for the network to learn long-term dependencies in the data. To address this problem, different types of RNNs have been developed, such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs), which use gating mechanisms to selectively update the memory at each time step.

The long short-term memory (LSTM) and gated recurrent unit (GRU) are popular RNN variants that have been successful in MTS analysis. These models use memory cells that can store information over long periods of time, allowing them to capture long-term dependencies in the data. RNN have been used for MTS analysis by many researchers for prediction or classification, example (Hermawan et al., 2020) used LSTM to aircraft maintenance prediction.

2.3.2. Convolutional Neural Networks

Another type of deep learning model that has been used for MTS analysis is the convolutional neural network (CNN), which has traditionally been used for image analysis but can also be applied to MTS data.

Convolutional Neural Networks (CNNs) are a type of deep learning algorithm that have been widely used for image recognition and computer vision tasks. CNNs are designed to automatically learn features from raw input data by employing convolutional filters, which allow the network to extract spatial and temporal patterns from the input signal.

CNNs consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers. Convolutional layers are responsible for extracting local features from the input

signal, while pooling layers reduce the spatial dimensions of the features. Fully connected layers are used to classify the input based on the extracted features.

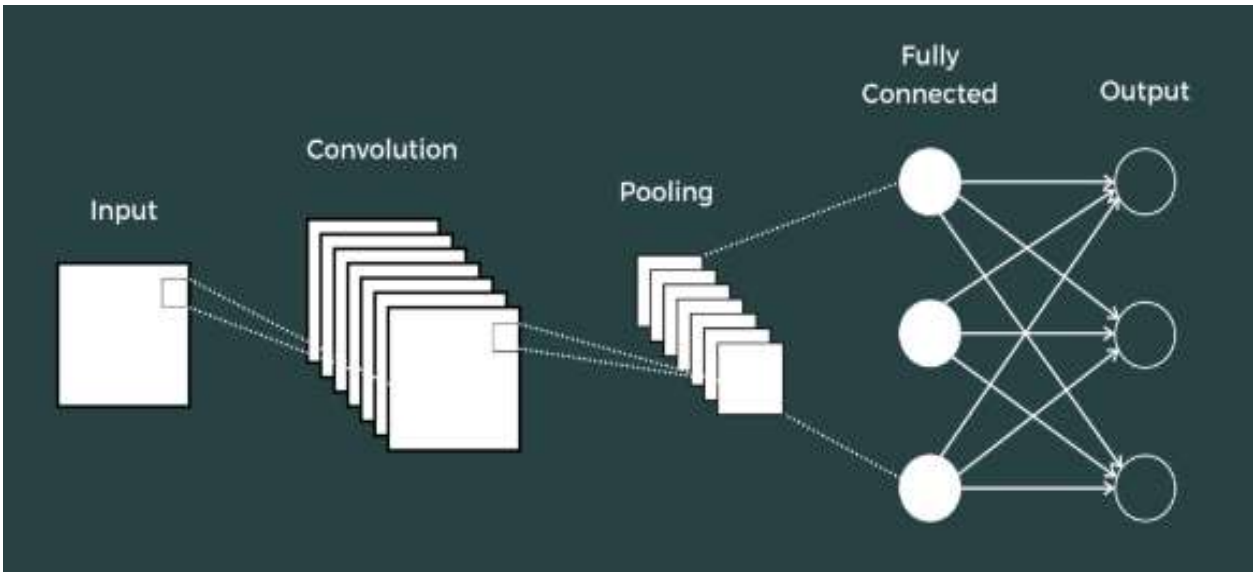

Figure 2-2 Generic Representation of the CNN[2]

The convolution operation involves sliding a small window called a filter or kernel over the input signal and computing the dot product between the filter and each overlapping subregion of the input. The output of the convolution operation is a feature map that highlights regions of the input that are similar to the filter. Multiple filters can be applied to the input, each extracting a different type of feature. After the convolutional layer, pooling layers can be applied to reduce the spatial dimensions of the features while preserving their important information. The most common type of pooling is max pooling, which outputs the maximum value in each region of the feature map. This reduces the size of the feature map while retaining the most salient information.

Finally, fully connected layers can be used to classify the input based on the extracted features. These layers take the flattened output of the previous layers and pass it through a series of fully connected neurons, which can learn to distinguish different classes based on the features.
CNNs can learn hierarchical features from the MTS data and have been shown to be effective in tasks such as forecasting and classification.

[2] https://www.neuronuts.in/tag/dl/

Using a 1D-CNN architecture, which applies convolutional filters across the temporal dimension of the MTS data, is one method of using CNNs to process MTS data. Hence, the CNN can discover regional patterns in the MTS data that are helpful for forecasting future values or locating interesting patterns. A different strategy is to employ a multi-channel 2D-CNN architecture, which simultaneously applies 2D convolutional filters across several variables and time steps. With the MTS data, this can be helpful for capturing intricate, interdependent interactions between many factors. CNN have been used by many researchers for MTS analysis for example (Yang et al., 2022)flight data for predicting aircraft maintenance using CNN with transformers. And another example of using CNNs in multivariate time series analysis is in predicting stock prices. A study by (Mehtab & Sen, 2020) used a CNN to predict the stock price movement based on a combination of technical and fundamental indicators. The CNN was able to outperform traditional machine learning algorithms in predicting the stock price direction.

While CNN and RNN are two significant deep learning methods for evaluating multivariate time series, it is crucial to highlight that there are other alternative methods as well. Yet, because they can effectively capture complex temporal and spatial correlations in the data, CNN and RNN are among the most popular and extensively used approaches for MTS analysis. As a result, while analyzing MTS data, experts in the field of time series analysis frequently use these two approaches as a starting point. Nonetheless, it is better to investigate and experiment with different deep learning approaches to acquire the best results. This includes choosing an approach based on the particular requirements of the problem and the features of the dataset.

2.3.3. Temporal Neural Network

TCN models are convolutional models that can account for the temporal features(Wan et al., 2022). Temporal CNN (TCN) is a type of convolutional neural network architecture that is specifically designed for processing temporal sequences. It has been widely used in the field of time series analysis, including for multivariate time series (MTS) data.

The fundamental principle of TCN is to learn local patterns in temporal sequence data using 1D convolutional filters, and then to capture long-term dependencies throughout the sequence using residual connections. As a result, TCN is able to model intricate temporal correlations in the data and make precise predictions even for lengthy sequences.

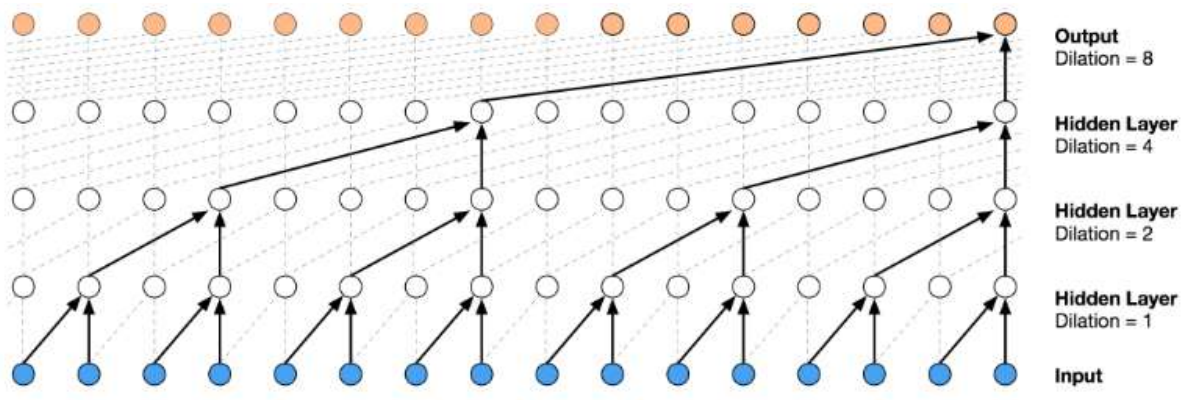

Figure 2-3 Generic Representation of the TCN[3]

Applying TCN to MTS data allows for the simultaneous modeling of numerous variables or features that change over time. This is accomplished by using a number of input channels, each of which stands for a distinct variable in the MTS data. TCN can identify local patterns in each variable as well as the interactions and dependencies among them by applying 1D convolutional filters across each channel.

The fact that TCN does not experience the vanishing gradient issue that RNNs frequently experience allows it to handle longer sequences more effectively than standard RNN-based approaches for MTS analysis.

Convolutional-based models, like TCNs, are thought to be less expensive than RNNs while yet producing comparable results(Gopali et al., 2021). RNNs is found to be hard to train because of the gradient problem while compared CNN based models. RNNs suffer from the problem of vanishing gradients which occurs when the gradient becomes too small, the parameter updates become insignificant this makes the learning of long data sequences difficult.

[3] https://towardsdatascience.com/how-wavenet-works-12e2420ef386

2.4. Related works in Aircraft Maintenance Prediction

When it comes to research studies on aircraft maintenance prediction specifically, there is a limited availability of publicly available research in this area. However, we can draw insights from related studies to gain a better understanding and develop an effective model for aircraft maintenance prediction. Research conducted in areas such as predictive maintenance and fault detection for aircraft and other similar industries can provide valuable inputs for developing an aircraft maintenance prediction model. While there are limited studies which focused solely on aircraft maintenance prediction, leveraging relevant research in related fields can serve as a foundation for building an effective predictive maintenance model for aircraft.

The authors of (Hong, et al., 2022) develop a deep learning architecture using convolutional transformers for general aviation predictive maintenance. The author used C28 and C37 dataset which are data sets of common maintenance issues This data set was compiled from NGAFID, which serves as a repository for general aviation flight data, with a web portal for viewing and tracking flight safety events for individual pilots as well as for fleets of aircraft. The author proposed a model called convolutional Multiheaded self-attention (Conv-MHSA) which achieved a greater classification performance than the other models they compared with such as Convolutional Long Short-Term Memory Networks(C-LSTM). The Conv-MHSA algorithm employed by this author includes 4 stacked MHSA encoder layers with 8 heads each and 64 dense units per head after reducing the temporal resolution from 4069 to 512 through a sequence of 1D convolutions. A dense layer receives the output after it has been pooled and globally averaged for classification. There model was trained on only the last 4096 seconds of flight. There model was not able to classify the type of issues present on a flight which was problematic.

In (Hermawan, et al., 2020) they have proposed a model which can be used for predictive maintenance of aircraft engine. The architecture proposed by this study is the combination of CNN and LSTM algorithms which was proved that it can handle time series data very well. In our proposed architecture, convolutional layer was used for the first layer to process the input data. After that max pooling was used with kernel size of 1. Dropout layer with value 0.1 also used after the max pooling layer to avoid overfitting in the system. Rectified linear unit (ReLU) activation function is used in this paper. Lastly, the dense layer with three of hidden layers is used associated

with the number of classes. This study relies on synthetic data, which may not reflect the noisy nature of data collection.

(Çelikmih, et al., 2020) utilized actual data that was acquired from an aviation firm in Ankara and only included the aircraft's landing gear system. To increase the accuracy of failure count prediction in two stages, (Çelikmih, et al., 2020) tested a few methods and developed a hybrid data preparation approach. They used Relief for feature selection and used K-mean algorithm used for eliminating noisy or inconsistent data. Through the use of LR, SVR, and MLP models, the hybrid data preparation method was put into effect. In comparison to the other model, the LR model performed better.

(Changchang, et al., 2019) in this study a PHM model combining Deep Belief Network (DBN) and LSTM was proposed for condition assessment, fault classification, sensor prediction and RUL estimation of aircraft. The study used NASA C-MAPSS data sets to test the proposed model. They used DBN model with three hidden layers for condition assessment. For Fault classification they used principal component analysis. The LSTM model was applied for predicting sensors. Finally, for estimating RUL based on time series data collected from the LSTM which was used for predicting the sensor is fed into the DBN model for health assessment which is used as threshold to get the estimation of RUL.

In the paper (Maren, et al., 2022) proposed a model for predicting rare failure of aircraft predictive modelling. The model they proposed was based on autoencoder and bidirectional gated recurrent unit network. The autoencoder was trained to detect rare failures and the result of the autoencoder was fed into convolutional bidirectional gated recurrent unit network to predict the next occurrence of failure. The model was evaluated using real-world test dataset. They compared their model with other deep learning model performed well.

Table 2-1Summary of related works

Related Papers	**Architecture**	**Dataset**	**Limitation**
Predictive Maintenance for General Aviation Using Convolutional Transformers(Yang et al., 2022)	CNN with Transformer	C_28 and C_37	➢ Limited number of maintenance issues trained on.
Predictive Maintenance of Aircraft Engine using Deep Learning Technique(Hermawan et al., 2020)	Combination of CNN and LSTM	Engine dataset containing 21 sensors	➢ Data Set used is a dataset generated syntactically.
Failure Prediction of Aircraft Equipment Using Machine Learning with a Hybrid Data Preparation Method(Celikmih et al., 2020)	Relief and K-mean for data preparation and LR for failure detection	Actual data only included the aircraft's landing gear system	➢ Just one part of the aircraft maintenance issues was the model trained on.
Combining multiple deep learning algorithms for prognostic and health management of aircraft.(Che et al., 2019)	LSTM and DBN	NASA's C-MAPSS dataset.	➢ Data Set used is a dataset generated syntactically.
A rare failure detection model for aircraft predictive maintenance using a deep hybrid learning approach.(Dangut et al., 2023)	Auto-encoder and bidirectional gated recurrent unit networks	Data collected from to data bases aircraft central maintenance system and flight deck effects	➢ Unclear whether employing a bidirectional strategy to train imbalanced time-series data would increase model performance

CHAPTER THREE

3. RESEARCH METHDOLOGY

3.1. Chapter Overview

This section outlines the approach employed to conduct the study, including the methods, tools, and techniques utilized to achieve the research objectives. It covers various stages, including data collection, data preprocessing, feature extraction, model training, and evaluation processes.

3.2. Dataset

Data plays a crucial role in training machine learning models, including those used in predictive maintenance. Similarly, in the context of predictive maintenance, acquiring the necessary data becomes one of the primary tasks in building a robust dataset for the study.

For this study we used a flight data which is published by the author of (Hong, et al., 2022) which is dataset collected from the NGAFID database. NGAFID is part of Aviation Safety Information Analysis and Sharing (ASIAS), a Federal Aviation Administration (FAA) funded, joint government-industry, collaborative information sharing program to proactively analyze broad and extensive data sources towards the advancement of safety initiatives and the discovery of vulnerabilities in the National Airspace System (NAS).

Currently the NGAFID database contains more than 900,000 hours of flight which are generated by over 780,00 flights by 12 different types of aircrafts. The dataset which is used by this study is a dataset publicly available by (Yang & Desell, 2022) from the NGAFID. The dataset is available as a dask dataframe containing 31,117 hours of flight data across 28,935 flights, with header csv that describes each flight and the associated maintenance file. The dataset obtained from NGAFID contains flight data of aircraft type Cessna 172 aircraft with 23 sensors recording every second. The flights occurred in a variety of seasons and weather conditions. The Figure and table below shows sample data entries and list the description of the aircraft sensors.

	volt1	volt2	amp1	amp2	FQtyL	FQtyR	E1 FFlow	E1 OilT	E1 OilP	E1 RPM	...	OAT	IAS	VSpd	NormAc	AltMSL	id	split	date_diff	before_after
0	28.09375	28.09375	6.398438	0.399902	20.218750	23.84375	2.539062	74.2500	67.5000	1229.0	...	-2.199219	0.0	13.210938	-0.029999	838.0	0	0	1	0
1	28.09375	28.09375	6.398438	0.399902	20.218750	23.84375	2.539062	74.2500	67.5000	1229.0	...	-2.199219	0.0	13.210938	-0.029999	838.0	0	0	1	0
2	28.09375	28.09375	6.398438	0.500000	20.265625	23.84375	2.470703	74.2500	67.4375	1226.0	...	-2.199219	0.0	11.140438	-0.029999	839.0	0	0	1	0
4	28.09375	28.09375	6.199219	0.399902	20.218750	23.84375	2.388859	74.2500	67.5000	1230.0	...	-2.500000	0.0	-22.515625	-0.020004	841.0	0	0	1	0
5	28.09375	28.09375	6.309781	0.399902	20.156250	23.81250	2.300781	74.3125	67.5000	1227.0	...	-2.500000	0.0	-13.546875	-0.010002	841.5	0	0	1	0

5 rows × 28 columns

Figure 3. 1 Sample Data Set entries

Table 3-1Dataset description of sensors

No	Column Name	Description
1	Volt1	Main electrical system bus voltage
2	Volt2	Essential bus (standby battery)
3	Amp1	Ammeter on the main battery
4	Amp2	Ammeter on the standby battery
5	FQtyL	Fuel quantity left
6	FQtyR	Fuel quantity right
7	E1 FFlow	Engine fuel flow rate
8	E1 OilT	Engine oil temperature
9	E1 OilP	Engine oil pressure
10	E1 RPM	Engine rotations per minute
11	E1 CHT1	1st cylinder head temperature
12	E1 CHT2	2nd cylinder head temperature
13	E1 CHT3	3rd cylinder head temperature
14	E1 CHT4	4th cylinder head temperature
15	E1 EGT1	1st Exhaust gas temperature
16	E1 EGT2	2nd Exhaust gas temperature
17	E1 EGT3	3rd Exhaust gas temperature
18	E1 EGT4	4th Exhaust gas temperature
19	OAT	Outside air temperature
20	IAS	Indicated air speed
21	VSpd	Vertical speed
22	NormAc	Normal Acceleration
23	AltMSL	Altitude miles above sea level

3.2.1. Dataset Preparation

Preparing the dataset for training a predictive maintenance model for aircraft also requires several crucial steps to ensure its quality and suitability for analysis. The initial step involves data cleaning to eliminate irrelevant or redundant information. This process includes removing features or variables that are not relevant to the specific objectives of the predictive maintenance model.

