TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

B-075-BAS-2078/79

FAULT DIAGNOSIS OF A BALL BEARING USING VIBRATION ANALYSIS

By:

Mikesh Paudel (075AER023)

Sumit Bhatta (075AER045)

Sushil Sapkota (075AER046)

A PROJECT REPORT

SUBMITTED TO THE DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR'S IN AEROSPACE ENGINEERING

DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING

LALITPUR, NEPAL

June, 2023

COPYRIGHT

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING

The undersigned certify that they have read, and recommended to the Institute of Engineering for acceptance, a project report entitled "FAULT DIAGNOSIS OF A BALL BEARING USING VIBRATION ANALYSIS" submitted by Mikesh Paudel, Sumit Bhatta and Sushil Sapkota in partial fulfillment of the requirements for the degree of Bachelor of Mechanical and Aerospace Engineering.

Supervisor, Prof. Dr. Mahesh Chandra Luintel

Department of Mechanical and Aerospace Engineering

Supervisor, Asst. Prof. Aayush Bhattarai

Department of Mechanical and Aerospace Engineering

External Examiner, Mr. Spad Acharya

Lecturer, Sagarmatha Engineering College

Assoc. Prof Dr. Surya Prasad Adhikari

Head of Department

Department of Mechanical and Aerospace Engineering

Date: _____

ABSTRACT

Failure of ball bearings is a major cause of rotatory machine failure resulting in large economic losses and possible injury to human lives. Correct diagnosis helps to identify bearing faults and use the bearings effectively, preventing catastrophic failures of rotating machines. Detecting potential problems early, and condition monitoring allows for proactive maintenance and reduces the downtime of machines. Improved equipment reliability and efficiency lead to lower maintenance costs and increased productivity. The study aimed to classify three types of bearing faults: Inner Raceway, Outer Raceway and Ball fault. Inner Raceway and Outer Raceway faults were introduced in the 608-deep groove ball bearing via an electric grinder making 1.5mm line cuts through the axis of the bearing. For ball fault, one ball was removed out of the 7 present. An accelerometer with sampling frequency of 1000Hz was fixed on the drive end of the AC induction motor to acquire the vibration signals. Models for Support Vector Machine (SVM), Convolutional Neural Network (CNN) (both 1D CNN and 2D CNN) and Long Short Term Memory Network (LSTM) were developed. Raw bearing fault data from an open-source database, CWRU was fed into the models to check their accuracy. A minimum accuracy of 92.47% was acquired from the raw CWRU data, thus validating the models. The acquired fault data from the accelerometer was processed through a (20Hz, 500Hz) band pass filter before feeding into the machine learning and deep learning models. 70% of the vibration data was used for training the models while the remaining 30% was used for testing. Out of the 4 models compared, 1D CNN gave a maximum test accuracy of 98.35%.

Keywords: *Fault Diagnosis, Ball Bearing, Machine Learning, Deep Learning, SVM, LSTM, CNN*

ACKNOWLEDGEMENT

We would like to extend our sincerest of gratitude to everyone who helped in the completion of the project. First of all, we would like to thank the Department of Mechanical and Aerospace Engineering for providing us with the opportunity, workspace and helping us with some needed instruments to complete this project. We would like to thank our supervisors, Assistant Professor Mr. Aayush Bhattarai and Professor Dr. Mahesh Chandra Luintel for providing us with their constant help and supervision throughout the entirety of the project.

We would like to thank, Mr. Kamal Darlami, Deputy Head of the Department, for helping us to with his valuable insights which helped to change the dynamics of the project. We are also grateful towards Assistant Professor, Dr. Sudip Bhattarai, for his constant help throughout the project, ranging from his valiant engine startup attempts to providing insights in signal processing and filtering the acquired data. We would like to thank Mr. Laxman Motra for providing us with tachometer and 3D Vibration Tester for our project. Our sincere gratitude towards Assistant Professor, Mr. Aashish Karki, for providing valuable insights to build the experimental setup.

We would like to express our hearty gratitude towards Mr. Bibek Parajuli for working as the unofficial fourth member of our project. We would also like to thank Mr. Ganesh Dhungana, Mr. Sandesh Parajuli and Mr. Sandip Gautam for helping us during the crucial phases of the project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

R	Real Number
N	Number of data points
Exp()	Exponential Function
X_n	N^{th} data point
C	Hyperparameter in SVM
g	Acceleration due to gravity
Xi	Input Vector
Yi	Label of input vector

LIST OF ACRONYMS AND ABBREVIATIONS

SVM	Support Vector Machine
CBM	Condition Based Monitoring
ML	Machine Learning
DL	Deep Learning
CNN	Convolutional Neural Network
LSTM	Long Short Term Memory
GAN	Generative Adversarial Network
RMSE	Root Mean Square Error
FT	Fourier Transform
FFT	Fast Fourier Transform
STFT	Short Time Fourier Transform
EMD	Empirical Mode Decomposition
WPT	Wavelet Packet Transformation
IMF	Intrinsic Mode Functions
AI	Artificial Intelligence
KNN	K Nearest Neighbor
ANN	Artificial Neural Network
CWRU	Case Western Reserve University
HDD	Hard Disk Drive
CPU	Central Processing Unit
RNN	Recurrent Neural Network
MLP	Multilayer Perceptron
OR	Outer Raceway
IR	Inner Raceway
PCA	Principle Component Analysis
DE	Drive End

Chapter 1 : INTRODUCTION

1.1 Background

Bearings are among the most crucial components of almost every rotating machine. Failure of bearings can lead to large downtime, and equipment failures and can cause injuries to human lives. Lack of lubrication, contamination, fatigue, and misalignment are some of the most common causes of bearing failure. Failure of bearing is the result of a failure of a specific component or components. When a specific bearing component fails, its vibration signature changes. Based on this change in vibration signature from the normal condition, the bearing faults can be diagnosed using Machine Learning and Deep Learning algorithms.

1.1.1 Ball Bearings

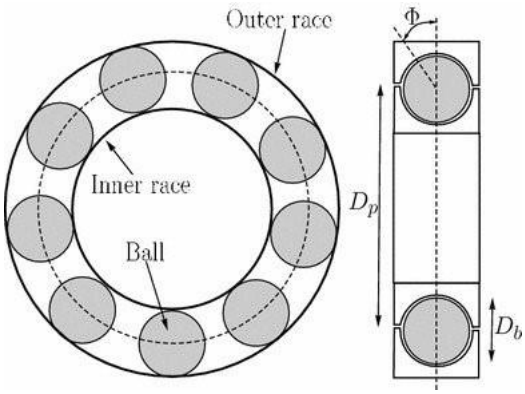

Figure 1.1: Parts of a Ball Bearing

Among the various types of bearings, ball bearings are a common type that are able to sustain both radial and axial loads [2]. Ball bearings are composed of two layers: an inner ring and an outer ring separated by rolling elements. The rolling elements allow smooth, high-speed motion and reduce friction. Ball bearings are mainly made out of ceramics, stainless steel or chrome steel depending upon the requirement of the machinery. They come in sizes ranging from 1mm to 26 mm. These bearings may have deep grove radial structure, single-row angular contact with internal clearances, raceway grooves, mast guide, v-grooves, etc. as per the required loading

conditions [3]. It is important to select proper ball bearings to reduce the risk of failure and avoid high maintenance costs.

Bearing faults are mainly classified on the basis of their parts as: Inner Raceway (IR) faults, Outer Raceway (OR) faults and ball faults.

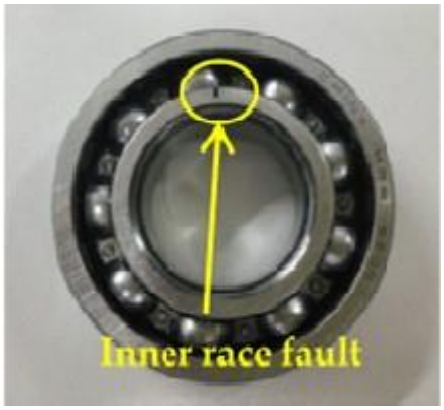

Figure 1. 2: Inner Raceway Fault

Figure 1. 3: Outer Raceway Fault

1.1.2 AC Induction Motor

Figure 1. 4: AC Induction Motor

An AC induction motor, also known as an asynchronous motor, is a type of electric motor that operates on the principle of electromagnetic induction. It is the most commonly used type of motor in industrial and commercial applications. The AC induction motor consists of a stator, which is the stationary part of the motor, and a rotor, which is the rotating part. The stator has a series of windings that are connected to an AC power source. When the power is turned on, the AC current in the stator windings produces a rotating magnetic field. The rotor is made up of a series of conductive bars or coils, and when the rotating magnetic field of the stator cuts across the rotor, it induces an electric current in the rotor. The interaction between the magnetic fields of the stator and the rotor causes the rotor to rotate, and the motor shaft turns. The speed of the motor is determined by the frequency of the AC power supply and the number of poles in the stator windings. The torque of the motor is determined by the strength of the magnetic field and the current in the rotor. AC induction motors are widely used in applications such as pumps, fans, compressors, conveyors, and machine tools. They are rugged, reliable, and efficient, and they require little maintenance. They are also relatively inexpensive to manufacture, making them a popular choice for many industrial and commercial applications.

1.1.3 Signal Processing

Signal processing is the process of analyzing, modifying and transforming signals that represent some form of information. Signal processing involves wide range of techniques for signal processing including filtering, Fourier analysis, spectral analysis, time-frequency analysis, digital signal processing and statistical signal processing. These techniques can be used to extract useful information from signals, remove noise or unwanted components, compress data and prepare signals for further analysis or transmission. Signal processing is an important step in preparing vibration data for machine learning (ML) and deep learning (DL) algorithms. The goal of signal processing is to extract meaningful features from the raw vibration data that can be used as inputs to ML/DL models.

Some of the common signal processing techniques that can be applied to vibration data are:

1. Filtering: This involves removing unwanted noise or frequencies from the signal. For example, a high-pass filter can be used to remove low-frequency noise from the data, while a low-pass filter can be used to remove high-frequency noise.
2. Feature extraction: This involves identifying specific characteristics of the signal that are relevant to the problem at hand. For example, the amplitude, frequency, and phase of specific vibration modes can be extracted from the signal and used as input features.
3. Time-frequency analysis: This involves analyzing the signal in both the time and frequency domains to identify patterns that are relevant to the problem at hand. For example, a spectrogram can be used to visualize how the frequency content of the signal changes over time.
4. Dimensionality reduction: This involves reducing the number of input features by projecting the data onto a lower-dimensional space. For example, principal component analysis (PCA) can be used to identify the most important features in the data and project the data onto a lower-dimensional space.

1.1.4 Machine Learning (ML)

Machine learning refers to a branch of artificial intelligence that enables systems to learn and improve from experience without being explicitly programmed, allowing them to automatically make predictions and recognize patterns. It can be mostly divided into three categories: unsupervised learning, supervised learning and reinforcement learning. Unsupervised learning deals with the clustering of data in a certain pattern that the machine learns to be fit. The output isn't labeled so most of the work the machine does is pattern recognition and division of the data into a number of subsets. Clustering, anomaly detection and dimensionality reduction are some of the tasks unsupervised learning algorithms are good at. K-means clustering, K Nearest Neighbor (KNN), Apriori are some widely used unsupervised learning algorithms. In vibration analysis, K-Nearest Neighbor has helped to classify fault via interfering vibration source with an accuracy of 98.53% [11].

Supervised learning requires algorithm to be trained by labeling the input variables and output datas. They are mostly used for regression and classification problems. Supervised learning algorithms including Linear regression, random forest and Support Vector Machines(SVM) have been used in recent years for ball bearing fault diagnosis with accuracy above 97%[12][13] .

Reinforcement learning implies that machines learn from their own mistakes and successes. The punishment-reward approach based on the mistakes and successes the machine encounters helps it to learn and unlearn. Robotics and gaming are some core fields that use reinforcement learning.

1.1.5 Deep Learning (DL)

Deep learning is a category of Machine learning based on Artificial Neural Networks(ANN) that use multiple hidden layers to extract features from the input signal or image. They are highly used in image processing, vibration analysis, computer vision and climate change programs. Recently, Convolutional Neural Networks (CNN), Long Short Term Memory Networks (LSTMs), Deep Belief

Networks[14], Deep Autoencoders[15] and General Adversarial Networks(GANs) are being used in bearing fault detection problems.

1.1.6 Case Western Reserve University Dataset

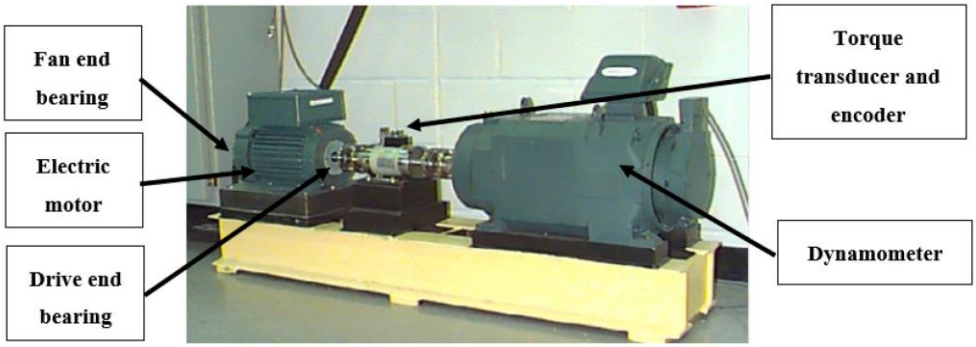

Figure 1. 5: CWRU Experimental Setup

The CWRU Bearing Dataset is a publicly available dataset of vibration signals from various fault conditions in bearings. It was created by researchers at Case Western Reserve University and is widely used for research purposes in the fields of machine learning, signal processing, and condition monitoring. The dataset contains time-domain vibration signals collected from four different bearing types under different fault conditions, including inner race, outer race, ball, and normal conditions. The dataset provides a valuable resource for researchers and engineers to develop, test, and compare various condition monitoring and fault diagnosis methods for bearings. For initial training and testing of algorithms the CWRU dataset has been used.

1.2 Problem Statement

Fault diagnosis on a ball bearing using vibration monitoring is crucial for the proper functioning and longevity of the engine.[16] The traditional methods of fault diagnosis are prone to human error and can lead to inaccurate results. This project aims to improve the accuracy and efficiency of fault diagnosis by using Machine Learning (ML) and Deep Learning (DL) algorithms to develop an automated system. The system will be trained on a dataset of engine vibration data collected from various operational scenarios and will be able to classify faults in real-time as one of the three types: Outer race fault, Inner race fault, and Ball fault. The goal of

this project is to accurately and quickly diagnose faults in engine bearings, reducing downtime and maintenance costs and ensuring optimal engine performance.

1.3 Objectives

1.3.1 Main Objective

- To correctly diagnose typical faults induced in ball bearing using vibration signature analysis via Machine Learning and Deep Learning.

1.3.2 Secondary Objectives

1. To improve the accuracy of fault diagnosis compared to traditional methods, reducing the number of false alarms and improving overall reliability.

2. To develop an experimental setup to obtain vibration signal for fault diagnosis.

3. To demonstrate the viability and benefits of using ML/DL algorithms for fault diagnosis.

1.4 Applications

The fault diagnosis system developed for a ball bearing using vibration monitoring and ML/DL algorithms has a wide range of applications in various industries. Some of the key applications are:

- Predictive Maintenance: The system can be integrated into the engine monitoring system to provide real-time fault diagnosis and allow for proactive maintenance. This will help reduce downtime and increase the lifespan of the engine.

- Quality Control: The system can be used in the manufacturing process of engines to perform quality control checks and ensure that the engines are free of faults before they are shipped.