Given that the full dataset for aircraft maintenance can be extensive and resource-intensive, our first task is to perform benchmarking to select a representative subset for training and testing the model. This involves carefully selecting a subset of the data that captures the essential characteristics and patterns of the larger dataset while being computationally feasible for model training and evaluation.

By benchmarking from the full dataset, we can ensure that the selected subset represents the diversity and complexity of the complete dataset. This subset serves as a representative sample that can be used to develop, train, and evaluate the predictive maintenance model effectively.

3.2.2. Data Preprocessing

Once the dataset for aircraft predictive maintenance is prepared, the next crucial step is preprocessing the data. This involves performing various tasks to ensure the dataset is in a suitable format for model training. Two important preprocessing tasks are removing *NaN* (Not-a-Number) values and normalization.

1. ***NaN* Values Removal:**

NaN values refer to missing or undefined data points within the dataset. These missing values can adversely affect the performance of the predictive maintenance model if not handled properly. In the preprocessing stage, *NaN* values are typically identified and removed from the dataset. This can be done by either eliminating the entire data entry containing *NaN* values or imputing the missing values with appropriate techniques such as mean, median, or interpolation.

2. **Data Augmentation:**

Data augmentation is a technique commonly used in machine learning and computer vision to artificially expand a training dataset by applying various transformations or modifications to the existing data. It aims to increase the diversity and size of the training set, thereby improving the generalization and performance of machine learning models.

Data augmentation is particularly useful when the available training data is limited, which is a common challenge in many machine learning tasks. By increasing the effective size of the training set, it helps mitigate the risk of overfitting and improves the model's ability to generalize well to unseen data.

In our aircraft maintenance prediction task, we encounter a challenge where certain classes or categories of maintenance events are underrepresented in the available training data. To address this issue and improve the performance of our predictive model, we can leverage data augmentation techniques such as Cutout, Mixup, and CutMix.

By applying these augmentation methods specifically to the underrepresented classes, we can artificially generate additional samples with variations and perturbations. This augmentation helps to balance the class distribution, providing the model with more diverse examples and reducing the bias towards the majority classes. By introducing these augmented samples during training, we enhance the model's ability to learn and generalize patterns and features associated with the underrepresented maintenance events, ultimately leading to more accurate and reliable predictions in real-world scenarios.

3. **Normalization:**

Normalization is a preprocessing technique used to scale the data within a specific range. It ensures that all features or variables in the dataset have similar scales and magnitudes, preventing any single feature from dominating the model's training process due to its larger values. Normalization also helps the model converge faster and reduces the chances of numerical instability during training. Common normalization methods include min-max scaling, z-score normalization, or decimal scaling.

By removing *NaN* values and normalizing the dataset, we can ensure that the data is clean, complete, and properly scaled for subsequent model training. These preprocessing steps contribute to enhancing the accuracy and reliability of the predictive maintenance model by reducing the impact of missing values and normalizing the feature values to a consistent range.

3.3. Model Development

After preparing and preprocessing the dataset for aircraft maintenance prediction, the next task is to develop the predictive models. In this study, we experimented with four different models:

1. **CNN (Convolutional Neural Network):**

The CNN model is a deep learning architecture commonly used for image analysis. For predictive maintenance, the CNN model can be adapted to handle sequential data by treating it as a 1D signal. It applies convolutional filters to extract relevant features from the input data and uses pooling layers to down sample the information. The CNN model has shown promising results in capturing patterns and dependencies in sequential data.

2. **CNN-MHSA (Convolutional Neural Network with Multi-Head Self-Attention):**

The CNN-MHSA model combines the power of CNNs and Multi-Head Self-Attention (MHSA). MHSA is a mechanism that allows the model to attend to different parts of the input sequence with varying weights, capturing long-range dependencies. By incorporating MHSA into the CNN architecture, the model can enhance its ability to capture complex relationships and dependencies in the aircraft maintenance data.

3. **TCN (Temporal Convolutional Network):**

The TCN model is a type of convolutional neural network specifically designed for processing temporal or sequential data. It utilizes dilated convolutions, which have exponentially increasing receptive fields, allowing the model to capture long-term dependencies in the data. The TCN model has been successful in various time series tasks and has the potential to effectively capture patterns in aircraft maintenance data.

4. **TCN-MHSA (Temporal Convolutional Network with Multi-Head Self-Attention):**
The TCN-MHSA model combines the temporal modeling capabilities of TCN with the attention mechanism of MHSA. This combination enables the model to capture both local temporal patterns through dilated convolutions and long-range dependencies through self-attention. The TCN-MHSA model aims to achieve a better understanding of the complex relationships in the aircraft maintenance data.

By experimenting with these four models, the study aims to evaluate their performance in predicting aircraft maintenance events. Each model has its own unique architecture and mechanisms for capturing patterns and dependencies in the data.

3.4. Model Evaluation

Evaluation is a crucial step in assessing the effectiveness and performance of predictive maintenance models. It allows us to measure the quality of the model and ensure its optimal functioning. In this study, we employed the k-fold cross-validation approach for model evaluation.

K-fold cross-validation is a widely used technique in machine learning, particularly for assessing model performance on limited datasets. It involves dividing the dataset into k equally sized folds or subsets. The model is then trained and evaluated k times, with each fold serving as the validation set once while the remaining folds are used for training. This approach ensures that all data points are used for both training and validation, reducing the risk of biased results.

3.4.1. Evaluation Metrics

By the use of evaluation metrics, the quality of the model is measured, and these metrics are also used to determine whether or not our model is operating correctly and optimally.
Several assessment techniques, such as training accuracy, validation accuracy, training loss, and validation loss, will enable us to provide a qualitative analysis and quantitative metrics to assess the models developed for aircraft maintenance prediction.
The evaluation metrics used for this study would be Area Under the Curve score for Precision-Recall (PR) and Receiver Operating Characteristic (ROC) and also Accuracy would be used to evaluate the model. Log loss would also be used a means to evaluate the performance of the model.

3.4.1.1. Area Under the Curve score for Precision-Recall (PR)

Precision is a common evaluation metric used in machine learning that measures the proportion of correctly predicted positive instances out of the total instances predicted as positive. It provides a measure of the model's ability to avoid false positives.

1. **True Positives (TP):** These are the instances that are actually positive and are correctly predicted as positive by the model.
2. **False Positives (FP):** These are the instances that are actually negative but are incorrectly predicted as positive by the model.

Precision is calculated using the following formula

$$Precision = {TP}/{(TP + FP)}$$

Precision is expressed as a value between 0 and 1, where 1 represents a perfect precision score, indicating that all positive predictions made by the model are correct.

A high precision score suggests that the model has a low rate of false positives, meaning it is reliable in identifying positive instances accurately. On the other hand, a low precision score indicates that the model tends to make more false positive predictions, which can be problematic depending on the application. precision in machine learning provides a measure of how well a model predicts positive instances accurately, by considering the ratio of true positives to the sum of true positives and false positives.

Recall is another commonly used evaluation metric in machine learning that measures the ability of a model to identify all the relevant positive instances. It quantifies the proportion of true positive instances that are correctly identified by the model.

Recall is calculated using the following formula:

$$Recall = {TP}/{(TP + FN)}$$

In the equation above **FN** stands for False Negative. Like precision, recall is also expressed as a value between 0 and 1, where 1 represents a perfect recall score, indicating that all positive instances were correctly identified by the model.

A high recall score suggests that the model has a low rate of false negatives, meaning it is effective at capturing positive instances. It is especially important in applications where the identification of all positive instances is critical, such as medical diagnoses or fraud detection.

On the other hand, a low recall score indicates that the model tends to miss a significant number of positive instances, resulting in false negatives. This can be problematic if the goal is to minimize false negatives and ensure a high level of sensitivity.

The area under the curve (**AUC**) score for **Precision-Recall** is an important metric used in machine learning to evaluate the performance of models. Unlike other metrics such as accuracy, precision, and recall, the AUC-PR score takes into account the trade-off between precision and recall, providing a more comprehensive picture of a model's performance.
The precision-recall curve is a graphical representation of the trade-off between precision and recall for different threshold values of the model's prediction score. The curve plots precision as a function of recall, where recall is the x-axis and precision are the y-axis. Each point on the curve represents a different threshold value. The AUC-PR score summarizes the area under this curve. The AUC-PR score ranges between 0 and 1, where a score of 1 indicates perfect precision and recall.

3.4.1.2. Area Under the ROC curve (AUC-ROC)

The receiver operating characteristic (ROC) curve is another popular tool used in machine learning to evaluate the performance of classification models. The ROC curve plots the true positive rate (TPR) against the false positive rate (FPR) for different threshold values of the model's prediction score. The TPR, also known as sensitivity, measures the proportion of true positives among all actual positives. The FPR measures the proportion of false positives among all actual negatives. The ROC curve shows the trade-off between TPR and FPR for different threshold values. The area under the ROC curve (AUC-ROC) summarizes the overall performance of the model.

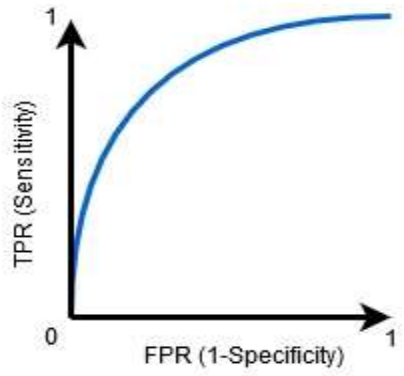

Figure 3-1 Graph AUC-ROC[4]

As show in Figure 3.1 AUC = 1, the classifier can correctly distinguish between all the Positive and the Negative class points. If, however, the AUC had been 0, then the classifier would predict all Negatives as Positives and all Positives as Negatives. The classifier cannot discriminate between Positive and Negative class points when AUC=0.5. i.e., the classifier predicts either a random class or a fixed class for each data point.

Therefore, a classifier's capacity to discriminate between positive and negative classes is improved by a higher AUC value.

3.5. Development Tools

3.5.1. Design Tools

Design concepts are created, presented, and interpreted using design tools. Enterprise Architect is the main design tool that is used to design the system. It is used to create flowcharts, network diagrams, and other design concepts which are professional-looking and it is lightweight, and powerful. Beside Enterprise Architect other online platform are used for the design purpose in this study.

[4] https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning

3.5.2. Hardware Tools

The following Hardware tools would be used in order to implement this research.

Table 3-2 Hardware Tools

No.	Tools	Specification	Used for
1.	GPU	EVGA GeForce GTX	To increase the computation and to expedite the training.
2.	Hard Drive (SSD)	KBG40ZNV512G KIOXIA	Used as storage for large data sets. Faster computational for the training.
3.	RAM	16GB Kingston 3200MHZ	To increase the computation and to expedite the training.

3.5.3. Software Tools

The following software tools would be used to implement this research.

Table 3-3 Software Tools

No	Tools	Description
1	Python	Programming used to implement and demonstrate this thesis requires many of the drivers used to configure and package to install some device with the computer.
2	Tensor Flow	Python deep learning library in the back end.
3	Keras	Will be used to construct deep neural network.
4	Matplotlib	Will be used for data visualization.
5	Anaconda	An application used to install the up-to-date version of python with its different modules and IDEs.
6	Jupyter Notebook	The most popular and handy IDE among AI and deep learning researchers to work with python.
7	Google Colab	Allows to write and execute python in browser, with no configuration required, Access to GPUs free of charge, Easy sharing

CHAPTER FOUR

4. PROPOSED MODEL ARCHITECTURE

4.1. Chapter Overview

The proposed solution architecture for the prediction of aircraft maintenance using deep learning is presented in this chapter, together with mathematical equations and the necessary diagrammatic representation. This chapter is divided into four sections: the first section covers the Problem Formulation, the second part describes the Architecture of the proposed model, the third part describes dataset, the third describes several preprocessing techniques and implementation, and the last portion describes the proposed model implementation.

4.2. Problem Formulation

The problem of predictive maintenance for aircraft can be formulated as a time series analysis task. In this context, we define the task as follows: Let X be a set of sensor measurements collected at a specific time $\boldsymbol{t}$. Based on the sensor data that is currently available, the major objective is to appropriately categorize flights as problematic (requiring maintenance) or in acceptable condition (post maintenance).

Using the flight sensor data, our aim is to determine the probability ($\boldsymbol{P(Yi \mid Xi)}$) that a specific time series was produced by a pre- or post-maintenance flight. where $\boldsymbol{Yi}$ stands for the ith flight's maintenance state (1 for pre-maintenance and 0 for post-maintenance), and $\boldsymbol{Xi}$ stands for the flight sensor data matrix.

4.3. The Proposed Architecture

The primary objective of this study is to develop a predictive maintenance model using deep learning. As stated above for this study we would be experimenting with four deep learning models by using the benchmark subset dataset of NGAFID. The figure below the high-level architecture of the proposed solution.

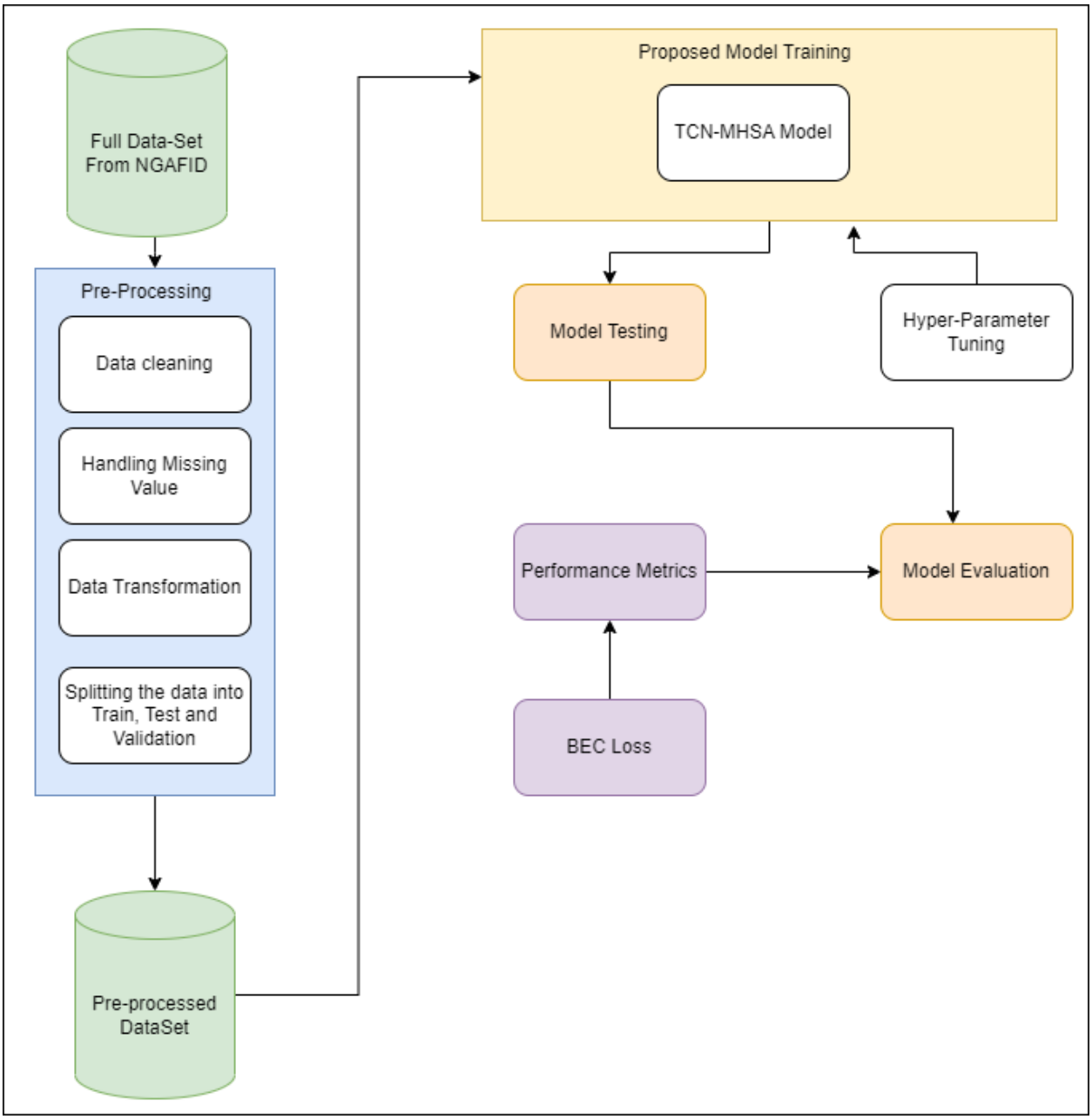

Figure 4-1 Proposed Architecture for Predictive Maintenance

4.4. Dataset Benchmarking

As stated on the above chapter this study used a dataset which is publicly available by the author of(Yang & Desell, 2022). This dataset contains 31,117 hours of flight data across 28,935 flights, with header csv that describes each flight and the associated maintenance file. This data set is clustered into 36 maintenance issues and the flight data contains the reading of 23 sensors which are also divided by the days before and after maintenance. The full dataset has more than 100 million rows of flight data.

Training the full data set would be challenging because of some of the maintenance issues have very few flight data as compared to others which results in class imbalance. To conduct this study, we utilized a smaller portion of the complete dataset. This decision was made due to the difficulty of training a model with 100 million data points, which requires a high-performance computer and significantly longer training time. To address the issue of class imbalance, we carefully selected a benchmark dataset by focusing on the five most frequent maintenance issues that had the largest number of flight data records. This approach aimed to mitigate the class imbalance and ensure a more representative dataset for analysis.

To address the issue of class imbalance and create a more balanced subset of the full dataset, we selected flights that occurred within 3 days before and after maintenance. By doing so, we aimed to reduce the number of data points while also mitigating the class imbalance between the maintenance classes.

The table below provides an overview of the number of records in the subset dataset for each maintenance class, illustrating the distribution of flights within the defined timeframe around maintenance.