- Aerospace and Defense: The system can be used in aerospace and defense applications where the reliability and performance of engines is critical. Real-time fault diagnosis will help ensure that the engines are functioning optimally and prevent potential failures.

- Automotive Industry: The system can be used in the automotive industry to diagnose faults in vehicles and improve their performance.

- Energy Generation: The system can be used in power plants and other energy generation facilities to diagnose faults in engines used for power generation and ensure their optimal performance.

Overall, the fault diagnosis system has the potential to revolutionize the way faults are diagnosed in engines, leading to improved reliability, performance, and cost savings across various industries.

1.5 Features

The project comes along with the following features:

- Real-time Fault Diagnosis: The system will be able to diagnose faults in engine bearings in real-time, providing immediate feedback to maintenance personnel.

- Accurate Classification: The system will be trained on a large dataset of engine vibration data, ensuring accurate and reliable classification of faults as one of the three types: Outer race fault, Inner race fault, and Ball fault.

- Cost-effective: The system will help reduce maintenance costs by allowing for proactive maintenance and reducing the need for manual inspection.

- Scalable: The system will be scalable, allowing for integration into various types of engines and industries.

- Continuous Improvement: The system will be continuously improved through the use of machine learning algorithms, allowing for improved accuracy and performance over time.

1.6 Feasibility

1.6.1 Economic Feasibility

The project was found to be economically feasible. The majority of the initial investment for our project involved the acquisition of an AC induction motor, ADXL 335 accelerometer, and DAQ system. Some of these materials were readily available within our department, while others were inexpensive to purchase from the market. Additionally, the project members themselves constructed the experimental setup using locally available materials. To ensure the economic feasibility of our project, we utilized an accelerometer with lower sensitivity and an AC induction motor with low KV ratings. This allowed for the accelerometer's sampling rate to be sufficient in collecting vibration data for fault diagnosis. Moreover, our project did not require significant operational costs.

1.6.2 Technical Feasibility

The technical feasibility of the project for fault diagnosis of bearings using vibration monitoring is positive. The project requires an engine/motor with a bearing to collect vibration data, and it was readily available in the Department of Mechanical and Aerospace Engineering. Data acquisition (DAQ) systems for logging vibration data were also available in the department, and it was easy to procure one for this project. Signal processing, machine learning, and deep learning are essential components of fault diagnosis using vibration data. Project members had the required knowledge in these areas to effectively utilize these techniques to analyze the vibration data and diagnose faults. Additionally, computational systems for data analysis and modeling are required, and these systems were available with the project members. Given the availability of required resources and expertise, it appears that the project is technically feasible and has a high likelihood of success in diagnosing faults in

bearings using vibration monitoring if someone wants to do similar project in the future.

1.6.3 Operational Feasibility

Necessary resources, including workspace and equipment, are available. User friendly design, integration with existing workflows, and real-time monitoring capabilities ensure efficient and effective implementation.

1.7 System Requirements

1.7.1 Software Requirements

For the ADXL 335 accelerometer, a data logger software (LABVIEW) is required to output the data along with excel and Matlab for data collection, storage and sorting. For Machine learning and Deep Learning along with signal processing, Visual Studio Code with Python and Jupyter have been used. The major libraries required are scipy, numpy, matplotlib, fft, sklearn, keras and tensorflow. Intel Core i7-9700K processor with a clock speed of 3.60 GHz, 16 GB of RAM were the available system specifications under which the project was completed. Matlab software is also required for data processing and importation.

1.7.2 Hardware Requirements

The project is focused upon experimental data collection and diagnosis program development. Experimentation is done on a single motor fastened bearing and the setup is similar to the CWRU fault bearing setup. The hardware used in the project are:

- **AC Induction Motor**

  An ac induction motor is a type of electric motor that used electromagnetic induction to generate torque. When AC voltage is applied to the stator windings a rotating magnetic field is produced that interacts with the rotor to generate torque.

The reason behind the use of ac induction motor for the fault diagnosis was that it was easy to start the motor and also the vibration produced in the bearing were pronounced enough to be detected by accelerometer.

- **ADXL 335 accelerometer**

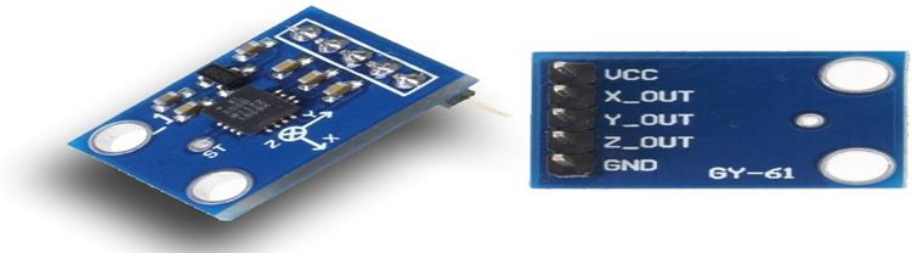

Figure 1. 6: ADXL 335 Accelerometer

ADXL 335 accelerometer was used for acquiring vibration data for fault diagnosis in bearings. It is a low power, three axis accelerometer that can measure acceleration in the range of ±3g. The accelerometer outputs analog voltage signals that correspond to the acceleration in each of the three axes. These signals are read and processed using an analog to digital converter (ADC) to obtain digital vibration data.

- **DAQ system**

Data Acquisition (DAQ) system was used to measure and record vibration data from ADXL 335 accelerometer. The accelerometer was connected to DAQ using jumper wires. The analog acceleration signal was converted to digital data and the digital data was captured and stored at a sampling rate of 1000Hz. myDAQ is a data acquisition device which is used to measure and record vibration data from bearing through accelerometer. myDAQ has high resolution analog to digital converters (ADCs) that can capture vibration signals with high accuracy and

precision. This ensures that the acquired data is reliable and accurate, which is essential for making informed decisions in fault diagnosis of bearing. This device comes with user-friendly software tools such as LabVIEW which allows the user to quickly and easily setup acquisition, acquire data and perform various types of analysis and visualization. myDAQ is a cost effective solution for vibration data acquisition which can be used with wide range of sensors including ADXL 335 accelerometer. Its features and benefits make it an ideal choice for industrial settings where monitoring and analysis of machine health are essential for optimal performance and efficiency.

Figure 1. 7: NI DAQ System

- **Bearings**

Four 608 deep groove ball bearings were used for the vibration data acquisition out of which one was a normal bearing, and three different faults namely inner race fault, outer race fault and ball fault were introduced in the remaining three. The 608-ball bearing is a standard deep groove ball bearing with an inner diameter (ID) of 8 mm, an outer diameter (OD) of 22 mm, and a width (W) of 7 mm. The bearing has a pitch (P) of 8.1 mm.

- **Computer System/PC**

  A PC computer equipped with Processor Intel® Core™ i7-8550 U CPU @ 1.80GHz, 1992 MHz, 4 Core(s), 8 Logical Processor(s), RAM 8GB & HDD 20GB was used for computation.

- **Tachometer**

Figure 1. 8: Tachometer

A handheld digital photo tachometer was used to measure the rotational speed of the AC induction motor. The device uses a laser beam to detect the speed of the motor and display it on its LED screen. The tachometer typically comes with reflective tape that is attached to the shaft of the motor. The reflective tape reflects the laser beam back to the tachometer, allowing for an accurate measurement of speed.

Chapter 2 : LITERATURE REVIEW

Among the various elements of a motor i.e., stator, rotor, shaft and bearing, faults in bearings accounts for 42% - 50% of unscheduled maintenance [17]. Improper installation, handling fault, contaminants, wear and tear, lubricating factors and operating environmental condition cause different faults. Based on the specific component at fault, bearing defects can be classified as outer race fault, inner race fault, roller element fault, etc.

Condition Based Maintenance (CBM) is a predictive maintenance approach which optimizes maintenance strategies to improve efficiency, safety, reliability and expenses. In CBM, extracted data from past and present are analyzed to predict future consequences and prevent unnecessary maintenance and unforeseen failures [18]. Condition based maintenance was first used by Rio Grande Railway Company in 1940s. The condition of coolant, oil and fuel leaks were predicted by the measurement of change in pressure and temperature. In the recent years, analysis of vibration data for fault feature extraction and fault diagnosis has been catching speed.

Fault diagnosis is the process of discovering fault in a component by comparing the data of its normal operating condition to its present condition and extracting fault features [19].

Bearing fault diagnosis has been mainly performed through three following methods:

- Conventional model using motor current and principal component analysis [20].
- Machine learning using algorithms such as support vector machine, K-nearest neighbor [21].
- Deep learning with artificial neural networks (auto-encoder, deep belief, generative adversarial, convolutional neural and recurrent neural networks) [22].

Signal processing consists of time, frequency and time frequency domain analysis. Out of many signal processing methods STFT was found to be a better fit to deep learning algorithm input in fault detection [4]. Among many other machine and deep learning algorithms, SVM, LSTM and CNN seem to have an easier implementation and produce results with a higher accuracy [4]. Support Vector Machine (SVM) is a supervised learning model with strong generalization capacity. It uses structural risk minimization concept to train the classifiers which classify the data into positive and negative classes. [23] LSTM works on the basis of memory block function where gates and memory cells are multiplied to keep the continuous flow of information, mainly self-recurrent memory cells keeping the long-term memory error flows being truncated when required. LSTM is among the best cost-effective recurrent neural network used for monitoring large dataset having large accuracy, short time interval for computation with resistance to over fitting [24]. CNN is a first-rate deep learning method having a feedforward neural network with major layers as convolutional, pooling and connected. It is especially useful to solve problems related to overfitting and has a high accuracy when used with preprocessed data [25].

Chapter 3 : RELATED THEORY

3.1 Support Vector Machine (SVM)

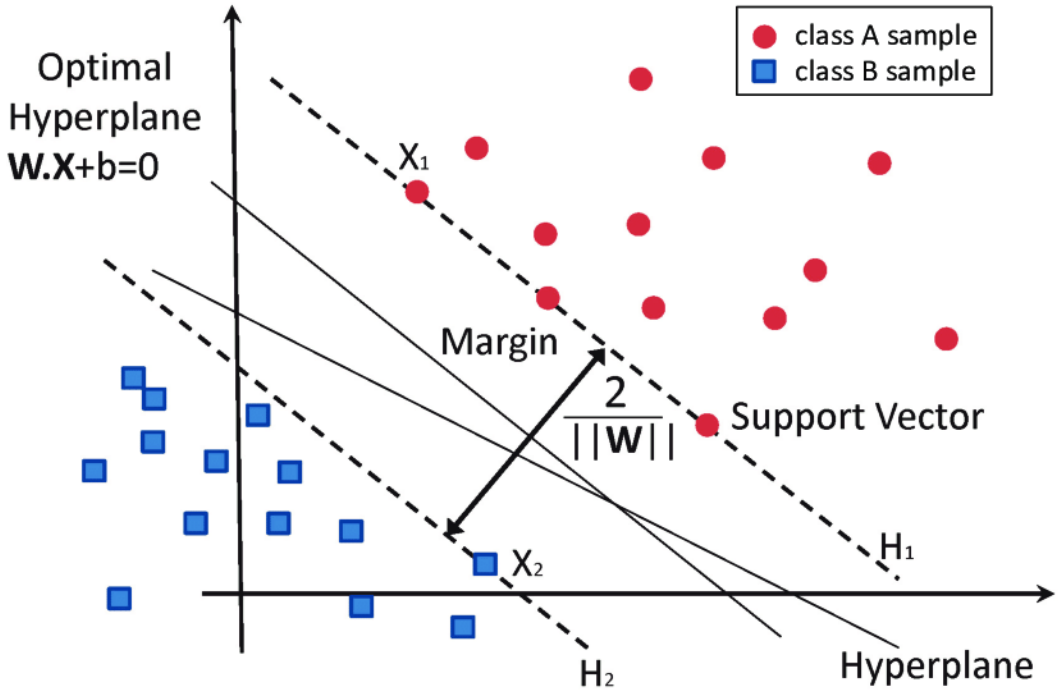

Figure 3. 1: Support Vectors and the Optimal Hyperplane

SVM is an ML algorithm that provides a linear model for classification and regression analysis. It is a supervised learning model. SVM creates a best-decision boundary (hyperplane) in an n-dimensional space for category separation thus helping with distinct classification. The extreme vectors of a category help to define the hyperplane and are called Support Vectors.

For a given dataset,

$$D = \{(x_i, y_i) | x_i \in R^n, y_i \in (-1, 1)\}_{i=1}^{m} \quad 3.1$$

Where x_i is the input vector and y_i is the label of x_i, the hyperplane is given by

$$w \cdot x + b = 0 \qquad 3.2$$

Here, $w = (a, -1)$ if the hyperplane is defined as $y = ax + b$.

To calculate the point closest to the hyperplane we define the parameter β which is calculated as

$$\beta = |wx \cdot + b| \qquad 3.3$$

For our required domain D, we get,

$$B = min\_(i = 1, n) \ |wx + b| \qquad 3.4$$

Where B is the smallest value of β. If there are s hyperplanes present, each will have a specific B_i value, and the hyperplane with the largest B_i value is selected.

3.1.1 Principal Component Analysis (PCA)

It is a widely used statistical method used in reducing dimensionality, in data analysis. It transforms a set of correlated variables into a new set of uncorrelated variables, known as principal components which captures maximum amount of variance in the given dataset.

PCA finds a set of orthogonal axes in the multidimensional space of original variables, along which data varies most. The 1st principal component captures maximum amount of variance in the data, 2nd captures remaining variance orthogonal to first components and so on for other components. The principal components are computed using eigenvalue decomposition of the covariance matrix of the original data or by singular value decomposition of data matrix. Once the principal

components are computed, they can be used to reduce dimensionality of the data as it selects a subset of the principal component which captures most of the variance in the data. PCA is applicable in:

- Data (Analysis, Compression, Feature Selection and Visualization),
- Outliners detection,
- Identification of patterns
- Noise removal from data

The limitation of PCA is that it assumes data are linearly related and variance is evenly distributed along each axis also, it is sensitive to outliers, scaling and normalization issues in the provided data.

3.1.2 Parameter Selection

Parameter selection is an important process in machine learning that involves selecting the optimal set of hyperparameters for a given model. Hyperparameters are values that are set before training and are not learned from the data. In SVM, several hyperparameters are used, including the kernel, C, gamma, and decision function shape.

- 'Kernel' specifies the type of kernel used in the SVM algorithm, and in the case of SVM, three different kernels are used: radial basis function (rbf), polynomial (poly), and sigmoid.
- 'C' is the regularization parameter that determines the trade-off between achieving low training error and low testing error due to overfitting, and two values of C can be specified, namely 0.01 and 1.
- 'Gamma' is used for non-linear hyperplanes, with higher gamma values defining a higher influence of training examples, which can result in overfitting. Two values of gamma can be specified, namely 0.01 and 1.
- 'Decision Function Shape' parameter specifies the decision function shape for multi-class problems, with ovo representing one-vs-one.

To select the best combination of hyperparameters for a given model, there are hyperparameter tuning methods such as GridSearchCV and RandomizedSearchCV. These methods search through the hyperparameters space to identify the optimal set of hyperparameters that result in the highest accuracy on the dataset.