Table 4-1 Number of Maintenance Record

Issues	**Labels**	**Number of Records**
Intake Gasket Leak/Damage	C28	25,665,609
Rocker Cover Leak/Loose/Damage	C37	14,181,700
Baffle Crack/Damage/Loose/miss	C29	3,908,999
Intake Tube/bolt/seal/boot lose or damage	C4	3,715,758
Baffle Plug needs Repair/Replace	C6	3,204,365

4.5. Data Preprocessing

The second phase of the proposed model is the preprocessing phase that occurs after finalizing our dataset. The full dataset which are used for benchmarking is provided without any preprocessing. Some flight records contain *NAN* values, so it is necessary to do pre-processing before training our dataset with our proposed model. So, the first task for pre-processing was removing *NAN* value which is about 1% value of the full dataset contains *NAN* value. Then we removed for records which duplicated.

For training our model we have scaled our model using MinMax scaler. We scaled our data by setting the minimum 0 and maximum 1 which is calculated using the whole data of our subset data.

4.6. Proposed Model Architectures

For this study we have selected four model after a careful consideration of various models for predictive maintenance in the aircraft domain. This study would be conducted on the following deep learning models.

1. CNN
2. CNN-MHSA
3. TCN
4. TCN-MHSA

We proposed TCN-MHSA as our preferred choice among the other model which we experimented on for various reasons. We extensively evaluated alternatives such as LSTM, CNN, and the combination of CNN with LSTM, but TCN-MHSA stood out due to its unique advantages in capturing temporal dependencies and handling complex patterns in aircraft sensor data.

Compared to traditional sequential models such as LSTM and GRU, TCN-MHSA offers several key benefits. Firstly, TCN-MHSA leverages the Temporal Convolutional Network (TCN) architecture, which excels at capturing long-term dependencies in time series data. This makes it particularly suitable for predictive maintenance tasks where detecting subtle patterns and anomalies over time is crucial.

4.6.1. TCN-MHSA Model Architecture

Our TCN-MHSA model begins with a temporal block, which serves as the initial component and consists of six dilated convolutional layers and followed by for MHSA encoder layers and which is connected to the global average pooling layer then finally connected to dense layer which is the output layer. the figure below illustrates the architecture of the proposed model.

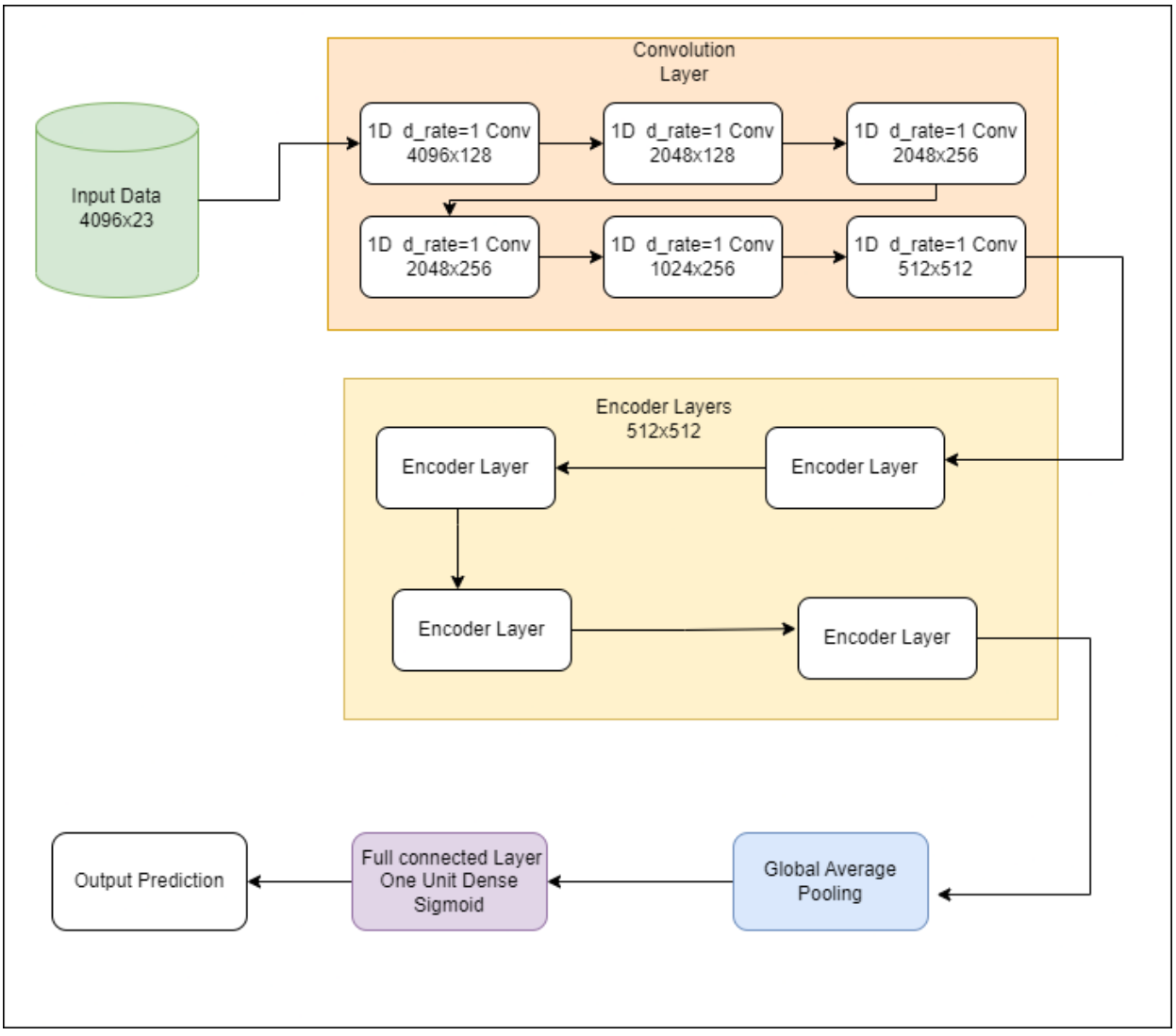

Figure 4-2 TCN-MHSA Architecture

Temporal Block:
The convolutional layer of the proposed model with their dilation rate and filters are listed on Table 4.2.

Table 4-2 Temporal Block of the proposed Model

Layer	Filters	Kernel Size	Padding	Activation	Dilation Rate
1	128	7	Same	ReLU	1
2	128	7	Same	ReLU	2
3	256	3	Same	ReLU	1
4	256	3	Same	ReLU	2
5	512	7	Same	ReLU	1
6	512	7	Same	ReLU	2

Encoder Block:
The encoder layers with their number of heads are listed on Table4.3.

Table 4-3Encoder Layer of the proposed Model

Layer	Input dimension	Number of Attention Heads	Dimensionality of the Feed forward Network
1	512	8	512
2	512	8	512
3	512	8	512
4	512	8	512

4.7. Loss Function of the Proposed Model

4.7.1. Binary Cross Entropy

For our proposed model we used Binary Cross entropy loss function which is a common loss function used in binary classification tasks, where the objective is to predict one of two possible classes. It measures the dissimilarity between the true labels and the predicted probabilities assigned to each class.

In binary classification, the model typically outputs a single scalar value representing the predicted probability of the positive class. The Binary Cross entropy loss function compares this predicted

probability with the true binary labels, which are typically represented as 0s and 1s. The loss is calculated using the formula:

$$BCELoss = -(y_{true} * \log(y_{predicted}) + (y_{true} - 1) * log(y_{predicted} - 1))$$

where $\boldsymbol{y\_true}$ represents the true labels and $\boldsymbol{y\_pred}$ represents the predicted probabilities.

The loss function penalizes incorrect predictions and encourages the model to adjust its parameters to minimize the loss. When the true label approaches to 1, the first term in the equation $(\boldsymbol{y\_true} * \boldsymbol{log}(\boldsymbol{y\_pred}))$ contributes to the loss, pushing the predicted probability towards 1.

Similarly, when the true label approaches to 0, the second term $((\mathbf{1} - \boldsymbol{y\_true}) * \boldsymbol{log}(\mathbf{1} - \boldsymbol{y\_pred}))$ penalizes the predicted probability being close to 1, pushing it towards 0.

The Binary Cross entropy loss function is commonly used in tasks such as binary classification problems, where the goal is to classify instances into one of two classes. Which is the same for our case.

CHAPTER FIVE

5. IMPLEMENTATION OF THE PROPOSED MODEL

5.1. Chapter Overview

In this chapter, the implementation of the proposed model architecture is discussed. The working environment, Temporal Convolutional Network combined with Multi-Head Self-Attention Transformer implementation, and experiments undertaken will be discussed.

5.2. Working Environment

This section describes the environment setup of the hardware components that are used in the implementation of the modeling process with their specifications.

- Laptop: utilized for design the model
  - Operating System: Windows 10
  - Processor: Intel ® Core™ i5-7200U CPU @ 2.50GHz
  - RAM: 8GB
  - System Type: 64-bit operating system, x64-based processor
- Desktop: utilized for pre-processing the data set
  - Operating System : Windows 11
  - Processor: Intel ® Core™ i7 CPU @ 2.50GHz
  - RAM : 32 RM
  - Graphics Card: EVGA GTX 4GB
  - System Type: 64-bit operating system, x64-based processor

5.3. Environment Setup

In order to do our study, we have used particular programming and IDEs.

JupyterLab: - is an open-source web-based interactive development environment for data science, machine learning, and scientific computing. It is a flexible and extensible platform that provides a user-friendly interface for working with Jupyter notebooks, code editors, terminals, and other interactive tools.

Colab: - is a Google cloud-based notebook that allows us to create Python code on any browser with sufficient computing power (TPU, 12GB RAM) and resources to train with the TPU and GPU.

5.3.1. Tools and Packages

This section describes all tools and python packages that are used in the implementation of the modeling process with their specifications. Those libraries are listed, along with their version and description.

Table 5-1 Packages used for the study

Libraries	Version	Description
Numpy	1.24.3	It is a powerful library in Python for numerical computing. It provides support for large, multi-dimensional arrays and matrices, along with a wide range of mathematical functions to operate on these arrays efficiently.
Tensorflow	2.12.0	It is an open-source machine learning framework used for building, training, and deploying machine learning models and neural networks.
Dask	2022.12.1	It is a flexible parallel computing library in Python that enables efficient distributed computing and parallel processing. It provides a high-level interface for working with large datasets and performing complex computations across multiple cores or machines.
Keras	2.12.0	It is a high-level neural networks API written in Python. It is designed for fast and easy experimentation with deep learning models.
Sklearn	1.2.2	Sklearn simplifies the development and deployment of machine learning solutions and is widely used in academia, research, and industry.
Pandas	1.5.3	It provides powerful data structures, such as DataFrame and Series, that allow for efficient handling and manipulation of structured data.
Seaborn	0.12.2	Seaborn simplifies the process of creating plots such as scatter plots, bar plots, box plots, and heatmaps.
Matplotlib	3.7.1	Matplotlib is a widely-used Python library for creating visualizations and plots.

5.4. Training Procedure

In the training procedure the following process has taken place in order to achieve the goal of this research.

- Data Preparation
- Data Preprocessing
- Design proposed solution.
- Training the model

The technique is outlined in detail in each phase to provide further information on what has been done.

5.5. Dataset Description and Loading of Dataset

5.5.1. Preprocessing Implementation

Our preprocessing technique involves working with a large dataset consisting of over 100 million flight and maintenance records. The dataset is represented as a Dask data frame, which is a parallel computing framework for big data processing.

The first challenge we encountered is the presence of high-class imbalance within the dataset. Class imbalance refers to a situation where the distribution of the target variable (or the maintenance issues in this case) is highly skewed, with some classes being significantly more prevalent than others. This can lead to biased model performance and inaccurate predictions. To address this issue, we decided to focus on a subset of maintenance issues for further analysis.

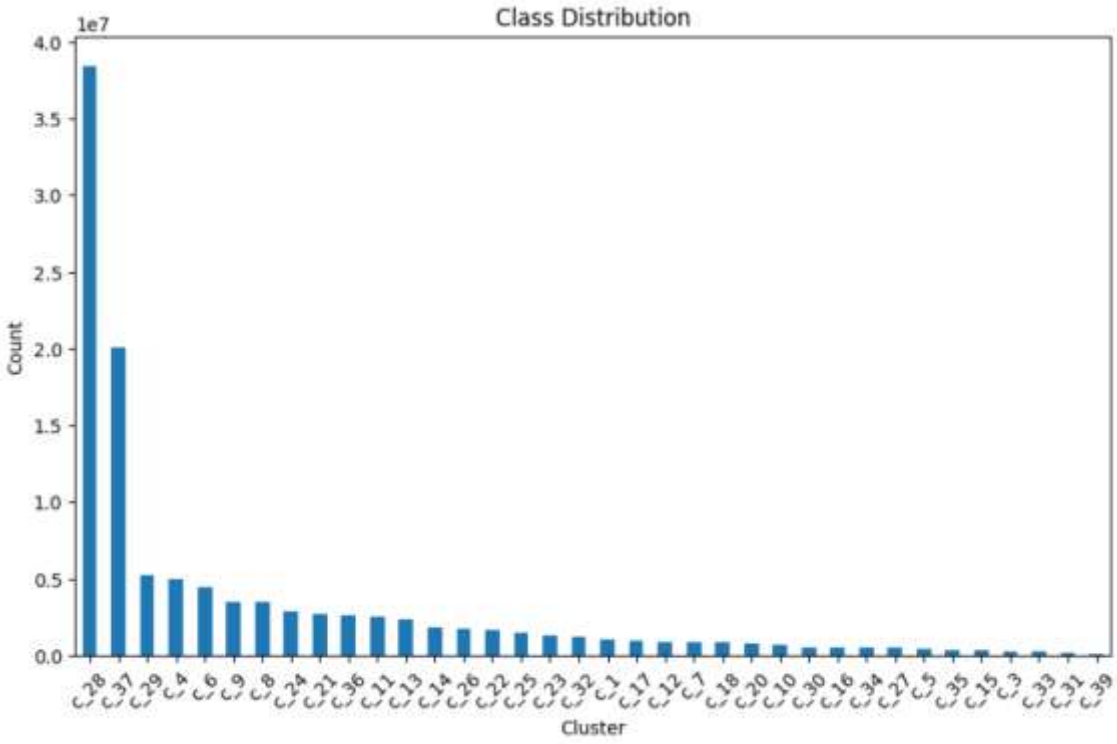

Figure 5-1Full Dataset Statistics Based on Maintenance Issues

Given that most of the maintenance issues have a limited amount of associated flight data, we made the decision to select only five maintenance issues from the entire set. This selection process allows us to concentrate on a manageable number of issues and ensures that each selected issue has a sufficient amount of corresponding flight data for meaningful analysis.

Additionally, the dataset contains more than one percent of missing values (*NaN*) across its features. Missing data can pose challenges during analysis and modeling, as it may lead to biased results or hinder the performance of certain machine learning algorithms. To address this issue, preprocessing techniques such as imputation or removal of missing values was applied to ensure the integrity and reliability of the dataset.

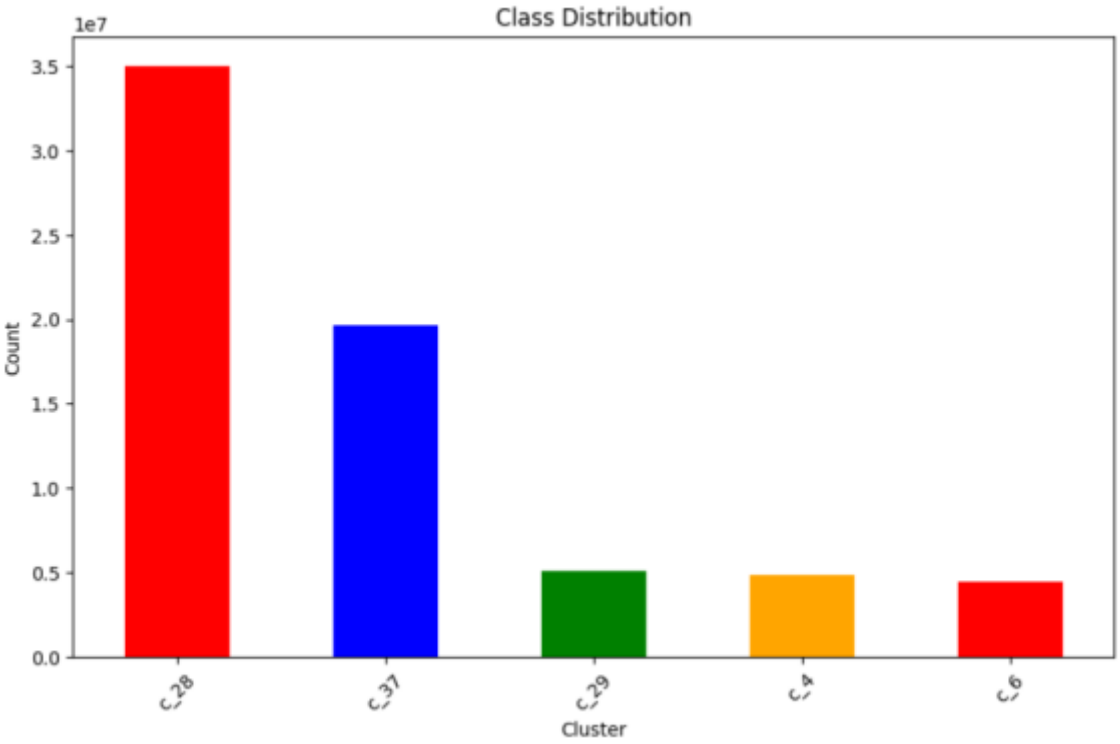

Figure 5-2 Bench mark dataset without preprocessing

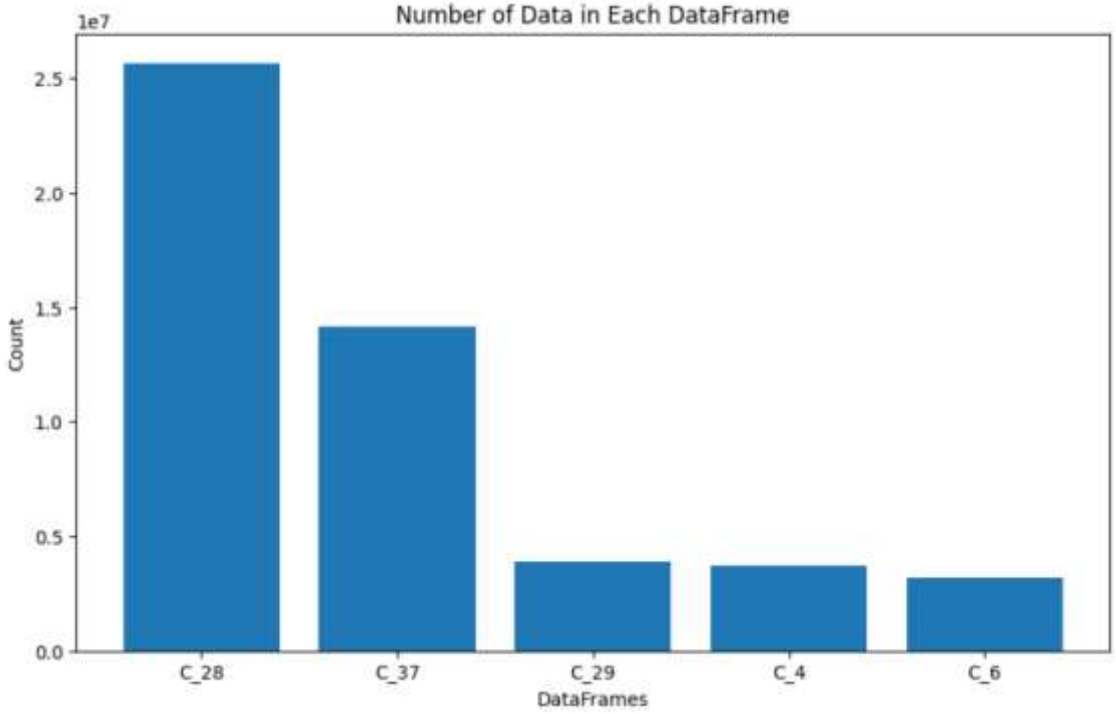

Figure 5-3 Benchmark data set after per-processing

Since our data set have some class imbalance it is necessary to do some data augmentation to make the maintenance class balanced. We use augmentation techniques like Cutout, MixUp and CutMix. Using these techniques we are able to improve our data set and the figure below shows the result after the data augmentation is performed on the data set.

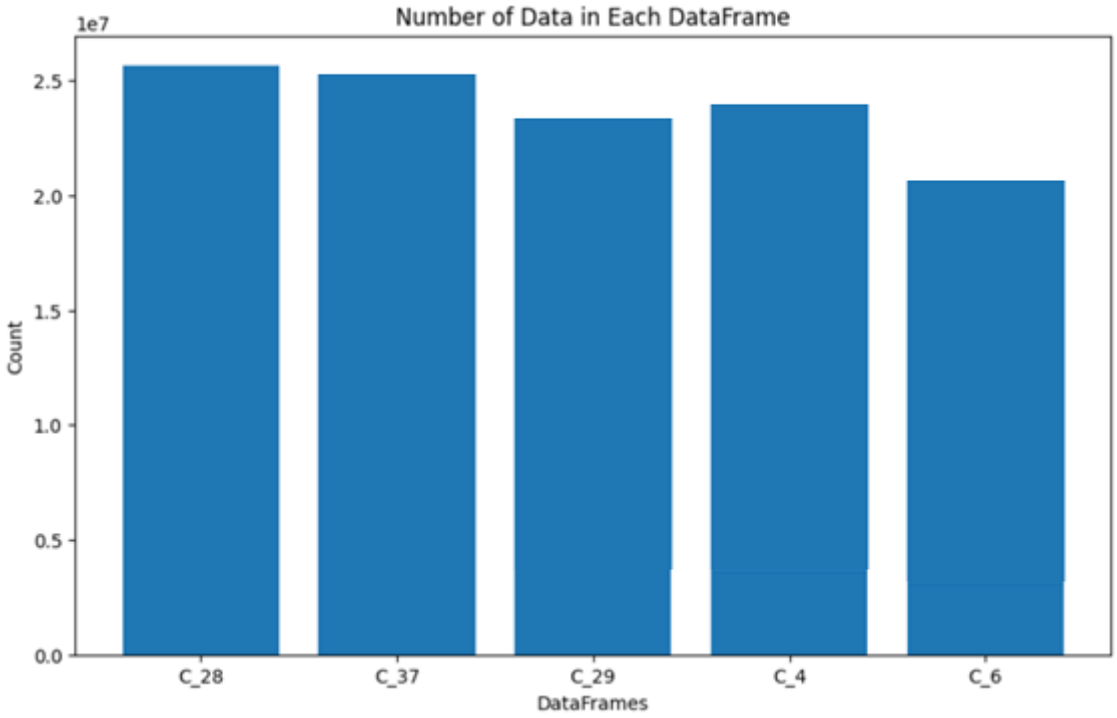

Figure 5-4 Data set after Data Augmentation

Finally, after preprocessing our data, we performed a transformation using the MinMax scaler. The MinMax scaler is a technique that scales the features of the dataset to a specific range, typically between 0 and 1. This normalization process ensures that all features have a consistent scale and helps prevent any particular feature from dominating the analysis due to its larger magnitude.

After applying the MinMax scaler, we exported the transformed data as individual CSV files for each of the five selected maintenance issues. This approach allows us to have separate datasets for each issue, facilitating further analysis and modeling specific to each maintenance problem. By exporting the transformed data as CSV files, we pushed it to our google drive in order to access it while training our model using Colab.

5.6. Implementation of the proposed Model

5.6.1. Loading the Dataset

In order to train our model using (TPU) Tensor Processing Unit we need to load our data set from our google drive to Colab. Our Data Set is divided into 5 individual data set based on the maintenance issues. Fig 5.5 show number of per and post maintenance in each data set for the data without augmentation. Fig 5.6 shows number of per and post maintenance in class after data augmentation.

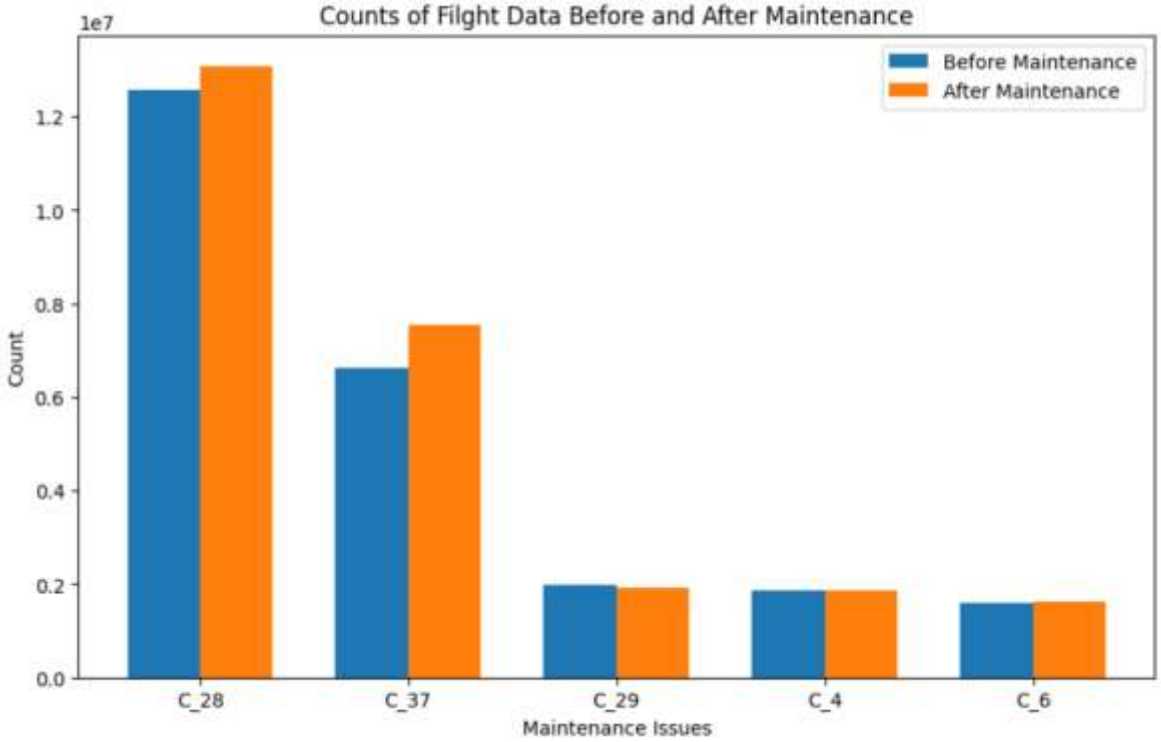

Figure 5-5 Before and after maintenance datapoints in each class without the data augmentation

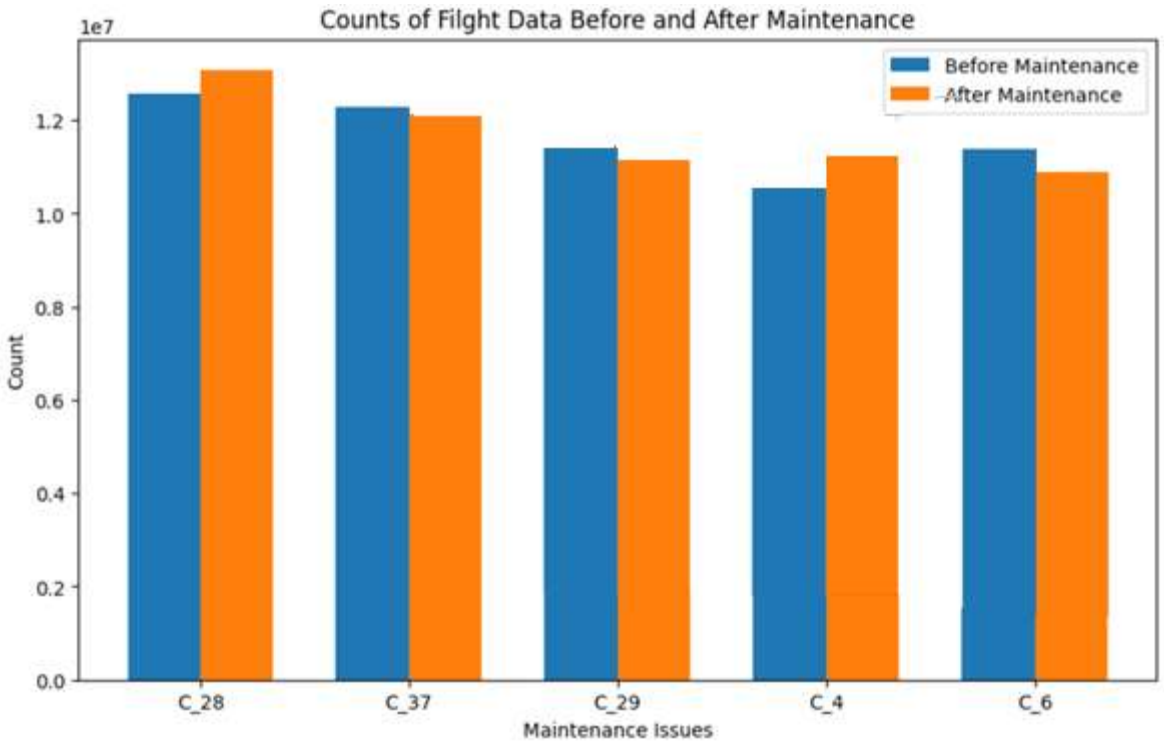

Figure 5-6 Before and after maintenance datapoints in each class after the data augmentation

5.6.2. Hyper-parameters of the model

Hyperparameter tuning is a crucial aspect of machine learning model development, as it directly influences the model's performance and generalization ability. Each hyperparameter plays a specific role in shaping the behavior and effectiveness of the model. The table below discusses the significance of each hyperparameter used for our models and how they may impact the model's training and performance.

Table 5-2 Hyperparameter for Proposed Model

Name	Value	Use
Batch Size	32	This hyperparameter determines the number of samples that are propagated through the model in each training iteration. A batch size of 32 means that the model will process 32 samples at a time before updating its parameters.

EPOCHS	30	This hyperparameter determines the number of times the entire dataset will be passed through the model during training. With 30 epochs, the model will be trained on the entire dataset 30 times.
Number Of Folds	5	This hyperparameter indicates the number of folds used for cross-validation or data splitting. It suggests that the dataset will be split into 5 folds for evaluation or training purposes.
Block Shape	(128,23)	This hyperparameter represents the shape of a block in our model. It is defined as (128, 23), implying that each block has 128 rows and 23 columns
Input Shape	(4096,23)	This hyperparameter represents the shape of a block in our model. It is defined as (128, 23), implying that each block has 128 rows and 23 columns.

5.7. Experiments

This section presents the implementation of the four models. By implementing these four models, we aimed to identify the most effective approach for predictive maintenance in the aircraft industry. The results and analysis obtained from each model's evaluation would help determine the model's performance and its suitability for real-world deployment in predicting maintenance needs and minimizing aircraft downtime.

5.7.1. CNN Model Implementation

The implementation of the CNN model for predictive maintenance involves the use of TensorFlow's deep learning framework. A function called `get_cnn()` which constructs the CNN architecture. It starts with an input layer that takes the specified shape of (4096,23). The model includes multiple convolutional layers with varying filter sizes, kernel sizes, and activation functions, which help capture important patterns and features from the input time series data. The output of the convolutional layers is then fed into a global average pooling layer to reduce the spatial dimensions. Finally, a dense output layer with a sigmoid activation function is added to

produce the binary predictions. The model is compiled with the Adam optimizer, and evaluation metrics such as accuracy, ROC AUC, and PR AUC are specified. Training and validation datasets are obtained for each fold in the k-fold cross-validation process, and the model is trained using the fit function. Throughout the training process, model checkpoints are saved based on the validation loss.

▾ CCN Model

```
def get_cnn():
    # Define input shape
    inputs = tf.keras.Input(shape=SHAPE)
    # Convolutional layers
    x = tf.keras.layers.Conv1D(filters=128, kernel_size=7, padding='same', activation='relu')(inputs)
    x = tf.keras.layers.Conv1D(filters=128, kernel_size=7, padding='same', activation='relu')(x)
    x = tf.keras.layers.Conv1D(filters=256, kernel_size=3, padding='same', activation='relu')(x)
    x = tf.keras.layers.Conv1D(filters=256, kernel_size=3, padding='same', activation='relu')(x)
    x = tf.keras.layers.Conv1D(filters=512, kernel_size=7, padding='same', activation='relu')(x)
    x = tf.keras.layers.Conv1D(filters=512, kernel_size=7, padding='same', activation='relu')(x)
    # Global average pooling layer
    x = tf.keras.layers.GlobalAveragePooling1D()(x)
    # Output layer
    outputs = tf.keras.layers.Dense(units=1, activation='sigmoid')(x)
    # Define model
    model = tf.keras.Model(inputs=inputs, outputs=outputs)
    return model

for i in range(NFOLD):
        if MODEL_TYPE == 'CNN':
            model = get_cnn()
        model.compile(optimizer=tf.optimizers.Adam(learning_rate=tf.keras.optimizers.schedules.CosineDecay(1e-3, decay_steps=EPOCHS * STEPS_PER_EPOCH * 2)),
                      metrics=['accuracy', tf.keras.metrics.AUC(curve='ROC', name='ROC'),tf.keras.metrics.AUC(curve='PR', name='PR')],
                      loss=[tfk.losses.BinaryCrossentropy(reduction=tf.keras.losses.Reduction.SUM_OVER_BATCH_SIZE)] )

        train_dataset, val_dataset = get_train_and_val_for_fold(folded_datasets=folded_datasets, fold=i)

        res = model.fit(train_dataset,epochs=EPOCHS, steps_per_epoch=STEPS_PER_EPOCH,validation_data=val_dataset,
                        callbacks=[tf.keras.callbacks.ModelCheckpoint('model.h5', monitor='val_loss', verbose=0, save_best_only=True, save_weights_only=True)])
```

Figure 5-7Implementation of the CNN Model

5.7.2. TCN Model Implementation

The implementation of the TCN (Temporal Convolutional Network) model for predictive maintenance is similar to the CNN model. We defined a function called `get_tcn()` that constructs the TCN architecture. The input shape of the time series data is specified, and the model includes multiple dilated convolutional layers. Each convolutional layer has a specified filter size, kernel size, activation function, and dilation rate, which helps capture temporal dependencies and patterns at different scales. Similar to our CNN model, a global average pooling layer is applied to reduce the spatial dimensions, followed by a dense output layer with a sigmoid activation function for binary predictions. The model is compiled with the Adam optimizer, and evaluation metrics such as accuracy, ROC AUC, and PR AUC are specified. Training and validation datasets are obtained for each fold in the k-fold cross-validation process, and the model is trained using the fit function. Model checkpoints are saved based on the validation loss during training.