3.1.3 Feature Extraction

The Diagnostic Feature Designer toolkit was used to extract the features from the time-domain vibration signal. The following values were used based on their properties to extract the features:

- Max Value: It is the highest value in a set of data points used to identify the upper bound of the range of values in the dataset.

$$max\ value = max(x_1, x_2, \dots, x_n) \quad 3.5$$

- Min Value: It is the lowest value in a set of data points and is used to identify the lower bound in a given range of values of a dataset.

$$min\ value = min(x_1, x_2, \dots, x_n) \quad 3.6$$

- Mean Value: It is the average value of a set of data points and measures their central tendency.

$$mean\ value(x_m) = \frac{x_1, x_2, \dots, x_n}{n} \quad 3.7$$

- Variance: It measures the spread of the data points around the mean value.

$$variance(x_v) = \frac{[(x_1 - x_m)^2 + (x_2 - x_m)^2 + \ldots + (x_n - x_m)^2]}{n} \quad 3.8$$

- Standard Deviation: It is given by the square root of variance and provides a measure of the spread of the data points in the same units as the original measurements.

$$standard\ deviation(x_s) = (x_v)^{\frac{1}{2}} \quad 3.9$$

- RMS Value: It is a measure of the average magnitude of the data points and helps to understand the overall magnitude of the data.

$$rms\ value(x_{rms}) = \left[\frac{(x_1^2 + x_2^2 + \ldots + x_n^2)}{n}\right]^{\frac{1}{2}} \quad 3.10$$

- Skewness: It is the measure of asymmetry of the data points. It represents the shape of the distribution of data points.

$$skew\ value = \frac{[(x_1 - x_m)^3 + (x_2 - x_m)^3 + \ldots + (x_n - x_m)^3]}{[(n - 1) * x_s^3]} \quad 3.11$$

- Crest Value: It is the ratio of the maximum value to the rms value. It represents the peakness of the signal.

$$crest\ value(x\_crest) = \frac{x_{max}}{x_{rms}} \qquad 3.12$$

Kurtosis Value: It is the measure of tailedness of the data points. It shows how clustered the data points are to the mean value.

$$kurtosis(x_k) = \frac{[(x_1 - x_m)^4 + (x_2 - x_m)^4 + \ldots + (x_n - x_m)^4]}{[(n-1) * x_s^4] - 3} \qquad 3.13$$

Here, x_1, x_2,…,x_n are the data points and n represents the number of data points.

3.2 Convolutional Neural Network (CNN)

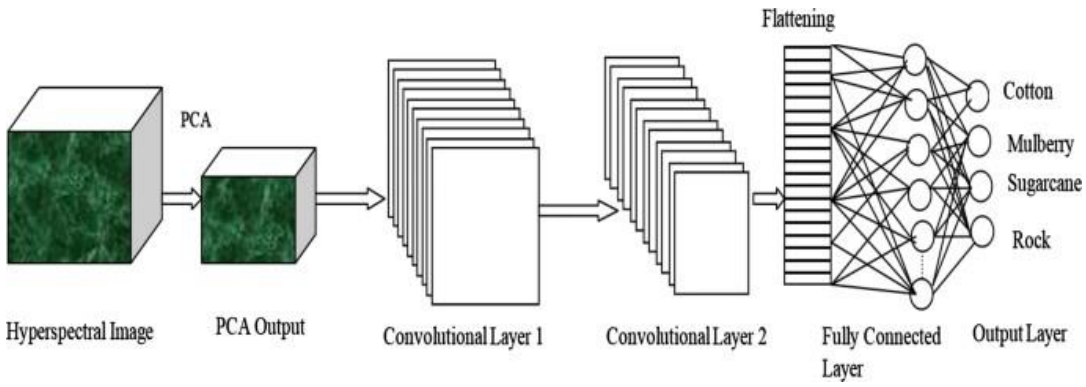

Figure 3. 2: Convolutional Neural Network

Convolutional Neural Network is an artificial neural network popularly used for analyzing images. CNN can also be used for signal analysis and data classification.

The architecture of CNN consists of an input layer, no of hidden layers and an output layer. The activation function masks the inputs and outputs of the middle layers due of which the middle layers are called hidden layers. The convolution is performed by the hidden layers. A dot product is performed between the input matrix of the layer and the convolution kernel. As a result, a feature map is generated as the convolution kernel slides along the input matrix. The feature map is the input for the next layer of CNN.

- Input layer

  Input layer is the first layer of CNN which receives the raw one-dimensional vibrational data. In case of 2D CNN the 1D vibration data is converted into a 2D representation that can be processed by the convolutional layers of the 2D CNN network. The 1D vibration data was used to generate spectrogram which is a 2D representation of the vibration signal. Convolutional layer performs the operation of convolution, activation and pooling on the input data. During the process of convolution different filters are employed to extract the features from the vibration data. The activation function introduces non-linearity in the CNN model

and the pooling process reduces the spatial dimensionality and improve the efficiency of the model.

- Kernel size

  In CNN model, the kernel size is the dimension of filter or window that slides over the input data or feature map. During the convolution operation, the kernel moves over the input data and at each location, it computes the dot product between its own values and the corresponding values in the input. This operation produces a single output value that becomes a part of the output feature map. The choice of kernel size depends upon the complexity of the input data and the desired level of feature extraction. A smaller kernel size captures fine-grained details in the input data, while a larger kernel size can capture more global features.

- Stride

  In CNN model, stride refers to the number of pixels the convolution kernel moves each time it is applied to the input data. The stride length determines the amount of overlap between adjacent regions of the input data that are covered by the kernel. For example, a stride of 1 means that the kernel moves one pixel at a time, resulting in adjacent regions of the input data overlapping by one pixel. Using a larger stride can reduce the size of the output feature map which may be useful in downsampling the input data and reducing overfitting. However, use of larger stride can result in the loss of important information from the input data.

- Activation functions

  Activation functions are the component of CNNs that introduce non-linearity into the network and help to capture complex patterns in the input data. They allow CNNs to model complex relationships between the input features and output classes and help to improve the accuracy. Some commonly used activation functions in CNN are:

1. ReLU (Rectified Linear Unit): ReLU activation function is defined as f(x) = max (0,x), which means that it gives the output if the input is positive and sets negative input to zero. ReLU function is computationally efficient and helps to overcome the vanishing gradient problem.
2. Tanh (Hyperbolic tangent): The hyperbolic tangent function is defined as f(x) = (exp(x) - exp(-x)) / (exp(x) + exp(-x)). It maps the input values to a range of -1 to 1.
3. Leaky ReLU: The Leaky ReLU is defined as f(x) = max(ax,x), where a is a small positive constant. This function allows a small gradient for negative inputs.

- Pooling

Pooling is an operation in CNN models to reduce the spatial dimensions of feature maps of convolutional layers. Most common types of pooling operation are:

1. Max pooling: This involves partitioning the feature map into non-overlapping rectangular regions (also called pooling regions) and taking the maximum value within each region
2. Average pooling: This operation takes the average value within each pooling region.

By the process of pooling, computational requirements were reduced and as a result the efficiency of the model is improved. The problem of overfitting is also prevented.

- Optimizer function

Optimizer function is a component of CNNs which updates the weights of the network during network to minimize the loss function. Selection of optimizer function has a significant impact on the training process and the performance of the model. Some of the popular optimizer functions used in CNN are:

1. Adam: Adam optimizer adjust the learning rate dynamically which results in the increasing convergence speed of the training and improving overall performance

of the CNN model as well. During the training process the learning rate is substantial for parameters with small gradients and learning rate is adjusted to smaller in case of larger gradient parameters.

2. Stochastic Gradient Descent (SGD): SGD updates the weights by taking the gradient of the loss function with respect to the weights and moving in the opposite direction of the gradient.
3. Adagrad: This optimizer function adapts the learning rate of each weight based on the historical gradient information.

- Flatten layer

Flatten layer is used in CNN model for fault diagnosis after final convolutional layer and before dense layer. The purpose behind the use of flatten layer is to transform the output of the convolutional layer into a form that can be easily fed into a fully connected layer. By flattening the output, the network can learn more complex features that capture higher level information about the vibration data. Flatten layer maintains balance between complexity and efficiency of network architecture during the reduction of dimensionality.

- Dense layer

A dense layer or fully connected layer is used in CNN model for extracting high level features and making final classification decision. The dense layer takes the flattened feature vector as input and applies a linear transformation followed by activation function to produce a new set of features. The output of the dense layer is then passed to the final layer for classification.

- Final layer

In a CNN for fault diagnosis of bearings, final layer is a fully connected layer that outputs a vector of probabilities corresponding to the different fault types of the bearing. The final layer typically consists of multiple neurons, with each neuron corresponding to a specific fault type. Some of the common activation functions used in the final layer are:

1. Sigmoid: The sigmoid function is defined as f(x) = 1/(1+exp(-x)). It maps the input values to a range of 0 to 1. Sigmoid is often used in the output layer of a CNN to predict binary outcomes.
2. Softmax: Softmax activation function is used in case of the output is multiclass. It transforms the output of the last convolutional layer into a probability distribution over different fault category.

3.2.1 K- Fold Cross- Validation

K-fold cross validation is a machine learning technique used to enhance the performance of different training models. In this particular technique the complete dataset is split into smaller k sets or folds and the k-1 folds are used for training purposes. The resulting model is used as a test set to compute a performance measure i.e., accuracy. Then the performance measured is the average of values computed in the loop. It is however computationally expensive yet doesn't waste data giving advantage to dataset having number of samples very small.

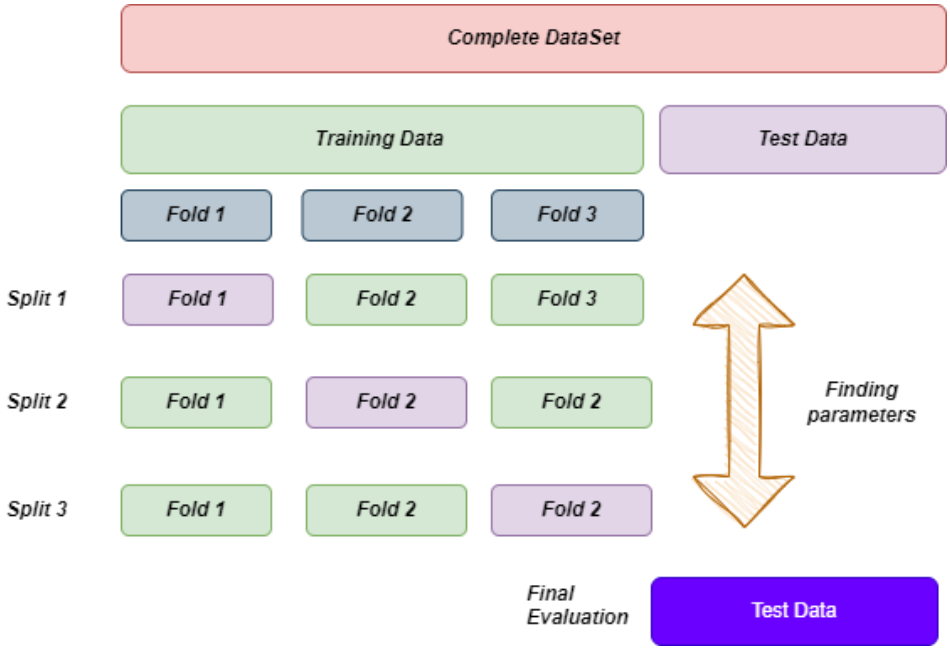

Figure 3. 3: K-fold Cross Validation

3.3 Long-Short Term Memory

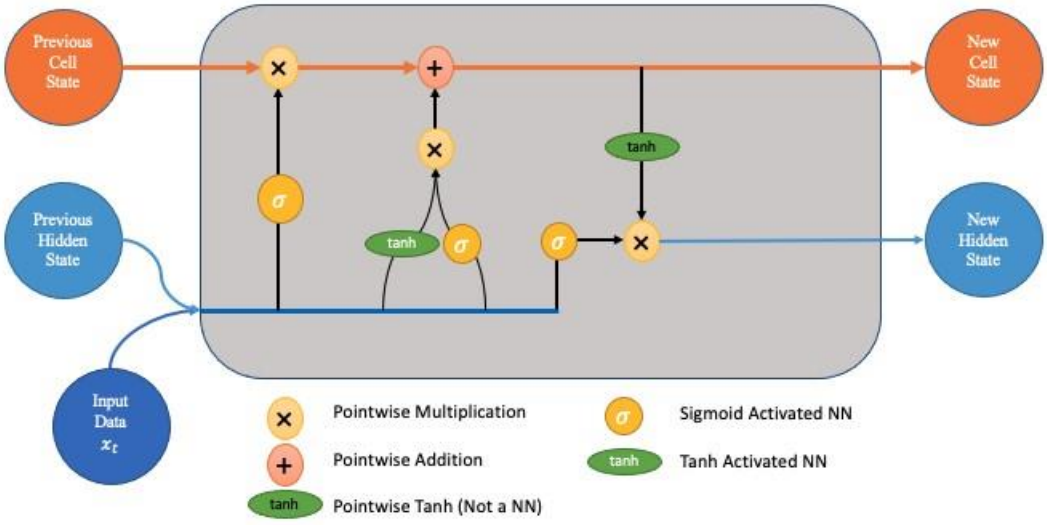

Figure 3. 4: LSTM Architechture

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) that utilizes a gate-oriented mechanism to control cells, taking input from previous stages as well as the current stage. It has been extensively used in speech recognition, text recognition, natural language processing, and fault detection and diagnosis programs. LSTM's unique feature is its use of multiplication of gates and memory cells to maintain a long-term flow of information in a sequence. These gates perform different functions, such as the input gate x(t), forget gate f(t), output gate o(t), input modulation gate g(t), and memory cell c(t). Additional neural parameters, including hidden state h(t), current state x(t), previous hidden state h(t-1), and output labels y(t), are used accordingly. Unlike short term memory which can't remember the important function in a long chain of network, LSTM overcomes it by introducing a memory block. LSTM is designed to overcome the limitations of short-term memory by introducing a memory block that can remember the essential functions in a long chain of the network. The equations used for LSTM are complex, but they enable the network to maintain context and effectively process sequential data. The equations for LSTM is given as follows:

LSTM equations:

- Input Gate

$$I_t = \sigma (W_i h_{t-1} + U_i x_t + b_i) \quad 3.14$$

- Forget Gate

$$F_t = \sigma (W_f h_{t-1} + U_f x_t + b_f) \quad 3.15$$

- Output Gate

$$O_t = \sigma (W_o h_{t-1} + U_o x_t + b_o) \quad 3.16$$

- Memory Cell Candidate

$$C_t = tanh (W_o h_{t-1} + U x_t + b) \quad 3.17$$

- Memory Cell

$$C_t = f_t * c_t - 1 + i_t * t \quad 3.18$$

- Shadow State

$$h_t = o_t * tanh(c_t) \quad 3.19$$

- Cell Output

$$y_t = h_t \quad 3.20$$

Here, W_o, W_i and W_f are the weights of output, input and forget gates.

3.4 Fast Fourier Transform (FFT)

The Fast Fourier Transform (FFT) algorithm is a mathematical algorithm that computes the Discrete Fourier transform (DFT) of a sequence of data points. The DFT is a transformation that converts a time-domain signal into its frequency-domain representation. The formula for the DFT is as follows:

$$X[k] = \sum_{n=0}^{N-1} x[n] * exp(-2\pi jkn/N) \quad 3.21$$

;where X[k] is the kth frequency component of the signal, x[n] is the nth time-domain data point, j is the imaginary unit, and N is the number of data points in the signal.

The FFT algorithm is a fast algorithm for computing the DFT and is based on a divide-and-conquer approach that recursively splits the sequence into smaller sub-sequences. The formula for the FFT algorithm is derived from the DFT formula and is as follows:

$$X[k] = \sum_{n=0}^{N-1} x[n] * W_N^{nk} \quad 3.22$$

Here W_N = exp(-2πj/N) is the Nth root of unity and k is an integer between 0 and N-1. The FFT algorithm calculates the DFT using a series of butterfly operations that combine pairs of sub-sequences to generate the frequency components of the signal.