▾ TCN Model

```
def get_tcn():
    # Define input shape
    inputs = tf.keras.Input(shape=SHAPE)
    # Dilated convolution layers
    x = tf.keras.layers.Conv1D(filters=128, kernel_size=7, padding='same', activation='relu', dilation_rate=1)(inputs)
    x = tf.keras.layers.Conv1D(filters=128, kernel_size=7, padding='same', activation='relu', dilation_rate=2)(x)
    x = tf.keras.layers.Conv1D(filters=256, kernel_size=3, padding='same', activation='relu', dilation_rate=1)(x)
    x = tf.keras.layers.Conv1D(filters=256, kernel_size=3, padding='same', activation='relu', dilation_rate=2)(x)
    x = tf.keras.layers.Conv1D(filters=512, kernel_size=7, padding='same', activation='relu', dilation_rate=1)(x)
    x = tf.keras.layers.Conv1D(filters=512, kernel_size=7, padding='same', activation='relu', dilation_rate=2)(x)
    # Global average pooling layer
    x = tf.keras.layers.GlobalAveragePooling1D()(x)
    # Output layer
    outputs = tf.keras.layers.Dense(units=1, activation='sigmoid')(x)
    # Define model
    model = tf.keras.Model(inputs=inputs, outputs=outputs)
    return model

for i in range(NFOLD):
            if MODEL_TYPE == 'TCN':
                model = get_tcn()
            model.compile(optimizer=tf.optimizers.Adam(learning_rate=tf.keras.optimizers.schedules.CosineDecay(2e-5, decay_steps=EPOCHS * STEPS_PER_EPOCH * 2)),
                          metrics=['accuracy', tf.keras.metrics.AUC(curve='ROC', name='ROC'),tf.keras.metrics.AUC(curve='PR', name='PR')],
                          loss=[tfk.losses.BinaryCrossentropy(reduction=tf.keras.losses.Reduction.SUM_OVER_BATCH_SIZE)] )

            train_dataset, val_dataset = get_train_and_val_for_fold(folded_datasets=folded_datasets, fold=i)

            res = model.fit(train_dataset,epochs=EPOCHS, steps_per_epoch=STEPS_PER_EPOCH,validation_data=val_dataset,
                            callbacks=[tf.keras.callbacks.ModelCheckpoint('model.h5', monitor='val_loss', verbose=0, save_best_only=True, save_weights_only=True)])
```

Figure 5-8Implementation of TCN Model

5.7.3. CNN-MHSA Model Implementation

The implements the CNN-MHSA (Convolutional Neural Network with Multi-Head Self-Attention) model for predictive maintenance is also the same as the above models. The ‘get_CNN_MHSA()’ function defines the architecture of the model using a sequential approach. The input shape is specified, and a series of convolutional layers are added to capture local patterns and features in the data. The convolutional layers are followed by multiple EncoderLayers, which implement the Multi-Head Self-Attention mechanism. These layers help the model capture global dependencies and long-range relationships in the time series data. After the EncoderLayers, a global average pooling layer is applied to reduce the spatial dimensions, and a dense output layer with a sigmoid activation function is added for binary classification.

Similar to the previous models, the model is compiled with the Adam optimizer and evaluation metrics such as accuracy, ROC AUC, and PR AUC are specified. The training and validation datasets are obtained for each fold in the cross-validation process, and the model is trained using the fit function. Model checkpoints are saved based on the validation loss during training.

CNN MHSA

```
def get_CNN_MHSA():
    model = tfk.Sequential([tf.keras.Input(shape = SHAPE),
                            tfkl.Conv1D(128, 7, strides = 1, padding='same', activation='relu'),
                            # tfkl.BatchNormalization(),
                            tfkl.Conv1D(128, 7, strides = 2, padding='same', activation='relu'),
                            # tfkl.BatchNormalization(),
                            tfkl.Conv1D(256, 3, strides = 1, padding='same', activation='relu'),
                            # tfkl.BatchNormalization(),
                            tfkl.Conv1D(256, 7, strides = 2, padding='same', activation='relu'),
                            # tfkl.BatchNormalization(),
                            tfkl.Conv1D(512, 7, strides = 2, padding='same', activation='relu'),
                            # tfkl.BatchNormalization(),
                            EncoderLayer(d_model=512, num_heads=8, dff=512),
                            EncoderLayer(d_model=512, num_heads=8, dff=512),
                            EncoderLayer(d_model=512, num_heads=8, dff=512),
                            EncoderLayer(d_model=512, num_heads=8, dff=512),
                            tf.keras.layers.GlobalAveragePooling1D(),
                            tfkl.Dense(1, activation='sigmoid'), ])

    return model

for i in range(NFOLD):
        if MODEL_TYPE == 'CNN-MHSA':
            model = get_CNN_MHSA()
        model.compile(optimizer=tf.optimizers.Adam(learning_rate=tf.keras.optimizers.schedules.CosineDecay(2e-5, decay_steps=EPOCHS * STEPS_PER_EPOCH * 2)),
                      metrics=['accuracy', tf.keras.metrics.AUC(curve='ROC', name='ROC'),tf.keras.metrics.AUC(curve='PR', name='PR')],
                      loss=[tfk.losses.BinaryCrossentropy(reduction=tf.keras.losses.Reduction.SUM_OVER_BATCH_SIZE)] )

        train_dataset, val_dataset = get_train_and_val_for_fold(folded_datasets=folded_datasets, fold=i)

        res = model.fit(train_dataset,epochs=EPOCHS, steps_per_epoch=STEPS_PER_EPOCH,validation_data=val_dataset,
                        callbacks=[tf.keras.callbacks.ModelCheckpoint('model.h5', monitor='val_loss', verbose=0, save_best_only=True, save_weights_only=True)])
```

Figure 5-9 Implementation of CNN-MHSA Model

5.7.4. TCN-MHSA Model Implementation

The implements the TCN-MHSA (Temporal Convolutional Network with Multi-Head Self-Attention) model for predictive maintenance is also the same as the above models. The 'get_TCN_MHSA()' function defines the architecture of the model using a sequential approach. The input shape is specified, and a series of convolutional layers with different dilation rate are added to capture local patterns and features in the data. The convolutional layers are followed by multiple EncoderLayers, which implement the Multi-Head Self-Attention mechanism. These layers help the model capture global dependencies and long-range relationships in the time series data. After the EncoderLayers, a global average pooling layer is applied to reduce the spatial dimensions, and a dense output layer with a sigmoid activation function is added for binary classification.

Similar to the previous models, the model is compiled with the Adam optimizer and evaluation metrics such as accuracy, ROC AUC, and PR AUC are specified. The training and validation datasets are obtained for each fold in the cross-validation process, and the model is trained using the fit function. Model checkpoints are saved based on the validation loss during training.

```
def get_TCN_MHSA():
    # Define input shape
    inputs = tf.keras.Input(shape=SHAPE)
    # Dilated convolution layers
    x = tf.keras.layers.Conv1D(filters=128, kernel_size=7, padding='same', activation='relu', dilation_rate=1)(inputs)
    x = tf.keras.layers.Conv1D(filters=128, kernel_size=7, padding='same', activation='relu', dilation_rate=1)(x)
    x = tf.keras.layers.Conv1D(filters=256, kernel_size=3, padding='same', activation='relu', dilation_rate=1)(x)
    x = tf.keras.layers.Conv1D(filters=256, kernel_size=3, padding='same', activation='relu', dilation_rate=1)(x)
    x = tf.keras.layers.Conv1D(filters=512, kernel_size=7, padding='same', activation='relu', dilation_rate=1)(x)
    x = tf.keras.layers.Conv1D(filters=512, kernel_size=7, padding='same', activation='relu', dilation_rate=1)(x)
    x= EncoderLayer(d_model=512, num_heads=8, dff=512)(x)
    x= EncoderLayer(d_model=512, num_heads=8, dff=512)(x)
    x= EncoderLayer(d_model=512, num_heads=8, dff=512)(x)
    x= EncoderLayer(d_model=512, num_heads=8, dff=512)(x)
    # Global average pooling layer
    x = tf.keras.layers.GlobalAveragePooling1D()(x)
    # Output layer
    outputs = tf.keras.layers.Dense(units=1, activation='sigmoid')(x)
    # Define model
    model = tf.keras.Model(inputs=inputs, outputs=outputs)
    return model
for i in range(NFOLD):
        if MODEL_TYPE == 'TCN-MHSA':
            model = get_TCN_MHSA()
        model.compile(optimizer=tf.optimizers.Adam(learning_rate=tf.keras.optimizers.schedules.CosineDecay(1e-5, decay_steps=EPOCHS * STEPS_PER_EPOCH * 2)),
                      metrics=['accuracy', tf.keras.metrics.AUC(curve='ROC', name='ROC'),tf.keras.metrics.AUC(curve='PR', name='PR')],
                      loss=[tfk.losses.BinaryCrossentropy(reduction=tf.keras.losses.Reduction.SUM_OVER_BATCH_SIZE)] )

        train_dataset, val_dataset = get_train_and_val_for_fold(folded_datasets=folded_datasets, fold=i)

        res = model.fit(train_dataset,epochs=EPOCHS, steps_per_epoch=STEPS_PER_EPOCH,validation_data=val_dataset,
                        callbacks=[tf.keras.callbacks.ModelCheckpoint('model.h5', monitor='val_loss', verbose=0, save_best_only=True, save_weights_only=True)])
```

Figure 5-10 Implementation of TCNN-MHSA Model

5.7.5. Encoder Layer Implementation

The encoder layer is the layer which have several components like the Multi-Head Self-Attention mechanism for our models for predictive maintenance.

In the Implementation we defined 2 class the Encoder class and the Multi-head Self-Ation.The ‘point_wise_feed_forward_network’ function defines a feed-forward network with a ReLU activation function. It consists of two dense layers, with the first layer having a hidden dimension and the second layer outputting a tensor of shape.

The ‘scaled_dot_product_attention’ function calculates the scaled dot product attention given query (q), key (k), and value (v) tensors. It performs matrix multiplication between q and k, scales the result, and applies a softmax activation. It then computes the weighted sum of the values (v) based on the attention weights.

The ‘MultiHeadAttention’ class implements the multi-head attention mechanism. It takes the input tensors (v, k, q) and applies linear transformations using the `wq`, `wk`, and `wv` dense layers. The input tensors are split into multiple heads, and the scaled dot product attention is applied to each head. The attention outputs are concatenated, transformed using the `dense` layer, and returned along with the attention weights.

The ‘EncoderLayer’ class represents a single layer in the encoder of the model. It consists of a multi-head attention sub-layer and a feed-forward network sub-layer. Layer normalization and dropout are applied to the outputs of each sub-layer, and residual connections are added.

```
class EncoderLayer(tf.keras.layers.Layer):
  def __init__(self, d_model, num_heads, dff, rate=0.1):
    super(EncoderLayer, self).__init__()
    self.mha = MultiHeadAttention(d_model, num_heads)
    self.ffn = point_wise_feed_forward_network(d_model, dff)
    self.layernorm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
    self.layernorm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
    self.dropout1 = tf.keras.layers.Dropout(rate)
    self.dropout2 = tf.keras.layers.Dropout(rate)

  def call(self, x, training, mask = None):
    attn_output, self.attention_values = self.mha(x, x, x, mask)
    attn_output = self.dropout1(attn_output, training=training)
    out1 = self.layernorm1(x + attn_output)
    ffn_output = self.ffn(out1)
    ffn_output = self.dropout2(ffn_output, training=training)
    out2 = self.layernorm2(out1 + ffn_output)
    return out2
```

Figure 5-11 Implementation of Encoder-Layer

```
class MultiHeadAttention(tf.keras.layers.Layer):
  def __init__(self, d_model, num_heads):
    super(MultiHeadAttention, self).__init__()
    self.num_heads = num_heads
    self.d_model = d_model
    assert d_model % self.num_heads == 0
    self.depth = d_model // self.num_heads
    self.wq = tf.keras.layers.Dense(d_model)
    self.wk = tf.keras.layers.Dense(d_model)
    self.wv = tf.keras.layers.Dense(d_model)
    self.dense = tf.keras.layers.Dense(d_model)

  def split_heads(self, x, batch_size):
    x = tf.reshape(x, (batch_size, -1, self.num_heads, self.depth))
    return tf.transpose(x, perm=[0, 2, 1, 3])

  def call(self, v, k, q, mask):
    batch_size = tf.shape(q)[0]
    q = self.wq(q)
    k = self.wk(k)
    v = self.wv(v)
    q = self.split_heads(q, batch_size)
    k = self.split_heads(k, batch_size)
    v = self.split_heads(v, batch_size)
    scaled_attention, attention_weights = scaled_dot_product_attention( q, k, v, mask)
    scaled_attention = tf.transpose(scaled_attention, perm=[0, 2, 1, 3])
    concat_attention = tf.reshape(scaled_attention,(batch_size, -1, self.d_model))
    output = self.dense(concat_attention)
    return output, attention_weights
```

Figure 5-12 Implementation of MHSA Layer

CHAPTER SIX

6. RESULTS AND DISCUSSIONS

6.1. Chapter Overview

This chapter covers the results obtained from the proposed solution and provides a comparative description of the results with graphs and tables, having previously discussed the concepts for implementation and the details of this work.

6.2. Result Description

In our research, we utilized our preprocessed dataset to assess how well our proposed models performed. We measured the effectiveness of the models by calculating the area under the curve of PR, area under the curve of ROC and accuracy scores. We developed four distinct deep learning models for our experiments and assessed their performance using standard metrics. To gauge the performance of each model in a consistent manner, we conducted multiple time series experiments using 5-fold cross-validation, determining the average area under curve of ROC, and average area under curve of PR values, average accuracy.

6.3. Results for the Deep Learning Models

6.3.1. Results for the CNN Model and CNN-MHSA

In the first experiment of our study, we employed the CNN model, a deep learning architecture commonly utilized for predictive maintenance tasks. To train our model, we adopted a 5-fold training approach, which involved dividing the dataset into five subsets or folds. Each fold was trained separately, while the remaining folds were used for validation and testing purposes.
During training, we conducted 30 epochs for each fold, with each epoch consisting of 250 training steps.
In the next experiment of our study, we explored the CNN-MHSA model, a model that combines both Convolutional Neural Network (CNN) and Multi-Head Self-Attention (MHSA) mechanisms. This hybrid architecture aims to leverage the strengths of both CNNs and self-attention mechanisms to improve predictive maintenance performance.

Our results indicated that the CNN-MHSA model outperformed the CNN model in terms of predictive accuracy and overall performance. The integration of self-attention mechanisms within

the CNN architecture allowed the model to capture long-range dependencies and establish stronger contextual relationships between different elements in the input data.

By incorporating self-attention mechanisms, the CNN-MHSA model gained the ability to focus on relevant features or patterns within the data, adaptively attending to important temporal and spatial information. This enhanced attention mechanism contributed to improved performance in capturing complex patterns and making accurate predictions in the context of predictive maintenance.

The superior performance of the CNN-MHSA model compared to the CNN model as shown on the table 6.1 blow suggests that the fusion of CNNs and self-attention mechanisms can effectively enhance the model's ability to identify and exploit important information for predictive maintenance tasks. The combination of these two architectures showcases the potential of leveraging multiple deep learning techniques to achieve more accurate and reliable maintenance predictions.

Table 6-1Results of CNN and CNN-MHSA for each maintenance Class

Maintenance Class	Models							
	CNN				CNN-MHSA			
	PR	ROC	Accuracy	BCE Loss	PR	ROC	Accuracy	BCE Loss
C28	0.649	0.705	0.651	0.628	0.813	0.835	0.760	0.526
C37	0.583	0.686	0.665	0.636	0.698	0.766	0.716	0.622
C29	0.637	0.702	0.645	0.631	0.807	0.826	0.758	0.531
C6	0.632	0.702	0.662	0.634	0.804	0.817	0.753	0.534
C4	0.628	0.693	0.656	0.634	0.796	0.802	0.743	0.538

Figure 6.1- 6.6 depicts the plotted graphs illustrating the performance of the CNN and CNN-MHSA models across the three maintenance classes. The curves represent the AUC ROC, AUC PR, and accuracy metrics, providing a comprehensive view of the models' predictive capabilities

for each maintenance class. The graph provides valuable insights into the comparative performance of the CNN and CNN-MHSA models, showcasing their proficiency in predicting maintenance needs across the different classes.

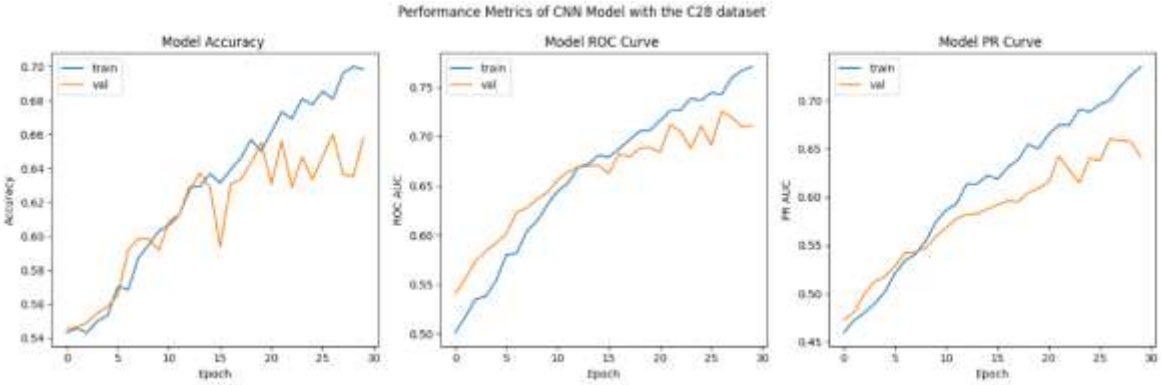

Figure 6-1Performance of CNN Model over the C28 maintenance Class

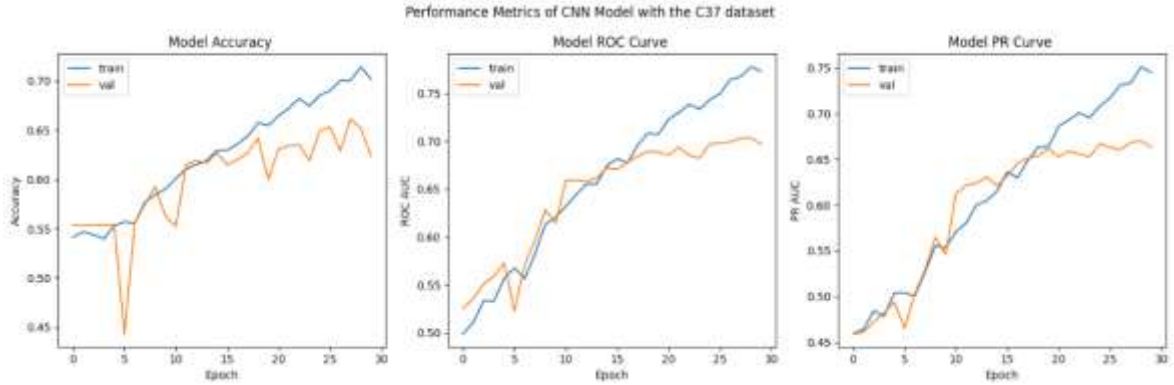

Figure 6-2Performance of CNN Model over the C37 maintenance Class

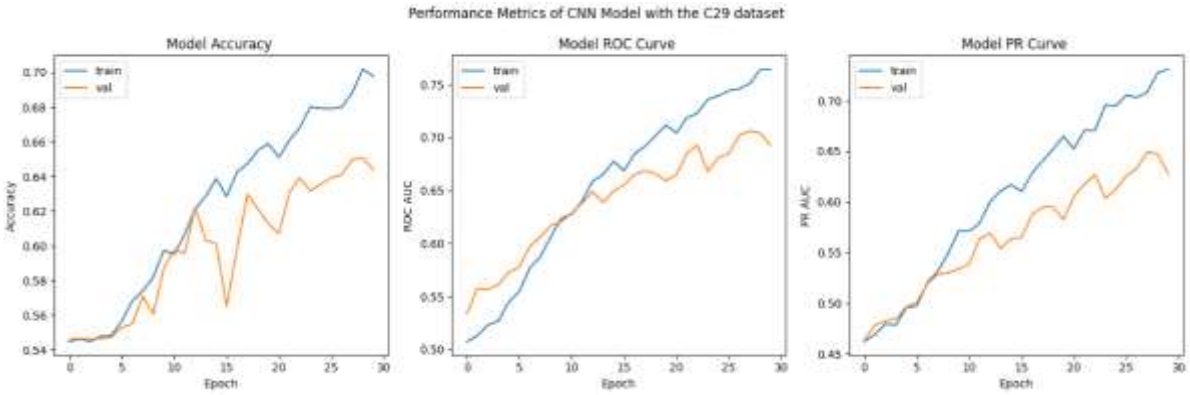

Figure 6-3Performance of CNN Model over the C29 maintenance Class

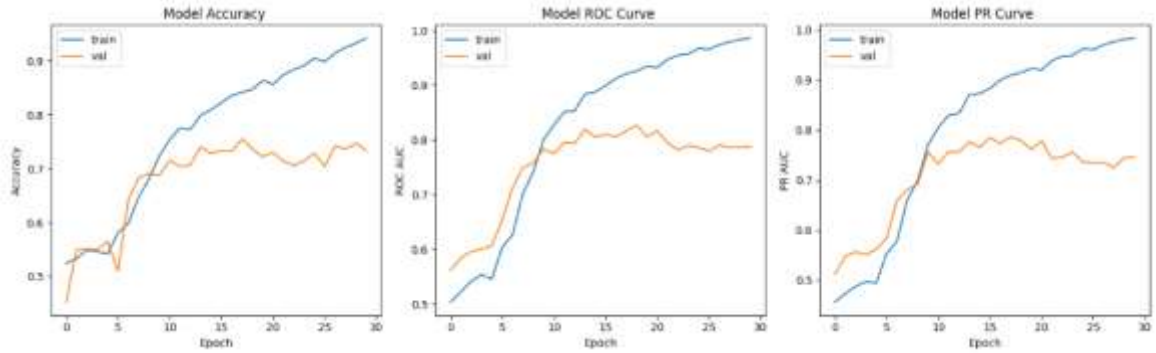

Figure 6-4 Performance of CNN-MHSA Model over the C28 maintenance Class

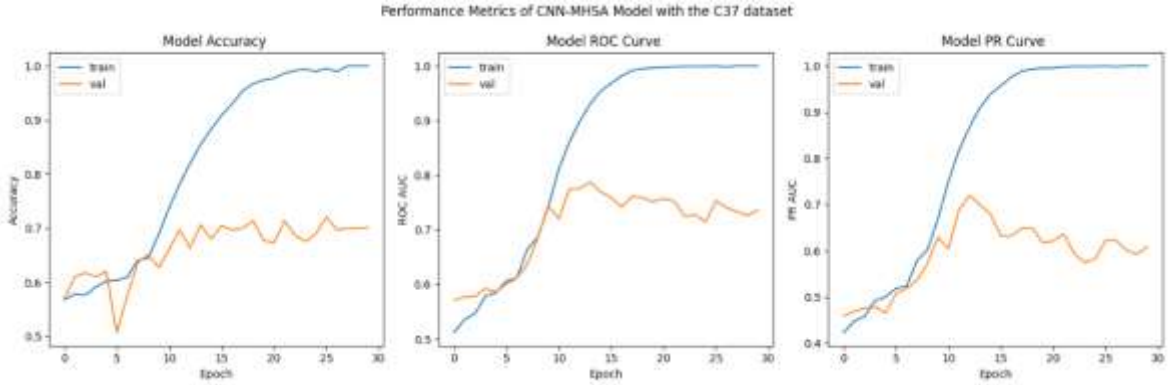

Figure 6-5 Performance of CNN-MHSA Model over the C28 maintenance Class

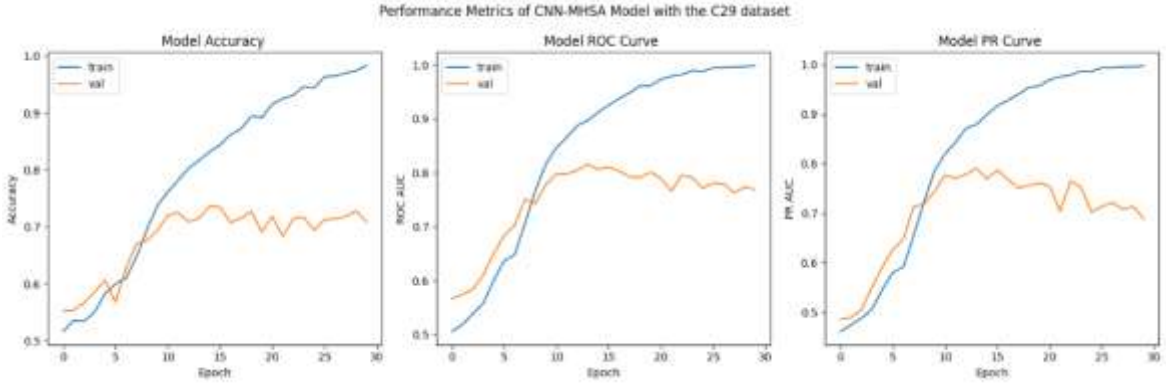

Figure 6-6 Performance of CNN-MHSA Model over the C28 maintenance Class

6.3.2. Results for the TCN model and TCN-MHSA

Our next experiment of our study was done on the Temporal Convolutional Network (TCN) model for predictive maintenance. TCN is a deep learning architecture specifically designed to model temporal dependencies in sequential data.

TCN models employ a series of dilated convolutions, which allow for an expanded receptive field and capture long-range dependencies across the temporal dimension. This architecture enables TCN models to effectively analyze time series data and capture complex temporal patterns, making them suitable for predictive maintenance tasks.

During the experiment, we trained and evaluated the TCN model using our dataset. The TCN model's performance was assessed based on various evaluation metrics, such as AUC-ROC, AUC-PR and Accuracy scores. These metrics allowed us to gauge the model's predictive capabilities and overall performance in comparison to the previous CNN and CNN-MHSA models.

The inclusion of the TCN model in our study demonstrates our exploration of diverse deep learning architectures for predictive maintenance. By specifically focusing on temporal dependencies within the data, the TCN model aimed to capture intricate patterns and improve predictive accuracy for maintenance needs.

Our proposed model, which combines the Temporal Convolutional Network (TCN) with Multi-Head Self-Attention (MHSA) mechanisms. The results demonstrated that the TCN-MHSA model outperformed all four previously tested models in terms of predictive performance.

By integrating the TCN architecture with the MHSA mechanism, the TCN-MHSA model benefitted from the temporal modeling capabilities of TCN and the enhanced attention mechanisms of MHSA. This fusion allowed the model to effectively capture temporal dependencies and extract salient features from the input data, leading to improved predictive performance.

The TCN-MHSA model demonstrated superior performance compared to the CNN, CNN-MHSA, TCN, and TCN-MHSA models, indicating its ability to capture complex patterns and extract valuable information from the data for accurate maintenance predictions. This suggests that leveraging both temporal convolutions and self-attention mechanisms can enhance the model's ability to capture long-range dependencies and focus on relevant features, ultimately leading to better predictive maintenance outcomes.

The results of these experiments are show in the table.

Table 6-2 Results of TCN and TCN-MHSA for each maintenance Class

Maintenance Class	Models							
	TCN				TCN-MHSA			
	PR	ROC	Accuracy	BCE Loss	PR	ROC	Accuracy	BCE Loss
C28	0.662	0.713	0.660	0.621	0.823	0.846	0.769	0.517
C37	0.674	0.722	0.664	0.617	0.817	0.838	0.765	0.516
C29	0.656	0.707	0.655	0.624	0.819	0.841	0.762	0.517
C6	0.663	0.711	0.659	0.628	0.821	0.83.7	0.760	0.509
C4	0.654	0.705	0.653	0.624	0.818	0.828	0.753	0.513

Figure 6.7- 6.10 depicts the plotted graphs illustrating the performance of the CNN and CNN-MHSA models across the five maintenance classes. The curves represent the AUC ROC, AUC PR, and accuracy metrics, providing a comprehensive view of the models' predictive capabilities for two of maintenance class. The graph provides valuable insights into the comparative performance of the TCN and TCN-MHSA models, showcasing their proficiency in predicting maintenance needs across the different classes.

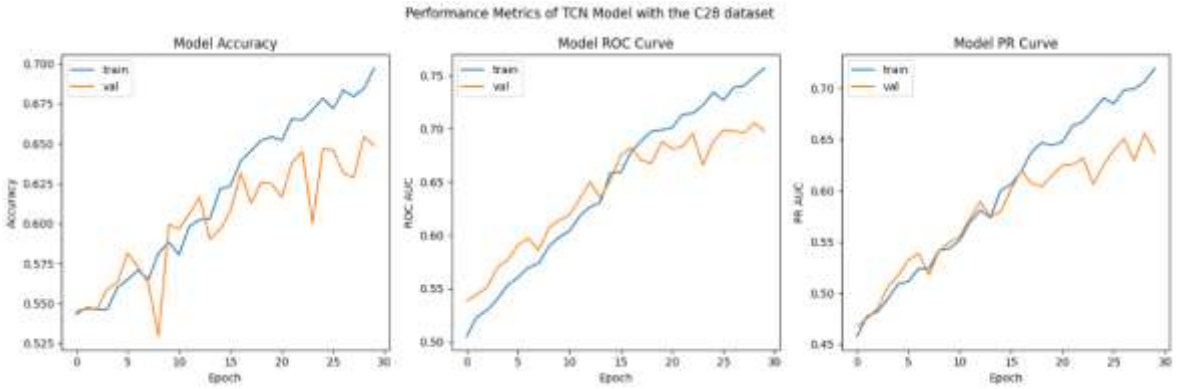

Figure 6-7Performance of TCN Model over the C28 maintenance Class

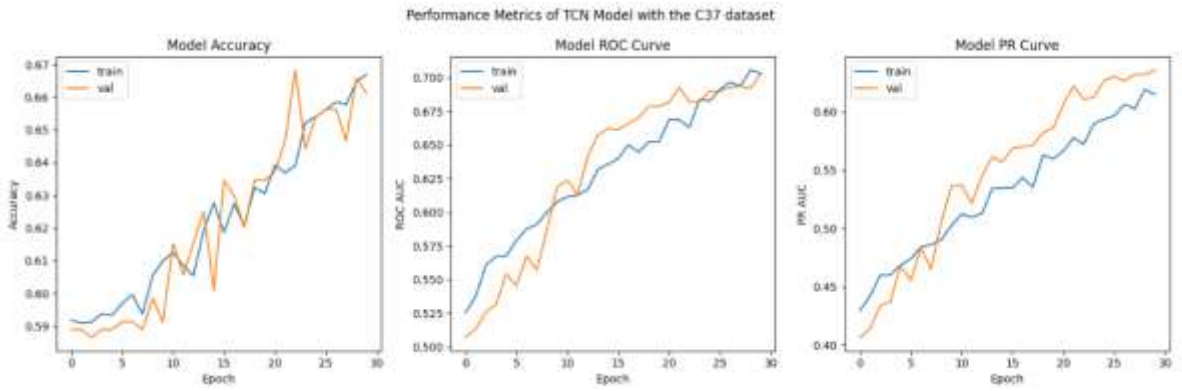

Figure 6-8 Performance of TCN Model over the C37 maintenance Class

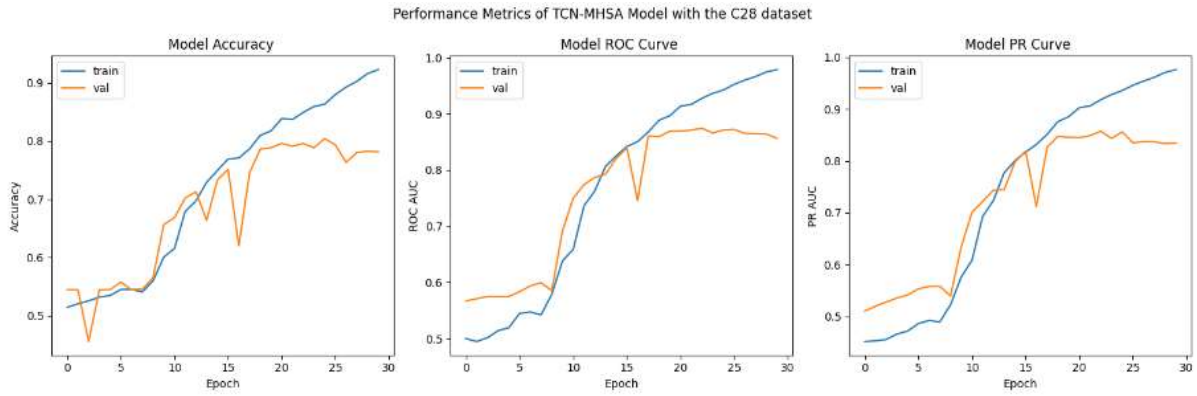

Figure 6-9 Performance of TCN-MHSA Model over the C28 maintenance Class

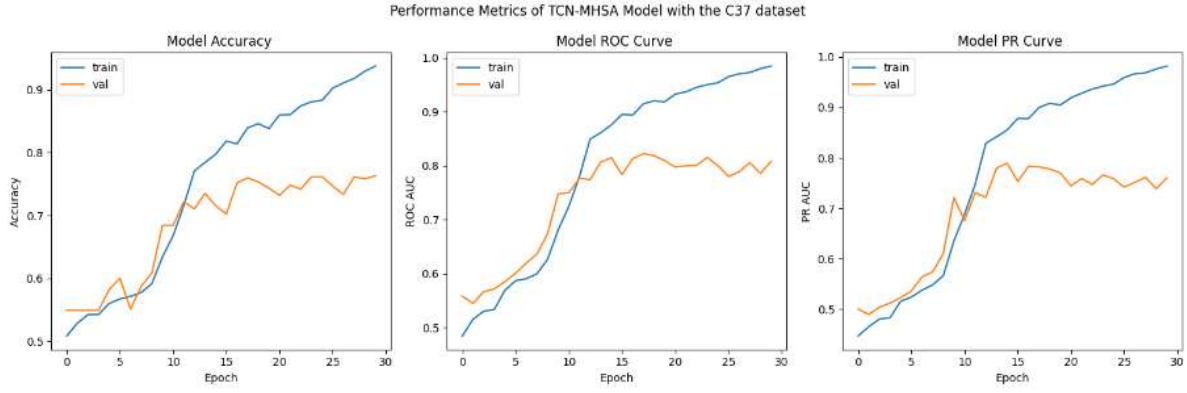

Figure 6-10 Performance of TCN-MHSA Model over the C37 maintenance Class

6.4. Hyperparameter Tuning Result

In order to optimize the performance of our models (CNN, CNN-MHSA, TCN, and TCN-MHSA), we conducted extensive hyperparameter tuning experiments. The primary focus of these experiments was to investigate the impact of varying the number of training epochs and the number of attention heads in the MHSA module. By varying these hyperparameters, we aimed to identify the optimal configurations that would enhance the models' performance.

For the hyperparameter tuning experiments, we trained each model for two different numbers of epochs: 30 and 40. This allowed us to evaluate their performance over different training durations and determine the effect of the training. Additionally, we evaluated the models based on multiple performance metrics, including AUC ROC, AUC PR, BCE Loss, and Accuracy.

The results of the hyperparameter tuning experiments on the maintenance issue C28 are summarized in Table 6.3. We observed that the performance of the models varied based on the number of epochs. For instance, in the CNN model, increasing the number of epochs from 30 to 40 resulted in a slight decrease in AUC ROC and AUC PR, while BCE Loss and Accuracy also experienced slight fluctuations.

Table 6-3 Hyperparameter (Epochs)Tuning Results

Model	Epochs	AUC ROC	AUC PR	BCE Loss	Accuracy
CNN	30	0.705	0.649	0.628	0.651
CNN	40	0.700	0.632	0.672	0.637
CNN-MHSA	30	0.835	0.813	0.526	0.760
CNN-MHSA	40	0.826	0.800	0.631	0.742
TCN	30	0.713	0.662	0.621	0.660
TCN	40	0.705	0.621	0.684	0.655
TCN-MHSA	30	0.846	0.823	0.517	0.769
TCN-MHSA	40	0.828	0.820	0.525	0.764

And also, we examined the influence of the number of attention heads in the MHSA module for the CNN-MHSA and TCN-MHSA models. By training these models with both 4 and 8 heads, we aimed to explore the impact of increasing the complexity of the attention mechanism. The results of this investigation are presented in Table 6.4.