The frequency domain signal obtained from FFT can be used to analyze the frequency content to identify important features and filter out unwanted noise. FFT is a crucial step in the preprocessing of vibration data for machine learning and deep learning algorithms. By computing the FFT of the vibration data, it is possible to obtain a more detailed understanding of the frequency content of the signal, which can then be used to extract relevant features. These features can be used as input to machine learning and deep learning algorithms to improve the accuracy and effectiveness of fault diagnosis.

Chapter 4 : METHODOLOGY

4.1 Flowchart for the BE Project

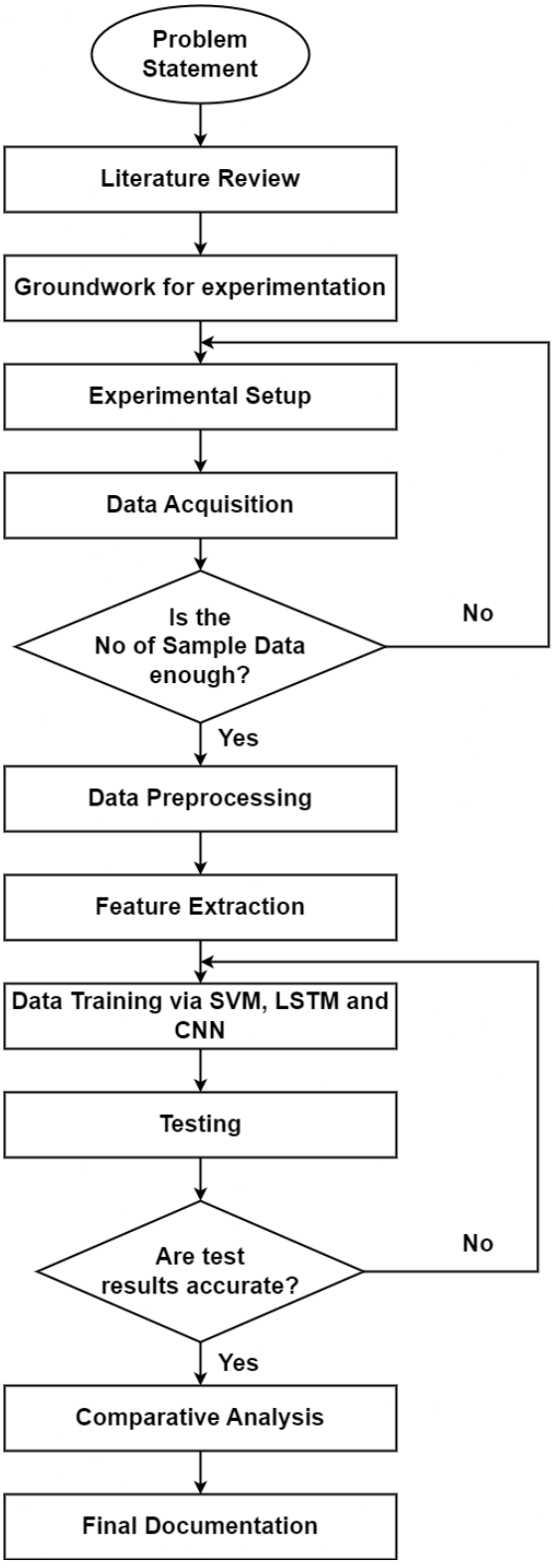

Figure 4. 1: Flowchart for the BE Project

4.2 Flowchart for the CWRU Dataset Fault Diagnosis

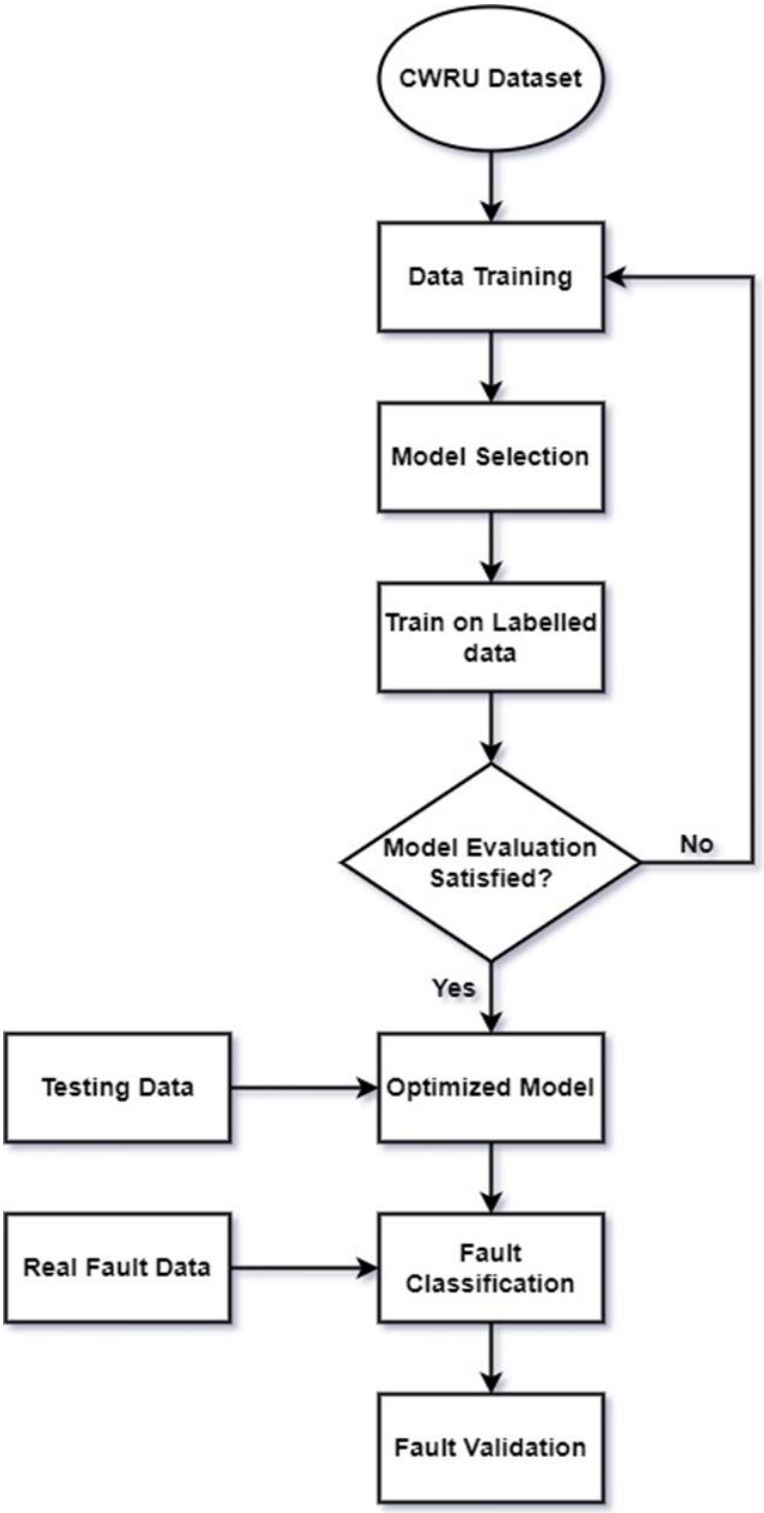

Figure 4. 2: Flowchart for CWRU Fault Diagnosis

4.3 Fault Introduction

Three 608 bearings were taken and faults were introduced to the inner race, outer race and the ball component of the respective bearings. For outer raceway fault introduction, a 1.5mm was made with a grinder till the ball cage was reached. For ball fault, out of the 7 balls in the 608 bearing, one was taken out. The bearing was dissembled and in the inner raceway, the grinder was used to make a 1.5mm line as a part of the inner raceway fault introduction.

Figure 4. 3: Inner Raceway Fault

Figure 4. 4: Ball Fault

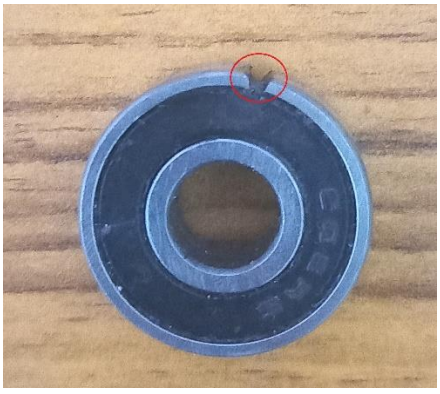

Figure 4. 5: Outer Raceway Fault

4.4 Experimental Setup

The following steps were taken to build the experimental setup:

- First the ac induction motor is setup in a stand and fixed firmly,
- ADXL 335 accelerometer is mounted on the top of the bearing housing at the drive end of motor.
- The accelerometer is then connected to DAQ system using jumper wires.
- The DAQ system is then connected to working computer via USB ports.
- LabVIEW is setup to acquire the vibration data. In this process we create a new virtual environment and necessary DAQ functions are added to the block diagram.
- Next, LabVIEW is configured to measure the vibration signals which involved setting up the sampling rate, selecting the input channels.
- The 2-phase ac induction motor is started by giving ac voltage through electric circuit.
- The vibration data with a sampling frequency of 1000 Hz was acquired. For each bearing condition, 2 minutes of data was acquired which consisted of 120,000 data points. The acceleration values for X and Y direction, the two perpendicular radial directions were acquired.
- Once the data for normal condition is taken, the drive end bearing is replaced with a faulty bearing. The process is then repeated for all faults & their respective data is acquired.

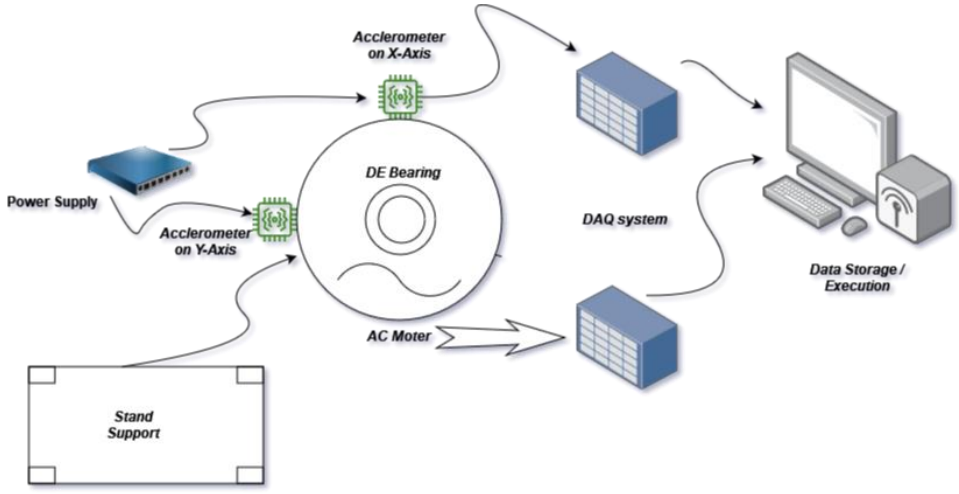

Figure 4. 6: Experimental Setup for the AC Induction Motor

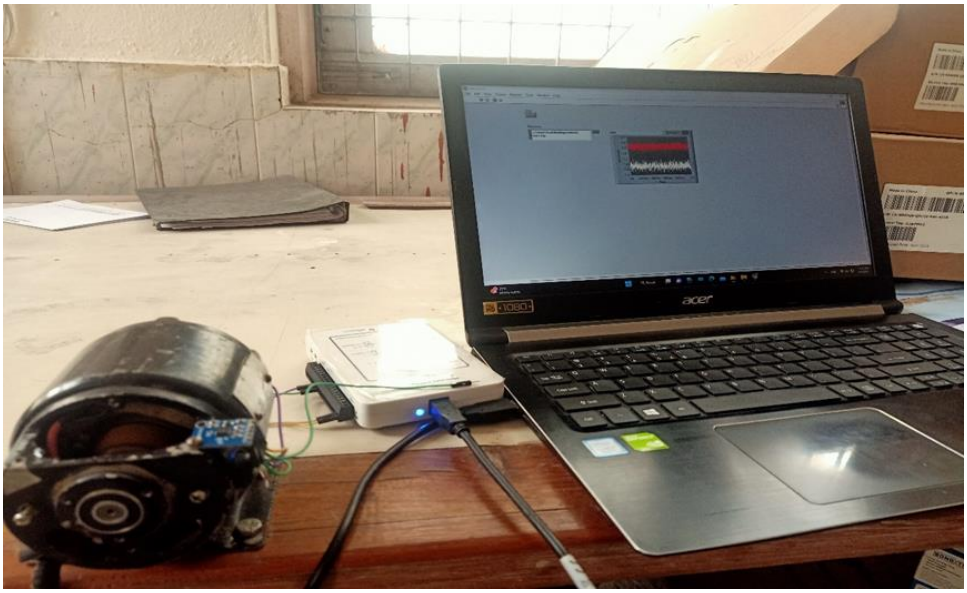

Figure 4. 7: AC induction motor, DAQ system and ADXL 335 accelerometer

4.5 Algorithm Development Models

4.5.1 SVM Architecture

The SVM architecture used in the SVM model imports new data containing features extracted from the experimental dataset. The features applied are an RBF kernel SVM model, GridSearchCV with cross-validation, and test set evaluation. Preprocessing involves standardization and dimensionality reduction using PCA. The optimized hyperparameters include C, gamma, and the decision function shape. The following table summarizes the key components and hyperparameters of the support vector machine (SVM) architecture used in our SVM model for fault diagnosis.

Table 4. 1: SVM Model Parameters

Step	Technical Term	Value
1.	Feature Extraction	Standardization (StandardScaler)
		Dimensionality Reduction (PCA)
2.	SVM Model	Radial Basis Function (RBF) Kernel
		Hyperparameters:
		C = 0.01, 1
		Gamma = 0.01, 1
		Decision function shape = One vs. One (ovo)
3.	Training	GridSearchCV
		Cross-validation (cv) = 5
4.	Evaluation	Test accuracy = 81.1515%

4.5.2 1D CNN Architecture

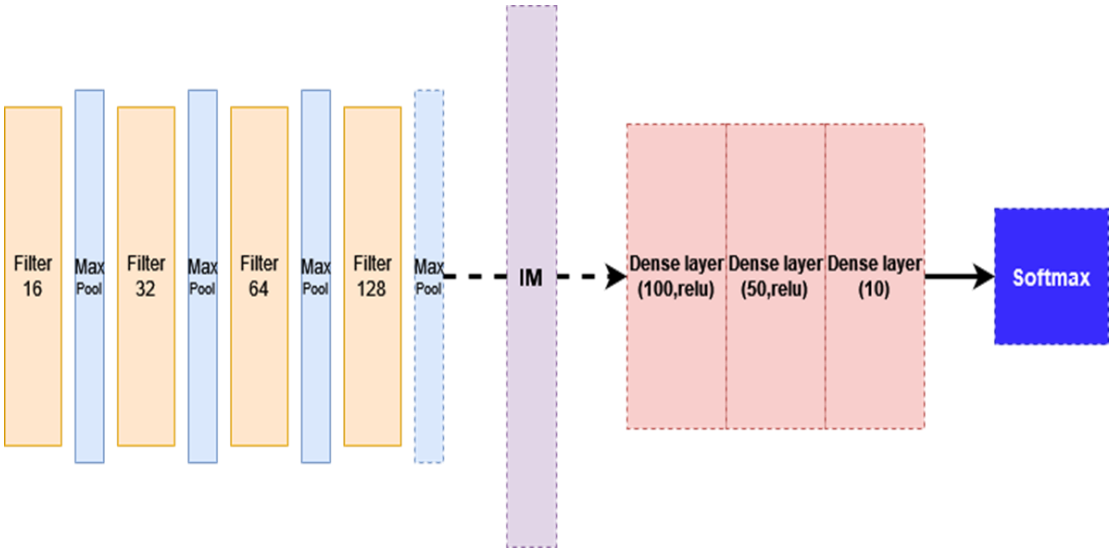

Figure 15: 1D CNN architecture with Activation Functions

The 1-D CNN model is mainly used for image and time-series training. Similarly, the data fed into the model has to be of an acceptable size i.e., Enough to train but not large enough to be overfitting. The 1-D CNN with its simplified characteristics has short computation duration with higher accuracy compared to other models.

In this 1-D CNN model it consists of shape of data from Input Data as [5486,1936] with Y-CNN as [5486,10] and Y as [5486,1]. The data is feed using k-fold cross validation with 5 splits and 101 random states. The data is then taken in a reshaped form X (-1,1936,1) as 1-D input. The model is split into (X, Y) dimensions for training and testing using train_test_split from sklearn library. The data is split into 70% training and 30% testing data.

Table 4. 2: 1D CNN Model Parameters

S. N	Layers	Filters	Kernal size	Strides	Activation	Pool size
1	Convolution 1D	16	3	2	relu	
2	MaxPool 1D					2
3	Convolution 1D	32	3	2	relu	
4	MaxPool 1D					2
5	Convolution 1D	64	3	2	relu	
6	MaxPool 1D					2
7	Convolution 1D	128	3	2	relu	
8	MaxPool 1D					2
9	Flatten					
10	InputLayer					
11	Dense (100)				relu	
12	Dense (50)				relu	
13	Dense (10)					
14	Softmax					

Here, the model consists of 14 different layers. Finally, the model is compiled using ‘adam ‘optimizer with defined terms for ‘loss’ and ‘metrics.

The model runs with 1 verbose and 15 epochs each round. After the model is computed it then gives out the accuracy for training and testing with their respective confusion matrices. As a safety measure the program was added with the feature to test for overfitting by comparison between test and train scores and plotting for validation and training curve.

4.5.3 2D CNN Architecture

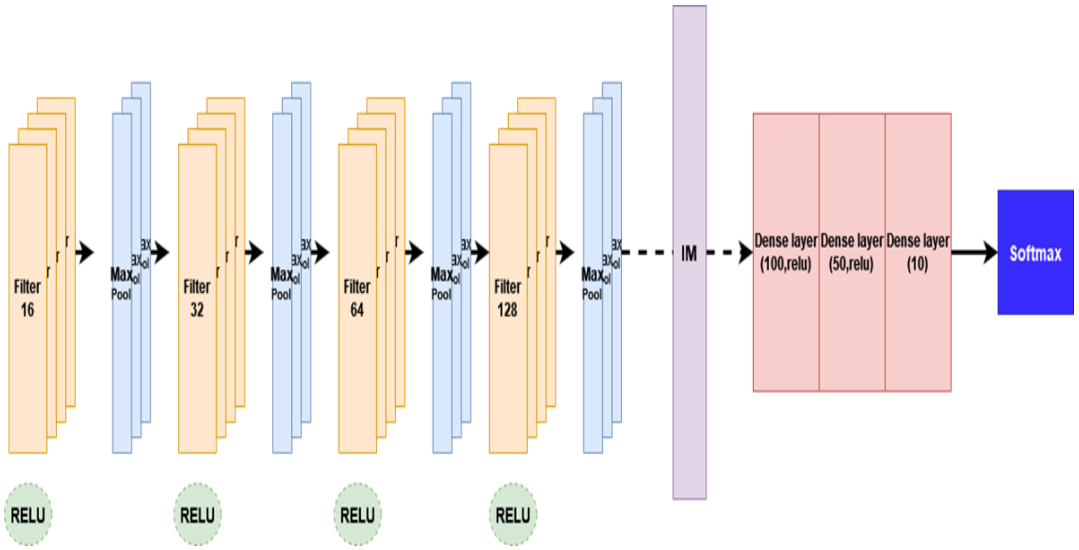

Figure 16: 2D CNN Architecture with Activation Functions

The 2-D CNN model is also mainly used for image and time-series training. Similarly, to 1D the data fed into the model has to be of an acceptable size i.e., Enough to train but not large enough to be overfitting. The 2-D CNN is a bit complex than the 1-D CNN, so it takes more computation time and has more loss.

In this 2-D CNN model it consists of shape of data from Input Data as [5486,1936] with Y-CNN as [5486,10] and Y as [5486,1]. The data is fed using k-fold cross validation with 5 splits and 101 random states. The data is then taken in a reshaped form X (-1,44,44,1) as 2-D input or square-root of data feed to 1-D CNN. The model is split into (X, Y) dimensions for training and testing using train_test_split from sklearn library. The data is split into 70% training and 30% testing data.

Table 4. 3: 2D CNN Model Parameters

S. N	Layers	Filters	Kernal size	Strides	Activation	Pool size	Padding
1	Convolution 1D	16	3,3	2,2	relu		Same
2	MaxPool 1D					2,2	Same
3	Convolution 1D	32	3,3	2,2	relu		Same
4	MaxPool 1D					2,2	Same
5	Convolution 1D	64	3,3	2,2	relu		Same
6	MaxPool 1D					2,2	Same
7	Convolution 1D	128	3,3	2,2	relu		Same
8	MaxPool 1D					2,2	same
9	Flatten						
10	Input Layer						
11	Dense (100)				relu		
12	Dense (50)				relu		
13	Dense (10)						
14	Softmax						

Finally, like in 1D CNN the model is compiled using ‘adam ‘optimizer with defined terms for ‘loss’ and ‘metrics.

The model runs with 1 verbose and 15 epochs and ‘true’ multiprocessing in each round. After the model is computed it then gives out the accuracy for training and testing with their respective confusion matrices. As a safety measure the program was added with the feature to test for overfitting by comparison between test and train scores and plotting for validation and training curve.

4.5.4 LSTM Architecture

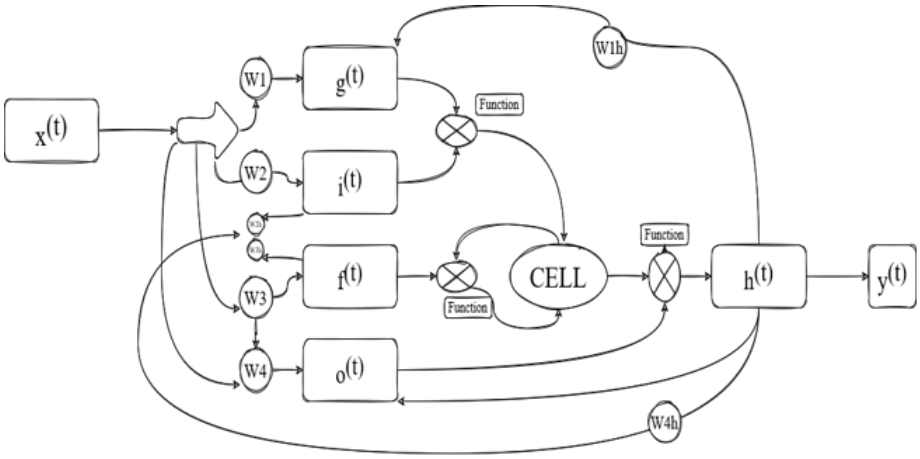

Figure 17: LSTM Architecture

The LSTM model is fed with one-dimensional Input Data as [5422, 5184], shape of Y_CNN as [5422,10] and shape of Y as [5422,1]. The data is shaped into 1D form [-1,5184,1]. The training and testing data is split in (X, Y) dimensions in 70% and 30% respectively. The LSTM function is inbuilt Python (Jupiter) for Visual Code Studio. For layering the same function has been called which consists of activation function of tanh with recurrent activation function hard-sigmoid, kernel initializer as 'glorot uniform', recurrent initializer 'orthogonal' & bias initializer as zeros. Intermediate layers are added which consist of flatten layer, Dense layer of batch size 10. The final layer consists of a SoftMax activation function.[26] The model is optimized with 'adam' set at a learning rate of 0.001 in Tensorflow and a 0-decay rate in Keras.

- Activation function

  In our LSTM model for fault diagnosis tanh was used as a activation function. Tanh maps input values to a range between -1 and 1 which can be useful for ensuring that the output values of neurons are bounded. The sigmoid shape of tanh function is useful in capturing the non-linearities in the data which made our model capable of learning complex patters and relationships between the input data and output.

- Recurrent activation function

  Hard sigmoid activation function was used as recurrent activation function in our LSTM network for fault diagnosis. This activation function maps input to a range between 0 and 1 with a linear activation around 0 which make LSTM network to easily learn and capture non-linear relationships in the vibration data.

- Kernel initializer

  Glorot uniform initializer was used as a kernel initializer in our LSTM network for fault diagnosis. In an LSTM network for analyzing vibration data, the Glorot uniform initializer helped to ensure that the initial weights of the network are optimized for learning the patterns and trends in the data. This improved the performance of the network and made it more efficient in terms of training time and computational resources.

- Recurrent initializer

  The orthogonal recurrent initializer which was used in our LSTM network is a weight initialization method that sets the recurrent weight matrix of an LSTM network to an orthogonal matrix. In the context of analyzing vibration data, orthogonal recurrent initializer is useful in capturing long-term dependencies without losing important information during training process.

- Bias initializer

  Bias initializer of zeros was used in LSTM network for fault diagnosis of bearing. In a neural network, biases are used to shift the activation function to the left or right, which can be useful for improving the accuracy of the model. In the context of fault diagnosis in vibration data, using a bias initializer of zeros can help to ensure that the initial predictions of the LSTM network are unbiased and based solely on the input data. This can be important for accurately detecting faults and anomalies in the data as biases can potentially mask the important information.[27]

Table 4. 4: : Layers, Return Sequence and Number of Neurons of LSTM

S.N	Layers	Return Sequence	No. of Neurons
1	LSTM	True	32
2	Flatten		
3	Dense		10
4	Softmax		
5	Adam Optimizer		

Chapter 5 : RESULTS AND DISCUSSION

5.1 Bearing Fundamental Defect Frequencies (BFDF) Calculation

To calculate the fundamental bearing defect frequencies for a 608 deep groove ball bearing, we need to know the geometry of the bearing and the number of rolling elements (balls) in the bearing. The 608 bearing has a bore diameter of 8 mm, an outer diameter of 22 mm, and a width of 7 mm. It has 7 balls.

The fundamental bearing defect frequencies for a 608 bearing can be calculated using the following equations:

Inner race defect frequency (BPFI) = (d/2) * (N/10^6)

Outer race defect frequency (BPFO) = (D/2) * (N/10^6)

Ball spin frequency (BSF) = (d/2) * ((1 - d/D) / (1 + d/D)) * (N/10^6)

;where d is the ball diameter, D is the pitch diameter, N is the bearing speed in revolutions per minute (rpm), and BPFI, BPFO, and BSF are in Hertz (Hz).

For a 608 bearing with a ball diameter of 4.763 mm and a pitch diameter of 18.66 mm (assuming a standard 0.5 mm radial clearance), the fundamental defect frequencies are:

BPFI = (8/2) * (10000/10^6) = 0.04 kHz = 40 Hz

BPFO = (22/2) * (10000/10^6) = 0.11 kHz = 110 Hz

BSF = (4.763/2) * ((1 - 4.763/18.66) / (1 + 4.763/18.66)) * (10000/10^6) = 0.98 kHz = 980 Hz

Therefore, the fundamental bearing defect frequencies for inner race, outer race, and ball fault in a 608 deep groove ball bearing with an rpm of 10000 are 40 Hz, 110 Hz, and 980 Hz, respectively.

5.2 Signal Processing

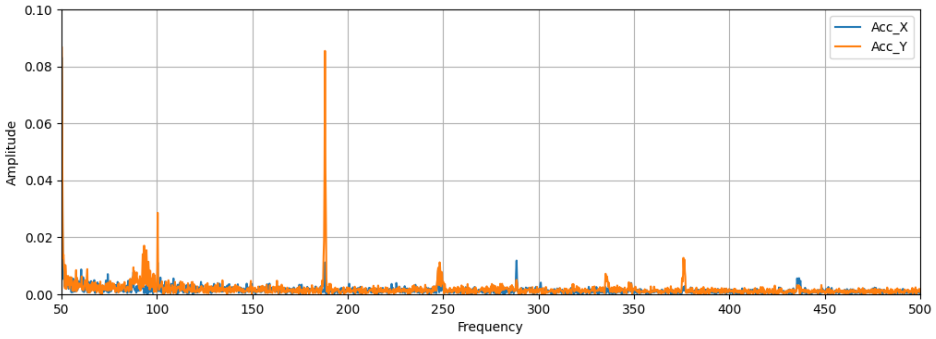

Figure 5. 1: FFT of the Normal Bearing Vibration Data

Performing Fast Fourier Transform (FFT) on the acquired time-series vibration data shows that there are distinctly noticeable peaks of frequencies representing the rpm in which the motor rotates and the specific bearing faults that have been introduced. The normal data shows only one distinct frequency peak in 187Hz representing the fundamental frequency in which the motor rotates. The motor rpm can be now calculated as:

187*60 = 11,220 rpm

During the experiment, for the normal bearing data acquisition, the rpm was held between 11,000rpm to 11,500rpm, thus the distinct peak observed at the normal plot is validated. For inner defect, outer defect and ball defect data, the motor rpm was held at the range of 9500rpm to 10,500 rpm.

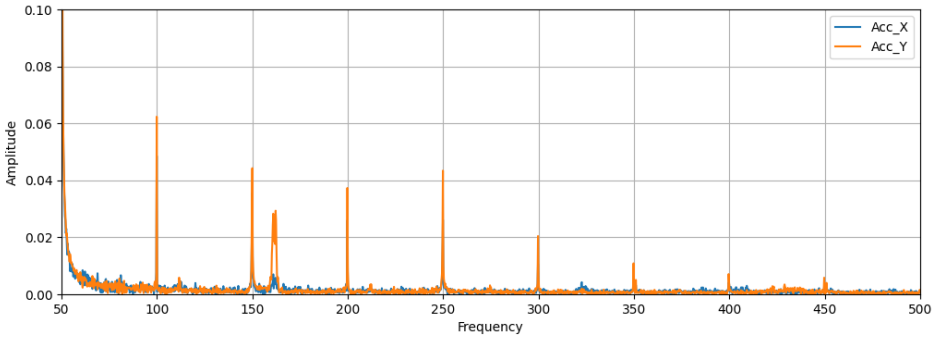

Figure 5. 2: FFT of Inner Race Defect

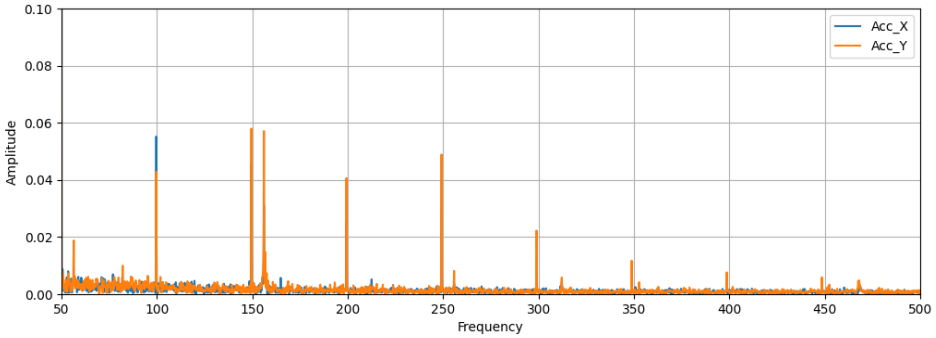

Figure 5. 3: FFT of Outer Race Defect

For the inner race and outer race defect, the bearing fundamental defect frequencies and their harmonics can be distinctly observed.