Table 6-4 Hyperparameter (Number of Attention heads) Tuning Results

Model	Heads	Epochs	AUC ROC	AUC PR	BCE Loss	Accuracy
CNN-MHSA	4	30	0.710	0.677	0.641	0.640
CNN-MHSA	8	30	0.835	0.813	0.526	0.760
TCN-MHSA	4	30	0.722	0.691	0.627	0.661
TCN-MHSA	8	30	0.846	0.823	0.517	0.769

From the results, we observed that the CNN-MHSA model achieved comparable performance with both 4 and 8 attention heads, indicating that the added complexity of more attention heads did not significantly improve the model's performance. On the other hand, the TCN-MHSA model displayed slightly better performance with 8 attention heads compared to 4, suggesting that the increased complexity of the attention mechanism had a positive impact on its performance.
Overall, these hyperparameter tuning experiments helped us identify the optimal configurations for each model. By selecting the appropriate number of training epochs and attention heads, we were able to enhance the models' performance in terms of AUC ROC, AUC PR, BCE Loss, and Accuracy.

6.5. Comparison of TCN-MHSA with the Augmented Dataset

In our experiment, we aimed to improve the performance of our proposed model by training it with a dataset that underwent data augmentation. Data augmentation is a technique used to artificially increase the size of a dataset by creating modified versions of the original data samples. By applying various transformations to the existing data, such as rotation, scaling, flipping, and adding noise, we can generate additional samples that capture different variations of the same class.

After training our proposed model on the augmented dataset, we evaluated its performance. We compared the results to a baseline model trained on the original, un-augmented dataset. Our observations revealed that the proposed model achieved better performance specifically for the classes that were initially underrepresented.

Overall, our experiment demonstrated that training our proposed model with a dataset that underwent data augmentation resulted in improved performance, specifically for the underrepresented classes. This approach effectively addressed the challenge of imbalanced datasets, allowing the model to achieve more balanced and accurate predictions across all classes.

Table 6-5 Augmented data-set Result

Maintenance Classes	Model	Augmented	AUC ROC	AUC PR	BCE Loss	Accuracy
C37	TCN-MHSA	No	0.838	0.817	0.516	0.765
		Yes	**0.841**	**0.820**	**0.515**	**0.767**
C29	TCN-MHSA	No	0.841	0.819	0.517	0.762
		Yes	**0.843**	**0.822**	**0.513**	**0.764**
C6	TCN-MHSA	No	0.837	0.821	0.509	0.760
		Yes	**0.840**	**0.823**	**0.508**	**0.762**
C4	TCN-MHSA	No	0.828	0.818	0.513	0.753
		Yes	**0.832**	**0.820**	**0.510**	**0.756**

6.6. Related Work Comparison

In comparison to the base paper(Yang et al., 2022)our proposed TCN-MHSA model exhibited notable improvements in AUC ROC, accuracy (ACCR), and AUC PR metrics.

Our TCN-MHSA model, which combines Temporal Convolutional Networks (TCN) with Multi-Head Self-Attention (MHSA) mechanisms, outperformed the Convolutional Transformers in terms of predictive performance. By incorporating TCN, our model effectively captured temporal dependencies and patterns in the data. Additionally, the integration of MHSA mechanisms enhanced the model's ability to focus on relevant features and extract valuable information.

As a result, our TCN-MHSA model demonstrated an increase in AUC ROC, Accuracy, and AUC PR metrics which is illustrated on the Figure 6.11. These improvements signify the enhanced predictive capabilities of our proposed model compared to the Convolutional Transformers approach presented in the baseline paper.

The proposed TCN-MHSA model has shown an improvement of +2.1% in AUC PR, +2.0% in AUC ROC, +1.8% in Accuracy for the C28 maintenance class and an improvement of +10.6% in AUC PR, +6.3% in AUC ROC, +4.2% in Accuracy for the C37 maintenance class

The table show the comparison between the proposed model and models evaluated by the base paper for the two-maintenance class.

Table 6-6Model Comparison between base paper and proposed model

	Model Type	Augmentation	BCE Loss	ROC	PR	Accuracy
C28	CNN-LSTM	Y	63.0%	70.1%	65.4%	65.3%
		N	61.7%	74.2%	69.7%	68.5%
	CNN-MHSA	Y	52.8%	82.6%	80.2%	74.4%
		N	55.7%	81.9%	79.2%	75.1%
	TCN-MHSA	N	**51.7%**	**84.6%**	**82.3%**	**76.9%**
C37	CNN-LSTM	Y	64.3%	67.4%	56.7%	65.5%
		N	67.9%	55.53%	48.9%	59.6%
	CNN-MHSA	Y	60.1%	77.5%	71.1%	72.3%
		N	68.0%	55.9%	48.5%	59.0%
	TCN-MHSA	N	**51.6%**	**83.8%**	**81.7%**	**76.5%**

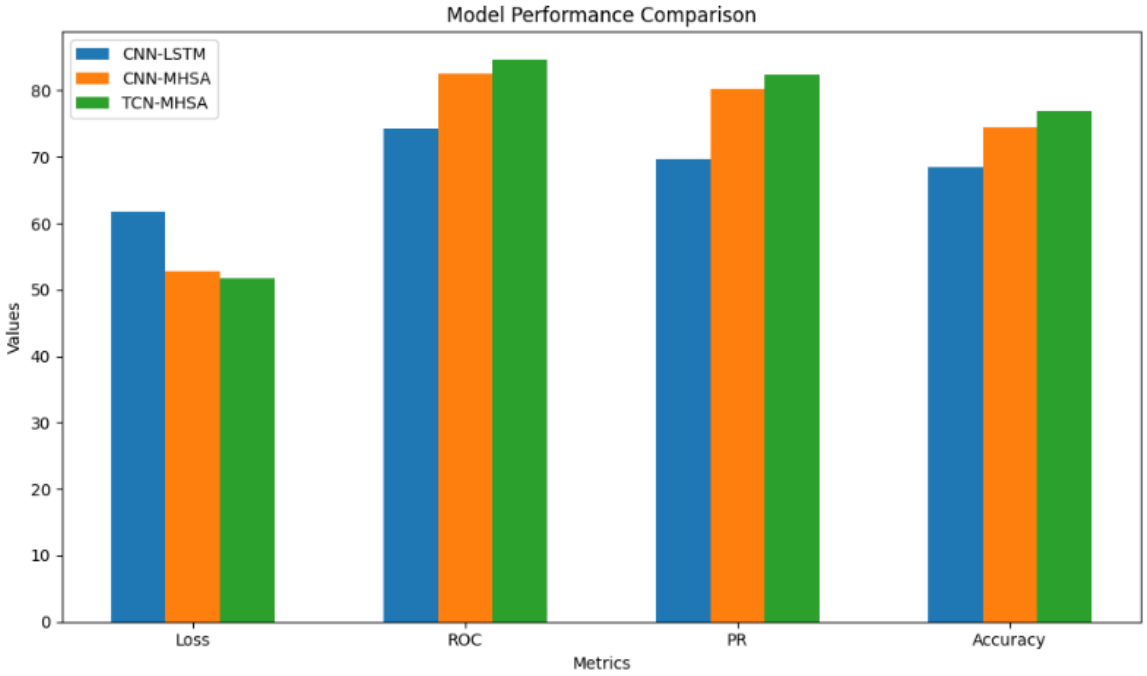

Figure 6-11 Model Comparison between baseline paper and proposed model

6.7. Result Discussion

The experimentation conducted in this research study sought to identify the most effective deep learning model for enhancing the performance of a predictive maintenance system. After some evaluation and analysis, we have determined that the combination of Temporal Convolutional Networks (TCNs) and Multihead Attention consistently outperformed other deep learning architectures in terms of predictive maintenance system enhancement.

The integration of TCNs in the proposed model proved highly advantageous due to their ability to capture temporal dependencies in sequential data. By utilizing dilated convolutions in multiple layers, TCNs effectively modeled the time-series nature of maintenance data, enabling the extraction of intricate temporal patterns. This characteristic made TCNs an optimal choice for analyzing maintenance datasets with varying time intervals and achieving accurate predictions.

Furthermore, the inclusion of Multihead Attention in the combined architecture significantly boosted the model's performance. Multihead Attention introduced a self-attention mechanism that allowed the model to attend to different parts of the input sequence, focusing on the most informative elements for accurate predictions. With the integration of multiple attention heads, the model became adept at capturing various types of dependencies and extracting diverse representations, thereby improving its understanding of critical features relevant to predictive maintenance tasks.

Through extensive experimentation on real-world predictive maintenance datasets, the proposed deep learning model consistently demonstrated superior performance compared to other models. The accuracy of predictions, the efficiency of anomaly detection, and the quality of decision-making associated with maintenance scheduling were notably enhanced by leveraging the strengths of TCNs and Multihead Attention.

6.8. Cost Analysis and Saving: Impact of the Proposed Model

In addition to evaluating the performance and accuracy of the deep learning models developed in this study, it is crucial to examine their financial implications and potential cost savings. This section focuses on the cost analysis associated with the implementation of the models and explores the substantial impact they can have on saving expenses in the maintenance domain.

By considering various scenarios and their corresponding costs, we assess the effectiveness of the models in predicting maintenance needs accurately. Furthermore, we quantify the potential savings that can be achieved when the models correctly identify the need for maintenance or accurately predict the absence of maintenance requirements. This analysis not only provides valuable insights into the financial benefits of deploying these deep learning models but also offers a comprehensive perspective on their practical and economic advantages. The following subsections detail the cost analysis and demonstrate how the models can contribute to significant cost reductions in maintenance operations of the maintenance issue C28 or Intake Gasket Leak/Damage which issue related to engine

Maintenance costs of an aircraft engine can vary significantly depending on the specific maintenance requirements and the extent of the work involved. However, it is important to highlight that proactive and accurate maintenance predictions can result in substantial cost savings. Based on the cost analysis of aircraft engine maintenance, we have conducted an evaluation to determine the potential savings generated by our model, considering the four scenarios outlined by [5] . By taking into account the specific costs associated with each scenario, we have quantified the financial impact of our model on maintenance operations. Through this assessment, we have gained insights into the potential cost savings that can be achieved by utilizing our model's predictions.

When the model correctly predicts no maintenance is needed and no maintenance is actually required, there is no additional cost incurred, resulting in savings of $0. Conversely, when the model predicts no maintenance is needed, but maintenance is actually required, the cost of unnecessary maintenance can amount to a significant loss of $1,000,000. Similarly, when the

[5] https://www.cloudera.com/tutorials/cml-validate-jet-engine-predictive-models.html

model predicts maintenance is needed, but no maintenance is actually required, there is a loss of $250,000 due to unwarranted maintenance expenses. However, when the model accurately predicts the need for maintenance, and maintenance is indeed required, substantial savings of $500,000 can be achieved. These scenarios demonstrated on the figure 6.12.

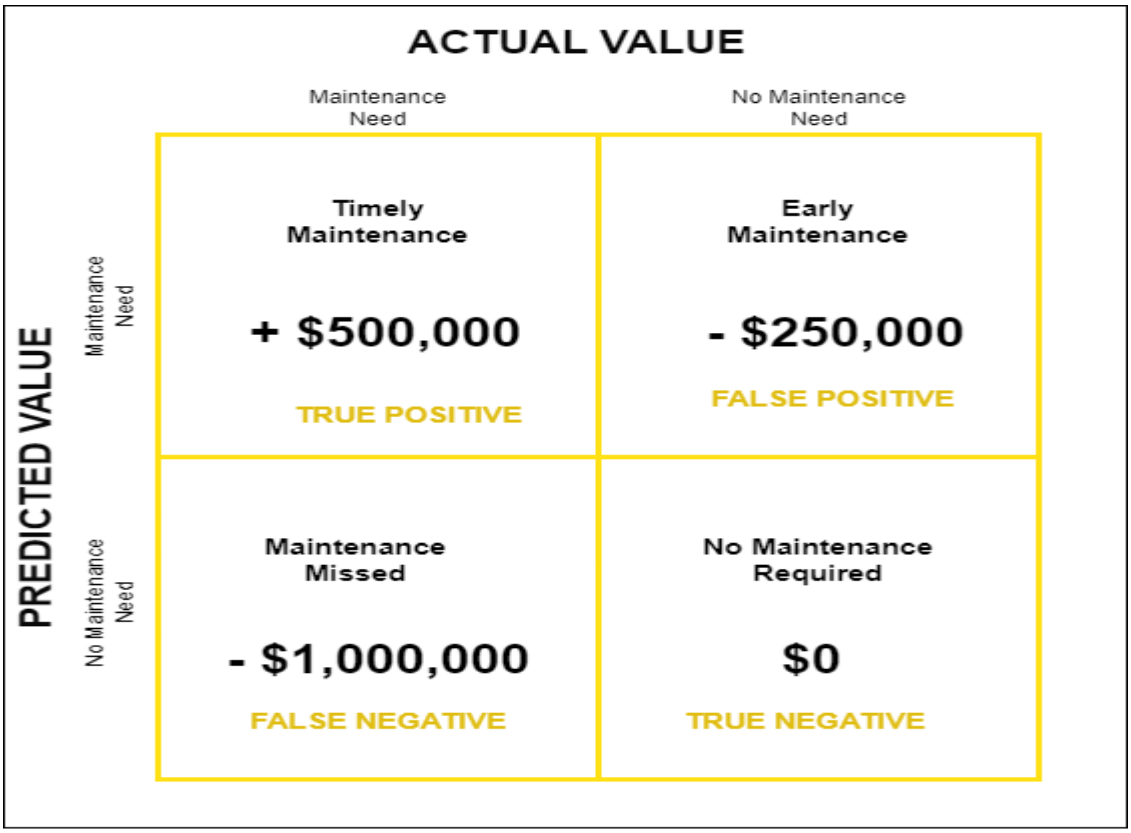

Figure 6-12 Cost analysis

Using the above scenarios:

1. When the model predicts no maintenance is needed and no maintenance is actually required:
   - Savings: $0
2. When the model predicts no maintenance is needed, but maintenance is actually required:
   - Loss: $1,000,000
3. When the model predicts maintenance is needed, but no maintenance is actually required:
   - Loss: $250,000
4. When the model predicts maintenance is needed and maintenance is actually required:
   - Savings: $500,000

Our proposed model's accuracy for the maintenance issue C28 is 76.9% and the cost saving analysis is calculated below: -

1. Probability of model predicting no maintenance needed and no maintenance actually required:
   - 76.9% (accuracy)
2. Probability of model predicting no maintenance needed, but maintenance actually required:
   - 23.1% (100% - accuracy)
3. Probability of model predicting maintenance needed, but no maintenance actually required:
   - 23.1% (100% - accuracy)
4. Probability of model predicting maintenance needed and maintenance actually required:
   - 76.9% (accuracy)

Expected saving = $(\boldsymbol{Probability\ of\ scenario\ 1} * \boldsymbol{Savings\ for\ scenario\ 1}) +$
$(\boldsymbol{Probability\ of\ scenario\ 2} * \boldsymbol{Loss\ for\ scenario\ 2}) +$
$(\boldsymbol{Probability\ of\ scenario\ 3} * \boldsymbol{Loss\ for\ scenario\ 3}) +$
$(\boldsymbol{Probability\ of\ scenario\ 4} * \boldsymbol{Savings\ for\ scenario\ 4})$

Expected savings $= (\mathbf{0.769} * \$\mathbf{0}) + (\mathbf{0.231} * -\$\mathbf{1,000,000}) + (\mathbf{0.231} * -\$\mathbf{250,000}) + (\mathbf{0.769} * \$\mathbf{500,000})$

Expected savings = $95,750

Based on the calculations, the expected savings from using our maintenance prediction model is $95,750. This show that using our proposed model we could on average save around $95,750 which is a huge cost save for the aviation industry.

CHAPTER SEVEN

7. CONCLUSION AND FUTURE WORK

7.1. Conclusion

In conclusion, this research focused on aircraft predictive maintenance and aimed to identify an effective model for accurately predicting whether a flight is in a pre-maintenance or post-maintenance state. Through a series of experiments and evaluations, it has been determined that the combination of Temporal Convolutional Networks (TCN) and Multi-Head Self-Attention performs exceptionally well in this task.

The integration of TCNs in the model proved advantageous due to their ability to capture the sequential nature of maintenance data and extract intricate temporal patterns. By incorporating dilated convolutions in multiple layers, TCNs effectively modeled the time-series data, enabling accurate predictions of the maintenance state.

Additionally, the inclusion of Multi-Head Attention significantly improved the model's performance. By utilizing a self-attention mechanism, the model could focus on the most informative elements of the input sequence for prediction. The use of multiple attention heads allowed the model to capture diverse dependencies and extract relevant features for the maintenance state prediction task.

The experiments conducted on real-world aircraft maintenance datasets demonstrated that the proposed TCN combined with Multi-Head Attention model outperformed other models in accurately predicting the probability of flights being in a pre-maintenance or post-maintenance state. The model's accuracy, reliability, and efficiency were notably enhanced, leading to improved maintenance planning and optimization of aircraft maintenance schedules.

The findings of this research have significant implications for the aviation industry, enabling informed decisions regarding maintenance scheduling, resource allocation, and operational planning. By accurately predicting the maintenance state of flights, disruptions can be minimized, safety can be improved, and maintenance costs can be optimized.

Overall, this research contributes to the field of aircraft predictive maintenance by showcasing the superior performance of the proposed TCN combined with Multihead Attention model. Its successful application highlights its potential for enhancing maintenance practices in the aviation industry, ultimately leading to more efficient and reliable aircraft maintenance operations.

7.2. Future Work

In the realm of predictive maintenance, there are promising avenues for future research and development. One crucial area is expanding the scope of maintenance classification to include a broader range of maintenance classes. By increasing the number of maintenance categories, the model can provide more nuanced insights into specific maintenance issues and enable more targeted actions. This expansion can be achieved through the collection of more comprehensive data and the incorporation of domain expertise to define and classify new maintenance classes. Additionally, another key direction for future work is to develop a predictive maintenance model that not only identifies the occurrence of maintenance but also predicts the specific maintenance issue that is likely to arise. This predictive capability can empower maintenance teams to proactively address potential problems and allocate resources efficiently. By combining these two directions, future research can contribute to a more comprehensive and accurate predictive maintenance framework that enables proactive decision-making and minimizes downtime.