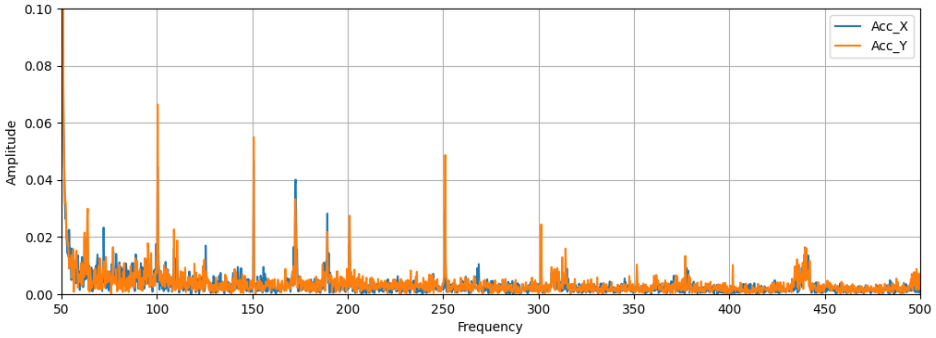

Figure 5. 4: FFT of Ball Defect

For ball fault, the FFT plot wasn't able to show the distinct fault frequency. As the ball spin frequency (BSF) (980Hz) is higher than the Nyquist frequency (500Hz), FFT was unable to show frequency above 500Hz, the frequency above the Nyquist frequency aliased in the 0 to 500 Hz range.

The FFT study of the vibration signal showed that the noises had to be filtered out before subjecting the vibration data into the models. The low frequency noises below 20Hz and the aliased frequencies above 500Hz had to be accounted for. To filter in such a range, a bandpass filter with the range of 20-500Hz was used. The FFT was only used to visualize the spectrum and the filtered time-series data was input into the ML and DL models.

5.3 Output

A Python (Jupyter Notebook) program was developed for the importation of data, proper segmentation and model development. The data imported was imported from MATLAB as column vectors i.e.

Table 5. 1: Types of Data and Data Size

S.N	Types of data	Data Size
1	Normal Data	120,000
2	Inner Race Fault Data	328,000
3	Outer Race Fault Data	258,000
4	Ball fault Data	400,000
	Total Datas	1,106,000

According to the type of model, the number of data interval length and samples per block were applied. The data are represented in terms of number I.e., normal data (0), inner fault (1), outer fault (2) and ball fault (3).

5.3.1 SVM

In the case of SVM model from the original dataset, the shape of Input Data Shape was [5498, 1444], shape of Y_CNN was [5498,10] and the Shape of Label was [5498,1]. The X_feature imported from the Features data contained of sample size [5498,8]. The data was split into 70% for training and 30% for testing. From the model the best parameters obtained were {C': 1,'decision_function_shape': 'ovo', 'gamma': 1, 'kernel': 'rbf'}. The model took computation time of 3min 7.5 seconds with accuracies as I.e.

Table 5. 2: SVM Test and Train Accuracy

SVM Train Accuracy	80.64%
SVM Test Accuracy	81.15%

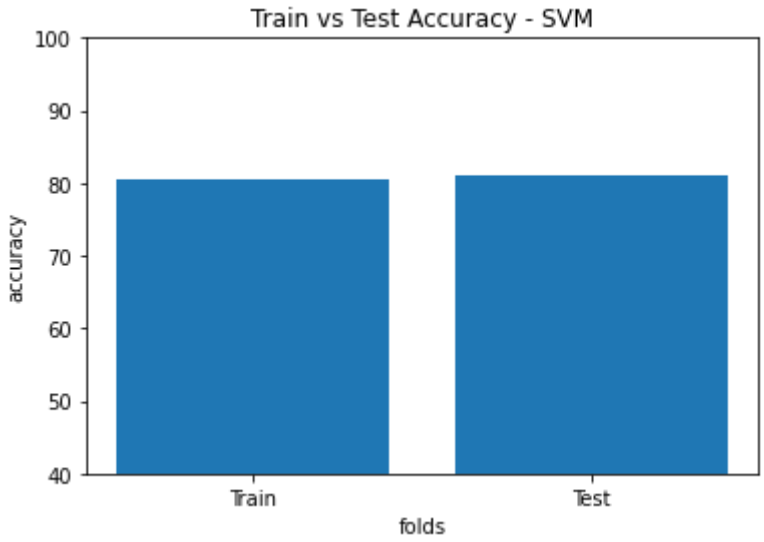

Figure 5. 5: Train vs Test Accuracy for SVM

Also, the confusion matrix obtained i.e.

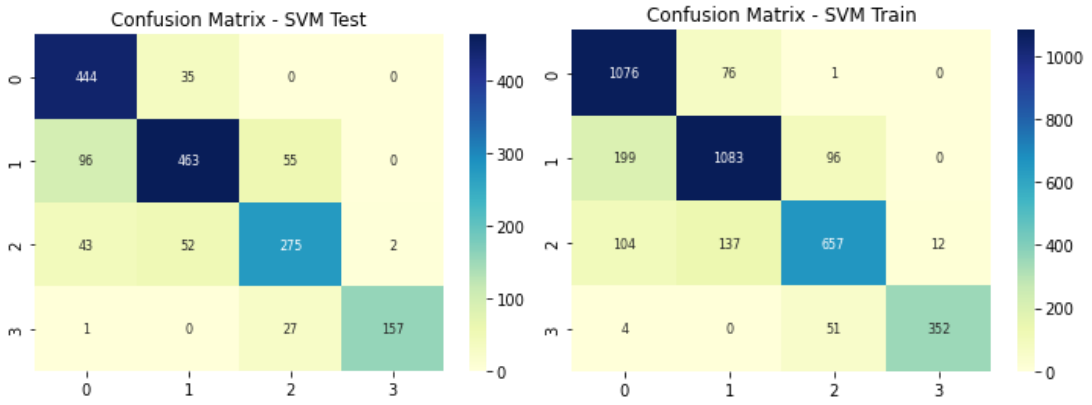

Figure 5. 6: Confusion Matrix for SVM- Train & SVM- Test

The confusion matrix shows the normal data, inner fault data and outer fault (0,1,2) data are easily predicted with high accuracy whereas the ball fault data (3) is not being recognized properly during training and testing.

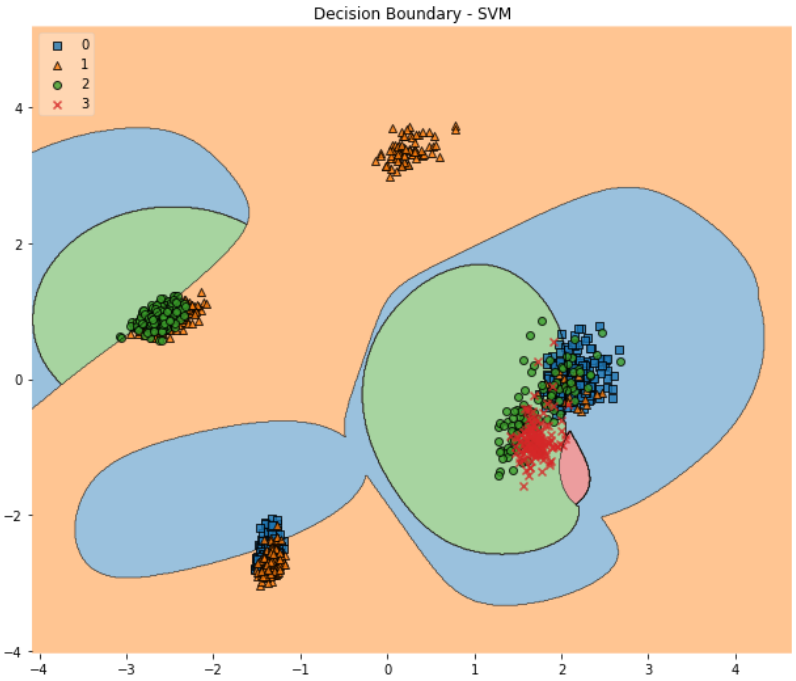

Figure 5. 7: Decision Boundary – SVM

Also, the decision boundary confirms the result obtained in confusion matrix, the features are being separated well enough for normal, inner and outer but the ball fault data feature is trying to interact with other data. Overall, with the data given the model was found to be acceptable.

5.3.2 1D CNN

For the 1D CNN model, the shape of input data was [5486,1936] while shape of label Y_CNN was [5486,10]. Similarly shape of label Y was [5486,1]. The data was split into 70% for training and 30% for testing. The overall computation time for executing the 1D CNN model was 2 minutes and 10 seconds.

Table 5. 3: 1D CNN Train and Test Accuracy

1D CNN Train Accuracy	90.078125%
1D CNN Test Accuracy	98.3596623%

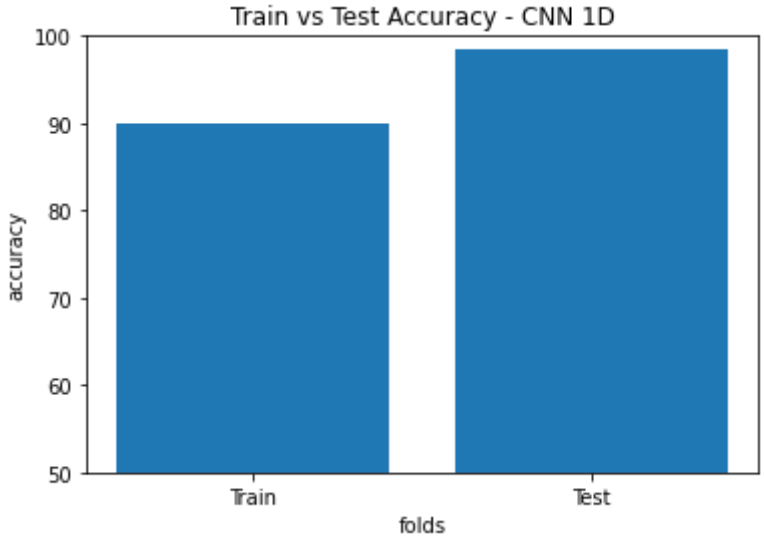

Figure 5. 8: Train vs Test Accuracy for 1D CNN

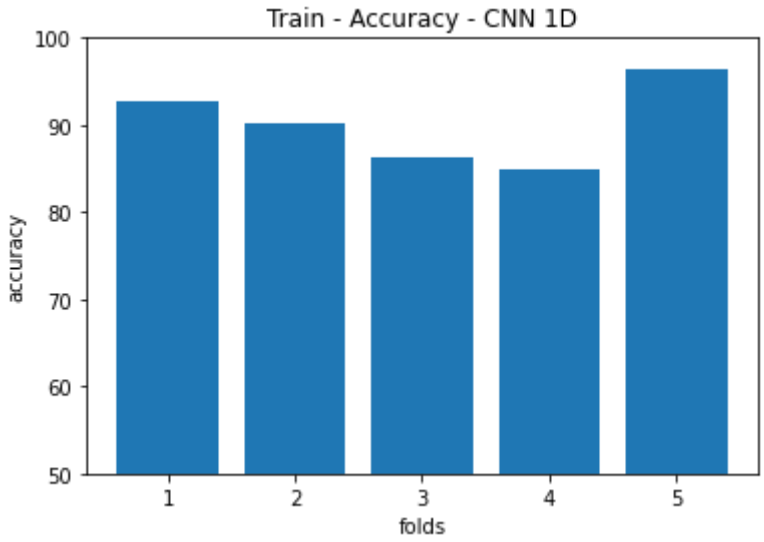

Figure 5. 9: Training Accuracy of 1D CNN at different folds

The accuracy of 1D CNN model can also be visualized through confusion matrix of training and test.

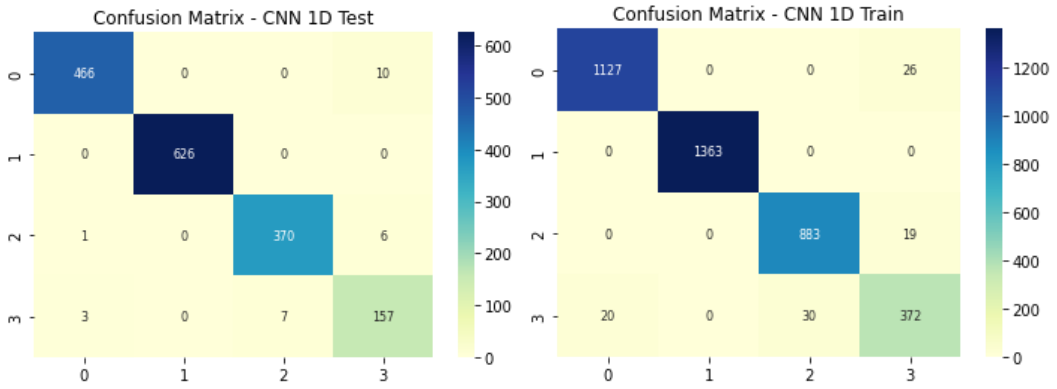

Figure 5. 10: Confusion Matrix for 1D CNN-Test and Train

The performance of the 1D CNN model across different classes can be known by analyzing the above confusion matrix. The diagonal elements representing the number of correctly classified examples are in higher numbers, compared to non-diagonal elements representing the number of misclassified examples which are in very small numbers. Although the ball fault data (3) has higher losses it is still acceptable.

Comparing the training and test accuracy, test accuracy is greater than train accuracy with the standard deviation for 1D CNN training just 4.2. From these results we can say that the model is not overfitting the data which is shown by the following validation graph as well.

Figure 5. 11: 1D CNN Train vs Validation Accuracies

5.3.3 2D CNN

In the case of 2D CNN model, the shape of input data taken was [5486, 44, 44, 1] as two-dimensional array while shape of label Y_CNN was [5486,10]. Similarly shape of label Y was [5486,1]. The data was split into 70% for training and 30% for testing. The overall time for execution of the 2D CNN model was 1 minute and 23 seconds. Accuracies obtained are as follows:

Table 5. 4: 2D CNN Train and Test Accuracy

2D CNN Train Accuracy	86.744791%
2D CNN Test Accuracy	87.181043%

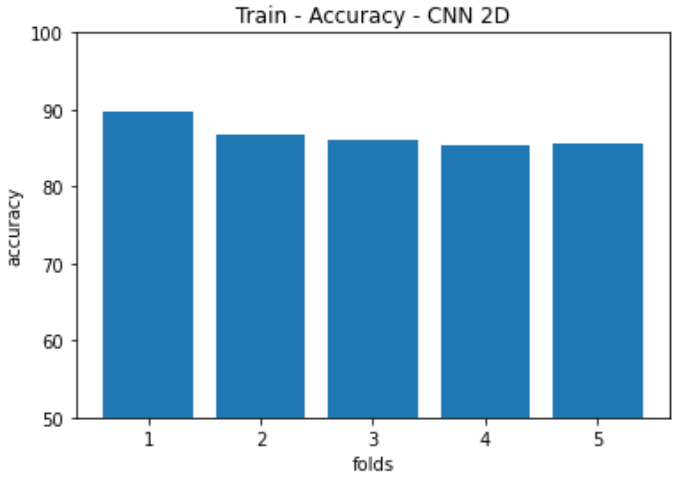

Figure 5. 12: Train vs Test Accuracy for 2D CNN

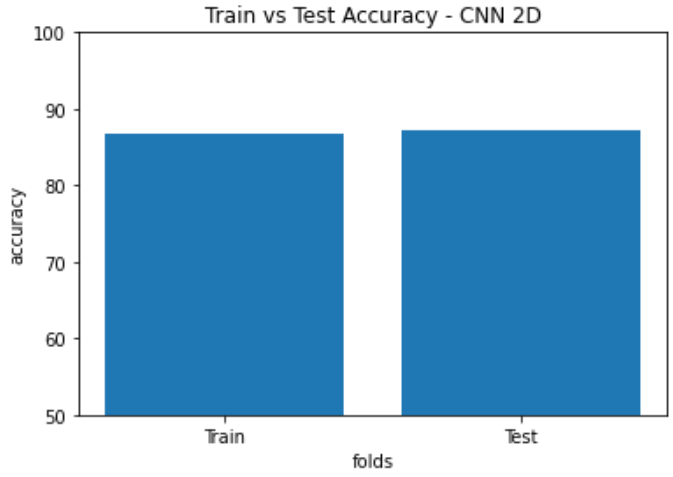

Figure 5. 13: Train vs Test Accuracy for 2D CNN

The accuracy of 2D CNN model can also be visualized through confusion matrix of training and test.