SPECIAL THANKS

This research project is funded by Adama Science and Technology University under the grant number:

ASTU/SM-R/880/23

Adama, Ethiopia

REFERENCES

Adhikari, P., Rao, H. G., & Buderath, Dipl.-I. M. (n.d.). *Machine Learning based Data Driven Diagnostics & Prognostics Framework for Aircraft Predictive Maintenance*.

Bouarfa, S., Doğru, A., Arizar, R., Aydoğan, R., & Serafico, J. (2020). Towards automated aircraft maintenance inspection. A use case of detecting aircraft dents using mask r-cnn. *AIAA Scitech 2020 Forum*, *1 PartF*. https://doi.org/10.2514/6.2020-0389

Celikmih, K., Inan, O., & Uguz, H. (2020). Failure Prediction of Aircraft Equipment Using Machine Learning with a Hybrid Data Preparation Method. *Scientific Programming*, *2020*. https://doi.org/10.1155/2020/8616039

Che, C., Wang, H., Fu, Q., & Ni, X. (2019). Combining multiple deep learning algorithms for prognostic and health management of aircraft. *Aerospace Science and Technology*, *94*, 105423. https://doi.org/10.1016/j.ast.2019.105423

Dangut, M. D., Jennions, I. K., King, S., & Skaf, Z. (2023). A rare failure detection model for aircraft predictive maintenance using a deep hybrid learning approach. *Neural Computing and Applications*, *35*(4), 2991–3009. https://doi.org/10.1007/s00521-022-07167-8

Dong, Y., Tao, J., Zhang, Y., Lin, W., & Ai, J. (2021). Deep Learning in Aircraft Design, Dynamics, and Control: Review and Prospects. *IEEE Transactions on Aerospace and Electronic Systems*, *57*(4), 2346–2368. https://doi.org/10.1109/TAES.2021.3056086

Gohel, H. A., Upadhyay, H., Lagos, L., Cooper, K., & Sanzetenea, A. (2020). *Predictive maintenance architecture development for nuclear infrastructure using machine learning*. https://doi.org/10.1016/j.net.2019.12.029

Gopali, S., Abri, F., Siami-Namini, S., & Namin, A. S. (2021). A Comparison of TCN and LSTM Models in Detecting Anomalies in Time Series Data. *2021 IEEE International Conference on Big Data (Big Data)*, 2415–2420. https://doi.org/10.1109/BigData52589.2021.9671488

Hashemian, H. M. (2011). State-of-the-Art Predictive Maintenance Techniques. *IEEE Transactions on Instrumentation and Measurement*, *60*(1), 226–236. https://doi.org/10.1109/TIM.2010.2047662

Hermawan, A. P., Kim, D.-S., & Lee, J.-M. (2020). Predictive Maintenance of Aircraft Engine using Deep Learning Technique. *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, 1296–1298. https://doi.org/10.1109/ICTC49870.2020.9289466

Hsieh, T.-Y., Wang, S., Sun, Y., & Honavar, V. (2020). *Explainable Multivariate Time Series Classification: A Deep Neural Network Which Learns To Attend To Important Variables As Well As Informative Time Intervals*. http://arxiv.org/abs/2011.11631

Kanawaday, A., & Sane, A. (2017). Machine learning for predictive maintenance of industrial machines using IoT sensor data. *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, 87–90. https://doi.org/10.1109/ICSESS.2017.8342870

Khan, K., Sohaib, M., Rashid, A., Ali, S., Akbar, H., Basit, A., & Ahmad, T. (2021). Recent trends and challenges in predictive maintenance of aircraft's engine and hydraulic system. In *Journal of the Brazilian Society of Mechanical Sciences and Engineering* (Vol. 43, Issue 8). Springer Science and Business Media Deutschland GmbH. https://doi.org/10.1007/s40430-021-03121-2

Lee, J., & Mitici, M. (2023). Deep reinforcement learning for predictive aircraft maintenance using probabilistic Remaining-Useful-Life prognostics. *Reliability Engineering & System Safety*, *230*, 108908. https://doi.org/https://doi.org/10.1016/j.ress.2022.108908

Makhija, P., & Chacko, E. (2021). Efficiency and Advancement of Artificial Intelligence in Service Sector with Special Reference to Banking Industry. In *Fourth Industrial Revolution and Business Dynamics* (pp. 21–35). Springer Singapore. https://doi.org/10.1007/978-981-16-3250-1_2

Mehtab, S., & Sen, J. (2020). *Stock Price Prediction Using CNN and LSTM-Based Deep Learning Models*.

Ning, S., Sun, J., Liu, C., & Yi, Y. (2021). Applications of deep learning in big data analytics for aircraft complex system anomaly detection. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, *235*(5), 923–940. https://doi.org/10.1177/1748006X211001979

Paolanti, M., Romeo, L., Felicetti, A., Mancini, A., Frontoni, E., & Loncarski, J. (2018). Machine Learning approach for Predictive Maintenance in Industry 4.0. *2018 14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA)*, 1–6. https://doi.org/10.1109/MESA.2018.8449150

Serradilla, O., Zugasti, E., Rodriguez, J., & Zurutuza, U. (2022). Deep learning models for predictive maintenance: a survey, comparison, challenges and prospects. *Applied Intelligence*, *52*(10), 10934–10964. https://doi.org/10.1007/s10489-021-03004-y

Verhagen, W. J. C., & De Boer, L. W. M. (2018). Predictive maintenance for aircraft components using proportional hazard models. *Journal of Industrial Information Integration*, *12*, 23–30. https://doi.org/10.1016/j.jii.2018.04.004

Wan, R., Tian, C., Zhang, W., Deng, W., & Yang, F. (2022). A Multivariate Temporal Convolutional Attention Network for Time-Series Forecasting. *Electronics (Switzerland)*, *11*(10). https://doi.org/10.3390/electronics11101516

Xu, G., Liu, M., Wang, J., Ma, Y., Wang, J., Li, F., & Shen, W. (2019). Data-Driven Fault Diagnostics and Prognostics for Predictive Maintenance: A Brief Overview. *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, 103–108. https://doi.org/10.1109/COASE.2019.8843068

Yang, H., & Desell, T. (2022). *A Large-Scale Annotated Multivariate Time Series Aviation Maintenance Dataset from the NGAFID*. http://arxiv.org/abs/2210.07317

Yang, H., Labella, A., & Desell, T. (2022). *Predictive Maintenance for General Aviation Using Convolutional Transformers*. www.aaai.org

APPENDICES

Appendix A : Maintenance Issues

No	Class Name
1	Aircraft start/external issue
2	Baffle bracket loose/damage
3	Baffle cracj/damage/loose/miss
4	Baffle mount loose/damage
5	Baffle plug need repair/replace
6	Baffle rivet loose/miss/damage
7	Baffle screw miss/loose
8	Baffle screw loose/damage
9	Baffle seal loose/damage
10	Baffle spring damage
11	Baffle tie/tie rod loose or damage
12	Cowling miss/loose/damage
13	Cylinder compression issue
14	Cylinder crack/fail/need part repair
15	Cylinder exhaust value/stuck value issue
16	Cylinder head/exhaust gas tempreature issue
17	Cylinder/exhaust push rod/tube damage
18	Drain line/tube damage
19	Engine crankcase/crankshaft/firewall near repair
20	Engine failure/fire/time out
21	Engine idle /rpm issue
22	Engine need repair/reinstall/clean
23	Engine run rough
24	Engine seal/tube/bolt loose or damage
25	Engine/propeller overspeed or damage
26	Induction damage/hardware fail
27	Intake gasket leak/damage
28	Intake tube/bolt/seal/boot loose or damage
29	Magneto failure
30	Mixture fail/need adjust
31	Oil cooler need maintenance
32	Oil dispstick/tube need repair
33	Oil leak/pressure issue
34	Oil return line issue
35	Pilot/in-flight noticed issue
36	Rocker cover leak/loose/damage
37	Spark plug need repair/replace

Appendix B : Sample Code

```
import dask.dataframe as dd
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn import preprocessing
import tensorflow as tf
import dask.dataframe.io as ddio

# Reading the DataSet
# Reading the PARQUET DataSet
flight_data_df = dd.read_parquet('all_flights/one_parq')

# Reading the hEADER DataSet
flight_header_df = pd.read_csv('all_flights/flight_header.csv', index_col = 'Master Index')

# Displaying the flight_header_df
flight_header_df.head(5)
```

```
# Compute the class distribution in the "cluster" column
class_distribution = flight_data_df["cluster"].value_counts().compute()

# Plot the class distribution as a side-by-side bar graph
sns.countplot(x=flight_data_df["cluster"], data=class_distribution)
# Filter out rows where the "cluster" column is equal to "c_28"
filtered_df_c_28 = flight_data_df.query("cluster == 'c_28'")
# Filter out rows where the "cluster" column is equal to "c_28"
filtered_df_c_37 = flight_data_df.query("cluster == 'c_37'")
# Filter out rows where the "cluster" column is equal to "c_28"

filtered_df_c_29 = flight_data_df.query("cluster == 'c_29'")
# Filter out rows where the "cluster" column is equal to "c_28"
filtered_df_c_4 = flight_data_df.query("cluster == 'c_4'")
# Filter out rows where the "cluster" column is equal to "c_28"
filtered_df_c_6 = flight_data_df.query("cluster == 'c_6'")
```

```python
# function to filter dataset which have date_diff between 3 days

def select_date_diff(dataset):
    selected_dates = [-3, -2, -1, 1, 2, 3]
    filtered_dataset = dataset[dataset.date_diff.isin(selected_dates)]
    return filtered_dataset

C_28 = dd.read_parquet('Clusters/C_28')
C_37 = dd.read_parquet('Clusters/C_37')
C_29 = dd.read_parquet('Clusters/C_29')
C_6 = dd.read_parquet('Clusters/C_6')
C_4 = dd.read_parquet('Clusters/C_4')

C_37_filtered = select_date_diff(C_37)
C_29_filtered = select_date_diff(C_29)
C_6_filtered = select_date_diff(C_6)
C_4_filtered = select_date_diff(C_4)

ddio.to_parquet(C_37_filtered, "Classes/C_37/")
ddio.to_parquet(C_29_filtered, "Classes/C_29/")
ddio.to_parquet(C_6_filtered, "Classes/C_6/")
ddio.to_parquet(C_4_filtered, "Classes/C_4/")
```

```
import numpy as np
import random
import tensorflow.compat.v2 as tf
tf.enable_v2_behavior()
import tensorflow_datasets as tfds
import tensorflow_probability as tfp
from tqdm.notebook import tqdm
tfk = tf.keras
tfkl = tf.keras.layers
tfpl = tfp.layers
tfd = tfp.distributions
from sklearn.model_selection import train_test_split, KFold
from sklearn import preprocessing
tfk = tf.keras
tfkl = tf.keras.layers
tfpl = tfp.layers
import tensorflow_addons as tfa
import pandas as pd
import random
import shutil
import sklearn as sk
import seaborn as se
import keras as k
import dask as dd

# Parameters
BATCH_SIZE = 32
SHAPE = (4096, 23)
BLOCK_SHAPE = (128, 23)
EPOCHS = 30
VARIABLES = 23
ENCODED_SIZE = 512
NFOLD = 5
AUGMENT = False
MODEL_TYPE = 'get_tcn_modelfinal' #  get_conmhsa, get_tcn, get_tcn_modelfinal,get_cnn
MODEL_LOSS_TYPE = 'bce'
PREDICT = MODEL_LOSS_TYPE == 'bce'
STEPS_PER_EPOCH = 250
DATASET = 'C28' #C28 , C37, C29, C6,C4
```

```python
# TF DATASET
# This converts the dataframe into TF Records.
def get_dataset(df):
    ids = df.id.unique()

    sensor_datas = []
    afters = []

    for id in tqdm(ids):
        sensor_data = df[df.id == id].iloc[-SHAPE[0]:, :23].values

        sensor_data = np.pad(sensor_data, [[0, SHAPE[0]- len(sensor_data)], [0,0]])

        sensor_data = tf.convert_to_tensor(sensor_data, dtype = tf.float32)

        after = df[df.id == id]['before_after'].iloc[0]

        sensor_datas.append(sensor_data)
        afters.append(after)


    sensor_datas = tf.stack(sensor_datas)
    afters = np.stack(afters)

    ds = tf.data.Dataset.from_tensor_slices( (sensor_datas, afters))

    return ds

def fix_type(x, y):
    return tf.cast(x, tf.float32), tf.cast(y, tf.float32)


def prepare_for_training(ds, shuffle = False, repeat = False, predict = True):

    ds = ds.map(fix_type)
    ds = ds.map(random_shift)
    ds = ds.shuffle(512) if shuffle else ds
    ds = ds.repeat() if repeat else ds
    ds = ds.batch(BATCH_SIZE, drop_remainder=True)


    if not predict:
        ds = ds.map(lambda x, y : (x, x) )
    else:
        # ds = ds.map(lambda x, y : (x, tf.one_hot(y, 4)) )
        ds = ds.map(lambda x, y : (x, tf.reshape(y, (-1, 1))) )

    return ds
```

```python
# MHSA ENCODER
def point_wise_feed_forward_network(d_model, dff):
  return tf.keras.Sequential([tf.keras.layers.Dense(dff, activation='relu'),
      tf.keras.layers.Dense(d_model) ])

def scaled_dot_product_attention(q, k, v, mask):
  matmul_qk = tf.matmul(q, k, transpose_b=True)

  # scale matmul_qk
  dk = tf.cast(tf.shape(k)[-1], tf.float32)
  scaled_attention_logits = matmul_qk / tf.math.sqrt(dk)
  if mask is not None:
    scaled_attention_logits += (mask * -1e9)
  attention_weights = tf.nn.softmax(scaled_attention_logits, axis=-1)

  output = tf.matmul(attention_weights, v)

  return output, attention_weights

class MultiHeadAttention(tf.keras.layers.Layer):
  def __init__(self, d_model, num_heads):
    super(MultiHeadAttention, self).__init__()
    self.num_heads = num_heads
    self.d_model = d_model

    assert d_model % self.num_heads == 0

    self.depth = d_model // self.num_heads

    self.wq = tf.keras.layers.Dense(d_model)
    self.wk = tf.keras.layers.Dense(d_model)
    self.wv = tf.keras.layers.Dense(d_model)

    self.dense = tf.keras.layers.Dense(d_model)

  def split_heads(self, x, batch_size):
    x = tf.reshape(x, (batch_size, -1, self.num_heads, self.depth))
    return tf.transpose(x, perm=[0, 2, 1, 3])

  def call(self, v, k, q, mask):
    batch_size = tf.shape(q)[0]

    q = self.wq(q)
    k = self.wk(k)
    v = self.wv(v)

    q = self.split_heads(q, batch_size)
    k = self.split_heads(k, batch_size)
    v = self.split_heads(v, batch_size)

    scaled_attention, attention_weights = scaled_dot_product_attention( q, k, v, mask)

    scaled_attention = tf.transpose(scaled_attention, perm=[0, 2, 1, 3])
    concat_attention = tf.reshape(scaled_attention,(batch_size, -1, self.d_model))
    output = self.dense(concat_attention)

    return output, attention_weights
```

```python
# MHSA ENCODER
class EncoderLayer(tf.keras.layers.Layer):
  def __init__(self, d_model, num_heads, dff, rate=0.1):
    super(EncoderLayer, self).__init__()

    self.mha = MultiHeadAttention(d_model, num_heads)
    self.ffn = point_wise_feed_forward_network(d_model, dff)

    self.layernorm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
    self.layernorm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)

    self.dropout1 = tf.keras.layers.Dropout(rate)
    self.dropout2 = tf.keras.layers.Dropout(rate)

  def call(self, x, training, mask = None):

    attn_output, self.attention_values = self.mha(x, x, x, mask)
    attn_output = self.dropout1(attn_output, training=training)
    out1 = self.layernorm1(x + attn_output)

    ffn_output = self.ffn(out1)
    ffn_output = self.dropout2(ffn_output, training=training)
    out2 = self.layernorm2(out1 + ffn_output)

    return out2
```

```python
#Proposed Model
def get_TCNMHSAfinal():

    # Define input shape
    inputs = tf.keras.Input(shape=SHAPE)

    # Dilated convolution layers
    x = tf.keras.layers.Conv1D(filters=128, kernel_size=7, padding='same', activation='relu', dilation_rate=1)(inputs)
    x = tf.keras.layers.Conv1D(filters=128, kernel_size=7, padding='same', activation='relu', dilation_rate=2)(x)
    x = tf.keras.layers.Conv1D(filters=256, kernel_size=3, padding='same', activation='relu', dilation_rate=1)(x)
    x = tf.keras.layers.Conv1D(filters=256, kernel_size=3, padding='same', activation='relu', dilation_rate=2)(x)
    x = tf.keras.layers.Conv1D(filters=512, kernel_size=7, padding='same', activation='relu', dilation_rate=1)(x)
    x = tf.keras.layers.Conv1D(filters=512, kernel_size=7, padding='same', activation='relu', dilation_rate=2)(x)
    x= EncoderLayer(d_model=512, num_heads=8, dff=512)(x)
    x= EncoderLayer(d_model=512, num_heads=8, dff=512)(x)
    x= EncoderLayer(d_model=512, num_heads=8, dff=512)(x)
    x= EncoderLayer(d_model=512, num_heads=8, dff=512)(x)

    # Global average pooling layer
    x = tf.keras.layers.GlobalAveragePooling1D()(x)

    # Output layer
    outputs = tf.keras.layers.Dense(units=1, activation='sigmoid')(x)

    # Define model
    model = tf.keras.Model(inputs=inputs, outputs=outputs)

    return model
```