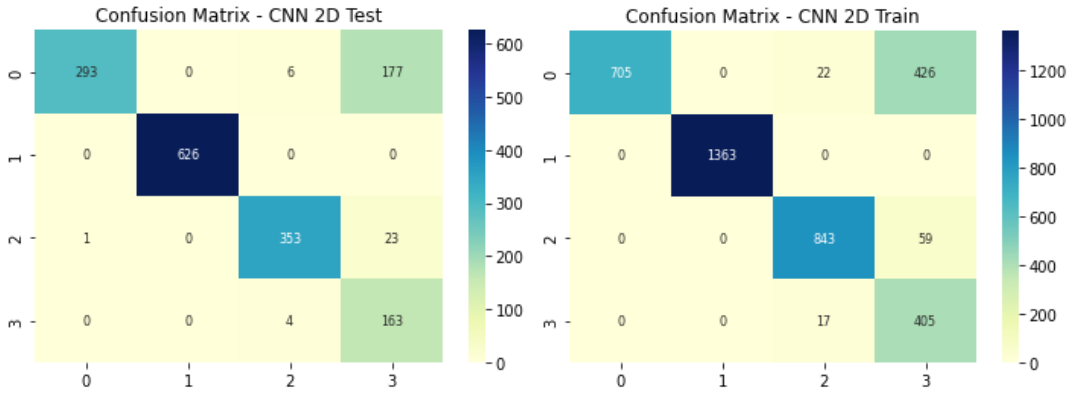

Figure 5. 14: Confusion Matrix for 2D CNN-Test and Train

The performance of the 2D CNN model across different classes can be known by analyzing the above confusion matrix. The diagonal elements representing the number of correctly classified examples are in higher numbers, compared to non-diagonal elements representing the number of misclassified examples which are in very small numbers. Also, like the previous model, major loss occurs in ball fault data.

Comparing the training and test accuracy for overfitting, test accuracy is greater than train accuracy. The standard deviation for 2D CNN training was found to be just 1.5612. From these results we can say that the model is not overfitting the data which is also verified by the following validation graph.

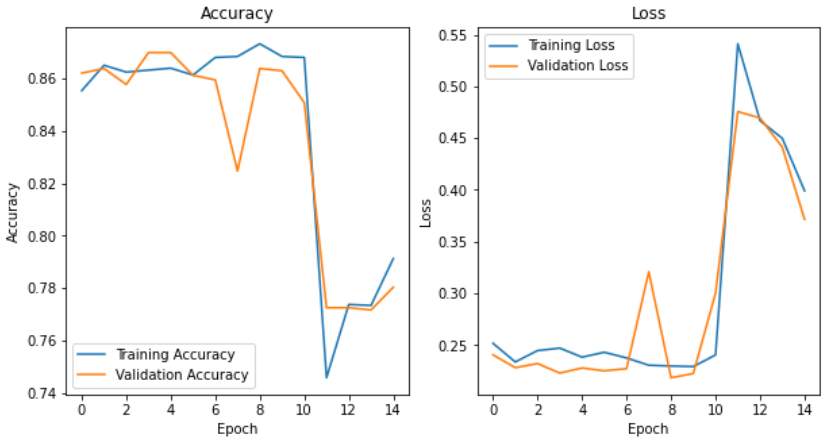

Figure 5. 15: 2D CNN Train vs Validation Accuracies

5.3.4 LSTM

In the LSTM model as per requirement of the model, the data sample taken was larger. From the dataset the shape of Input Data was [5422, 5184], the shape of Y_CNN was [5422, 10] and shape of Y was [5422,1]. K fold validation with 5 splits and 101 random states were applied. The dataset was split into 70% training set and 30% testing set. The model took the longest computation time of 216mins 55.2 secs. The obtained accuracies i.e.

Table 5. 5: LSTM Train and Test Accuracy

LSTM Train Accuracy	77.57%
LSTM Test Accuracy	77.25%

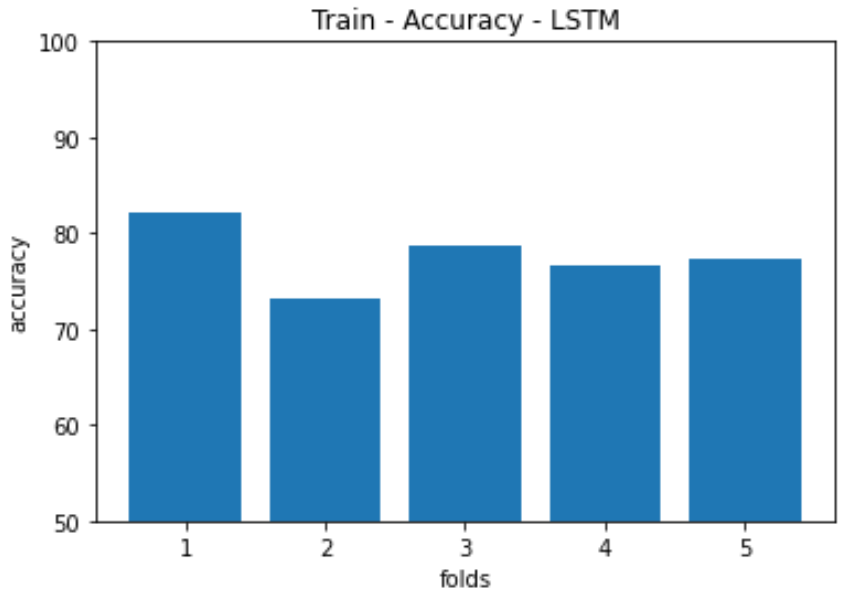

Figure 5. 16: Training Accuracy of LSTM at different folds

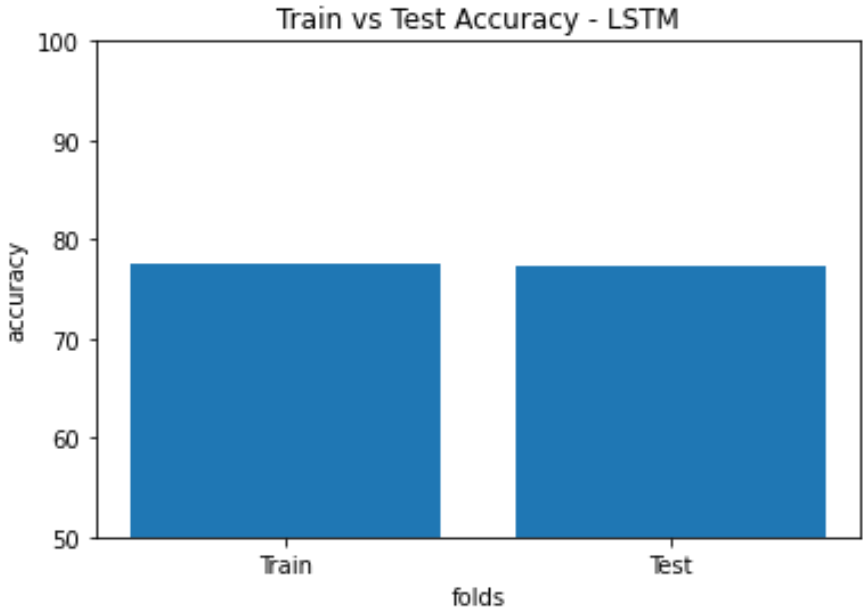

Figure 5. 17: Train vs Test Accuracy for LSTM

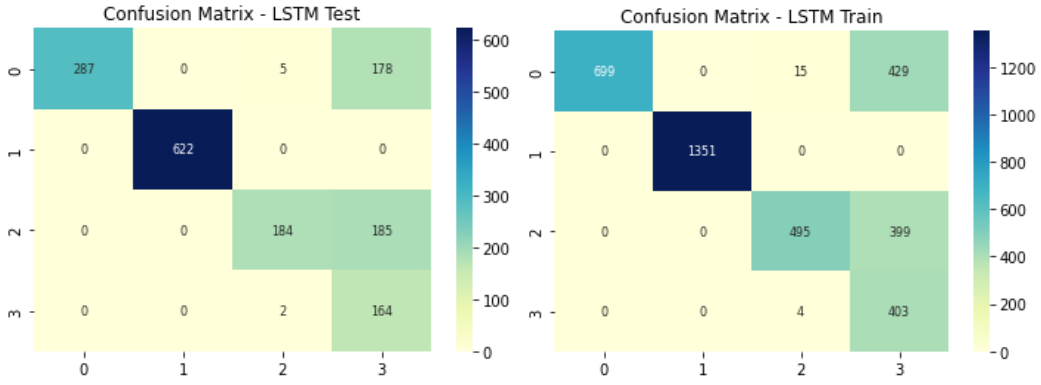

Figure 5. 18: Confusion Matrix for LSTM – Test and Train

Since LSTM is a recurrent neural network and specializes in text and arrays, a clearer idea of the dataset can be obtained from the confusion matrix. As mentioned in other models the normal data (0), inner fault data (1) and even outer fault data (2) have some unique parameters, but the ball fault (3) tries to overlap with others showing problems within the obtained data.

5.3.5 Overall Models Comparison

The overall accuracies obtained from the different deep learning and machine learning models are as:

Table 5. 6: Test vs Train Accuracy Comparison Between Models

S.N.	Models in Experimental Setup	Train Accuracy (%)	Test Accuracy (%)
1	1D CNN	90.07	98.35
2	2D CNN	86.74	87.18
3	LSTM	77.57	77.25
4	SVM	80.64	81.15

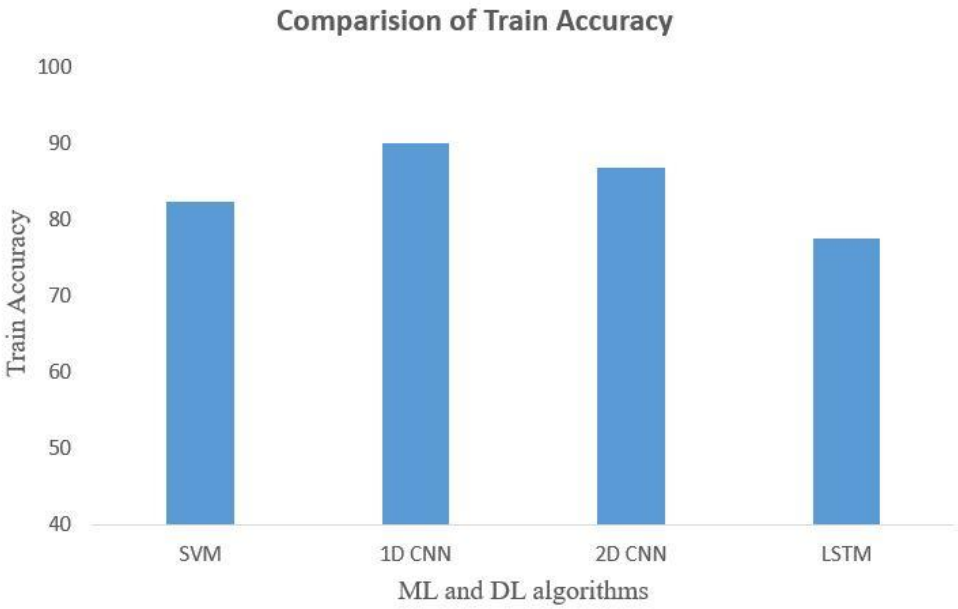

Figure 5. 19: Model Accuracies for Training in Experimental Setup

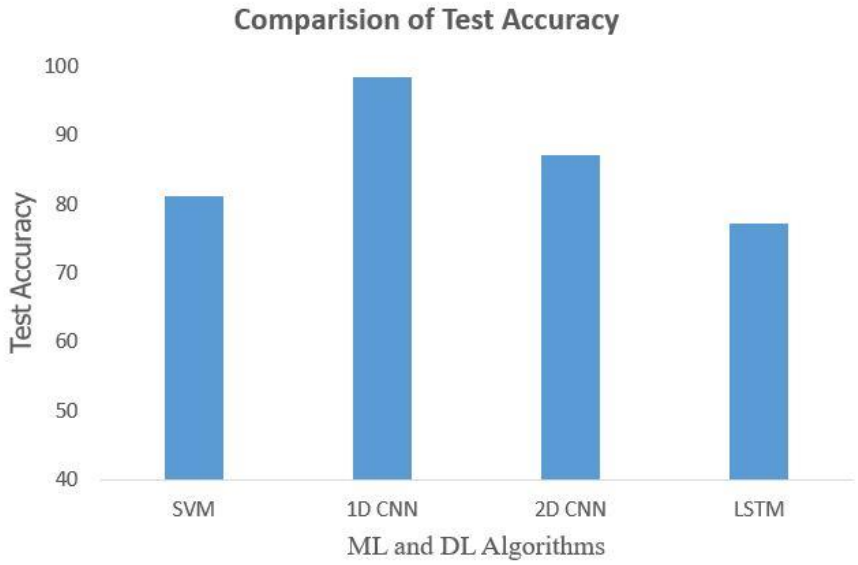

Figure 5. 20: Model Accuracies for Testing in Experimental Setup

5.4 Model validation for CWRU bearing Dataset

The above-mentioned models were tested for 10 sample data using CWRU bearing dataset. The shape of Input Data was [24276, 1681], shape of Y_CNN was [24276, 10] and shape of Y was [24276,1]. For each model the dataset was divided into 65% training data and 35% testing data.

The accuracies obtained were as follows:

Table 5. 7: Train and Test Accuracy for algorithms using CWRU Dataset

S.N.	Models	Train Accuracy (%)	Test Accuracy (%)
1	1D CNN	98.14	98.52
2	2D CNN	96.11	93.92
3	LSTM	94.58	96.09
4	SVM	92.73	92.47

- **Results for SVM:**

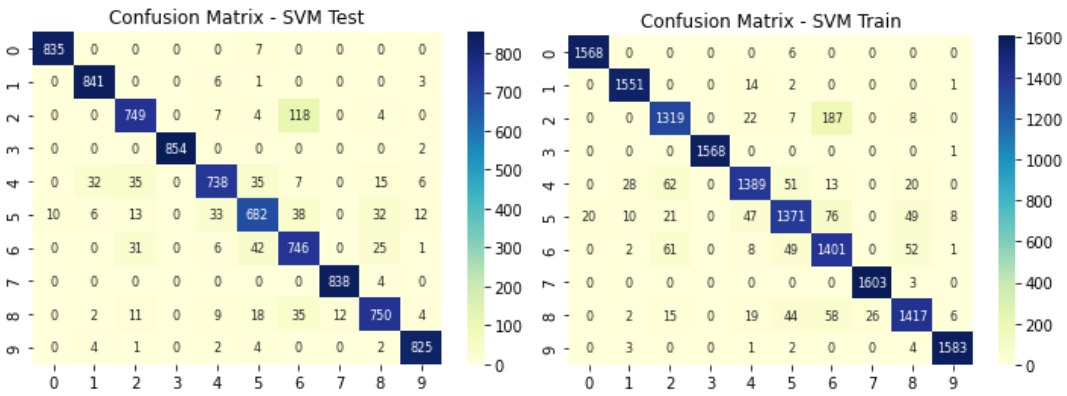

Figure 5. 21: Confusion Matrix for Testing Data using SVM

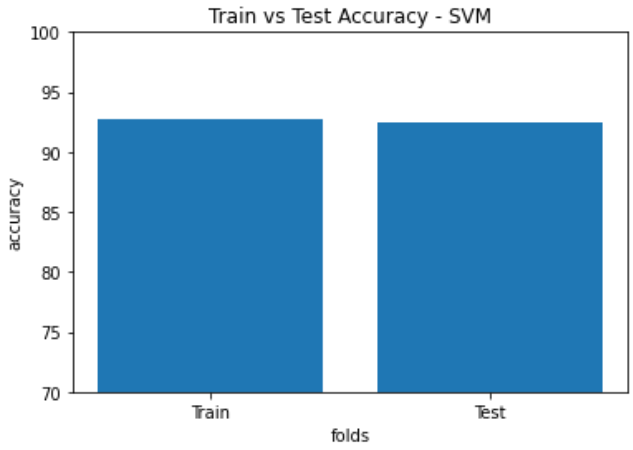

Figure 5. 22: CWRU Train vs Test Accuracy for SVM

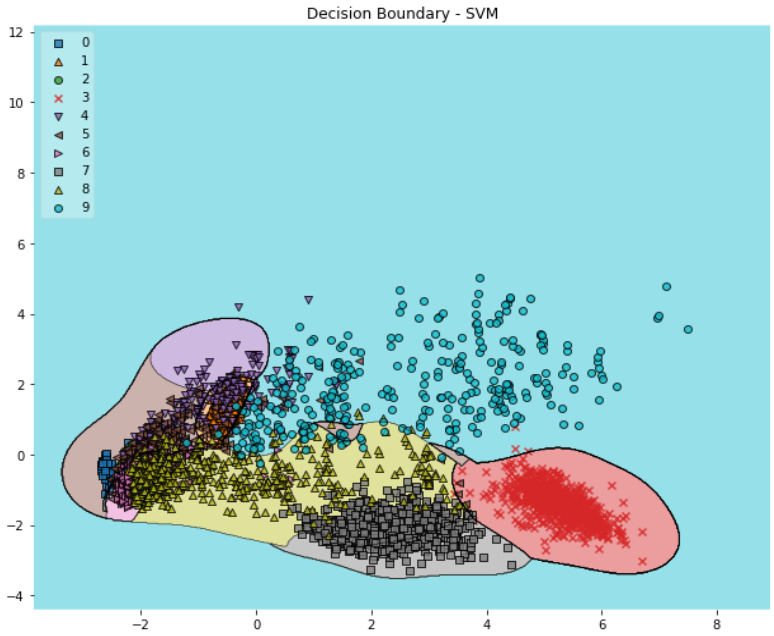

Figure 5. 23: CWRU- Decision Boundary for SVM

- **Results for 1D CNN:**

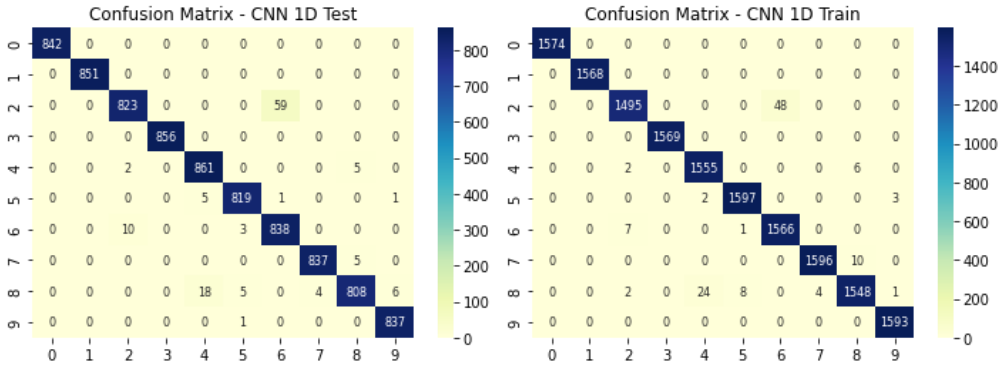

Figure 5. 24: CWRU- Confusion Matrix for Test Data via 1D CNN

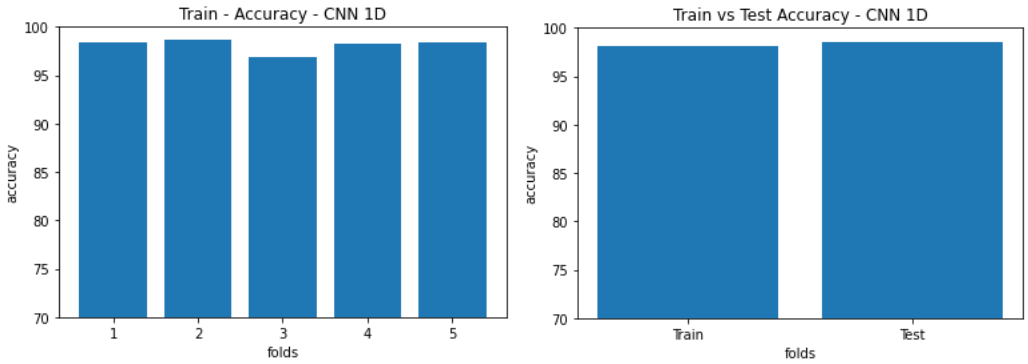

Figure 5. 25: CWRU- Test and Train Accuracy via 1D CNN

- **Results for 2D CNN:**

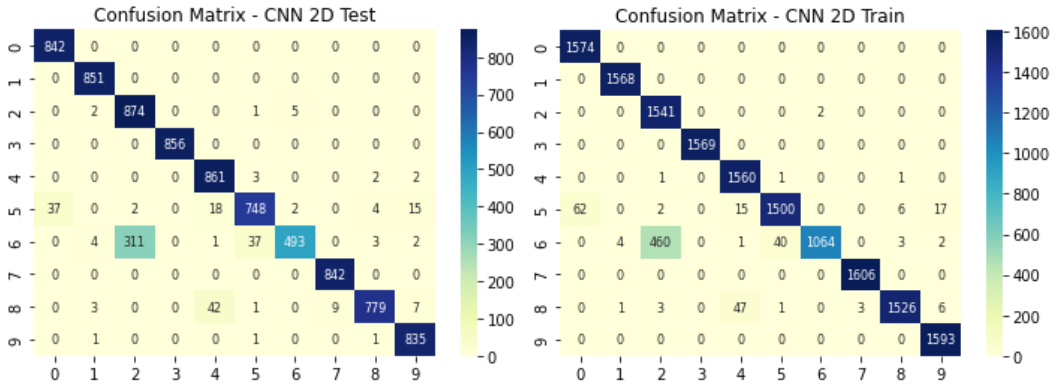

Figure 5. 26: CWRU – Test and Train Confusion Matrices for 2D CNN

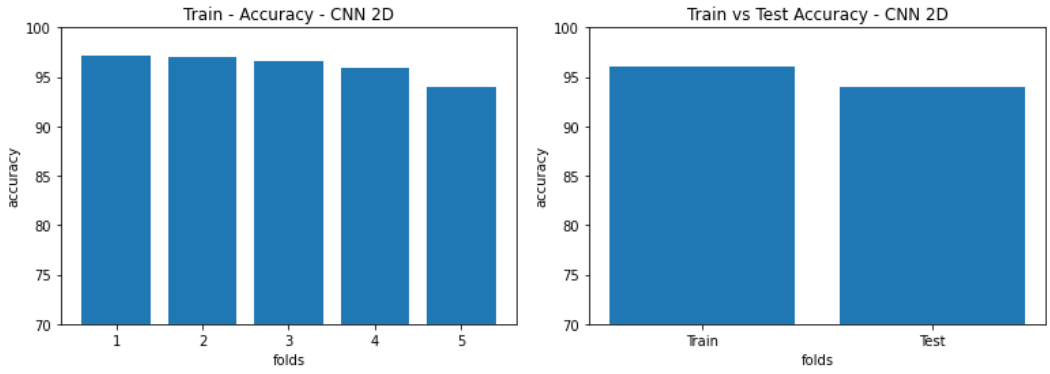

Figure 5. 27: CWRU- Train and Test Accuracies for 2D CNN

- **Results in LSTM:**

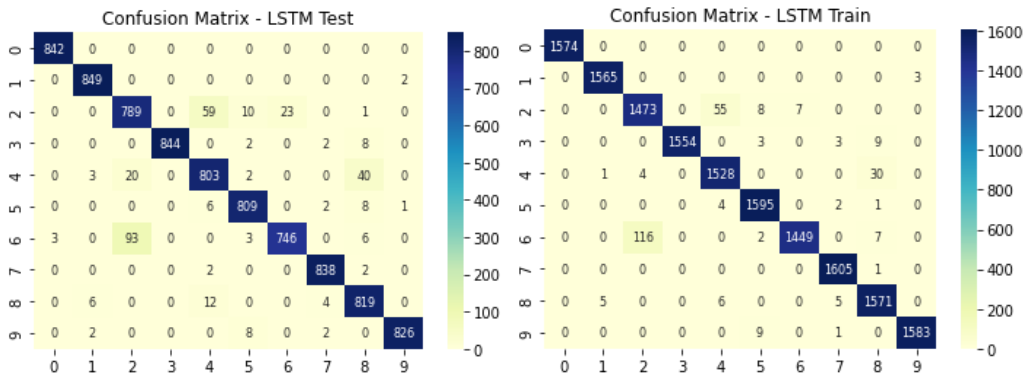

Figure 5. 28: CWRU- Test and Train Confusion Matrices in LSTM

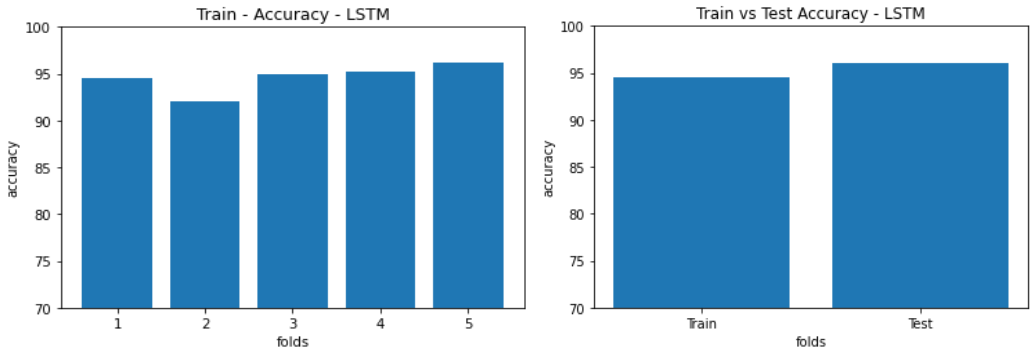

Figure 5. 29: CWRU – Train and Test Accuracy for LSTM

Overall results:

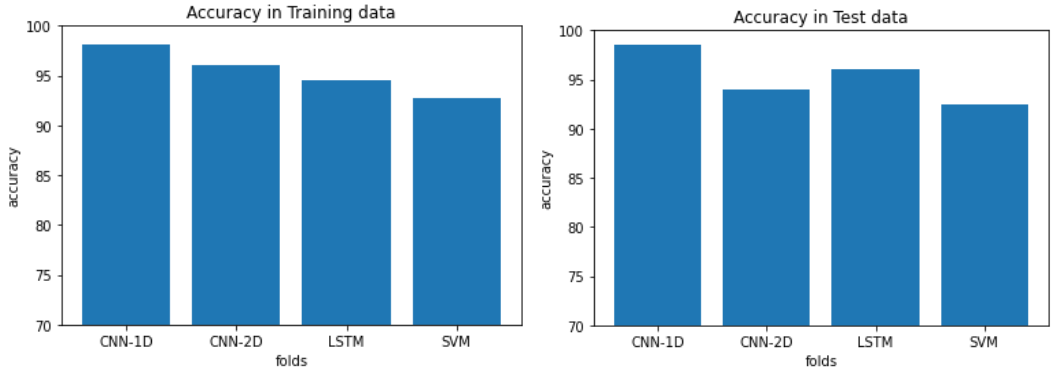

Figure 5. 30: Overall Training and Testing Accuracies

5.5 Limitations

The following are the limitations faced in this project:

- Unavailability of the data logger software for the 3D Vibration Tester.
- High sensitivity and high sampling frequency accelerometer was unavailable and beyond the scope of our budget.
- Due to the lack of instruments for standard fault introduction such as Electrical Discharge Machining, precise faults could not be introduced.
- Damping effects have been neglected.
- Variations in rpm and loading conditions have not been considered.

5.6 Problems Faced and Recommendations

- Starting the DLE engine: Manually starting the DLE55 Piston Engine is a labor intensive task. Despite numerous attempts to start the engine manually, the engine only started a handful of times. An electric starter is recommended to start the engine.

- Configuring the 3D vibration tester: The 3D vibration tester available was not equipped with its own data logger due to which the data could not be acquired according to the required sampling frequency. A high sensitivity accelerometer with the required sampling frequency along with the DAQ system shall be a suitable alternative to the 3D Vibration tester.

5.7 Cost Analysis

Table 5.8: Cost Analysis

S.N.	Instruments	Quantity	Price(Rs.)
1	AC Induction Motor	1	2200
2	Arduino Uno	1	1500
3	TTL Connector	1	500
4	Ball Bearings	3	450
5	ADXL 335 Accelerometer	1	350
6	Fuel	3 liters	550
7	USB to RS-232 connector	1	400
8	12V LiPo Battery	1	4200
9	Engine Workbench	1	3000
10	Miscellaneous		5000
	Total		18,150

Chapter 6 : CONCLUSION AND FUTURE ENHANCEMENTS

6.1 Conclusion

In conclusion, the project aimed to diagnose faulty bearings using vibration data and deep learning models, with an emphasis on real-life experimentation. The experimental setup collected vibration data in time-series. The obtained dataset was tested on a developed program consisting of 1-D CNN, 2D CNN, LSTM, and SVM models, which yielded accuracies of (82.328,81.155), (90.78,98.359), (86.744, 87.181), and (77.575,77.258), respectively. The results showed that the developed program had high accuracies for all models tested, indicating the potential of the program for practical applications in predictive maintenance and fault diagnosis in various industries. The experimental setup provided a real-world simulation of a faulty bearing, enhancing the accuracy and reliability of the developed program. This highlights the importance of acquiring accurate and reliable data for developing a robust model for fault diagnosis in industrial applications.

Overall, the project provided a good understanding of vibration analysis and deep learning programs with an acceptable level of accuracy. However, the project can be further developed by using better sensing devices and advanced models to improve the accuracy levels even further. In summary, the project demonstrated the potential of using deep learning models and machine learning for fault diagnosis of faulty bearings and highlighted the significance of experimentation in model development and fault diagnosis in industrial applications.

6.2 Scope for Future Enhancement

The potential areas for future enhancement are:

- Variations in loading conditions, rpm and fault severity can be considered.
- A variety of other signal processing methods, especially STFT can be used to preprocess the data.
- Faults can be precisely introduced and accurately measured using Electrical Discharge Machining or other techniques.
- Effects of variations in vibration signature caused due to different types of lubricants, lubrication conditions and presence of contaminants can be explored.
- Accelerometer with a higher sensitivity along with a high sampling rate could be used.
- Comparison studies can be done against other different methods such as Motor Current Signature Analysis, acoustics and image processing to compare with the accelerometer vibration analysis.
- Other different advanced Deep Learning models such as GAN (Generative Adversarial Network) and Variational Autoencoders can be used to generate better diagnosis models.

REFERENCES

[1] Fan Feilong, Cao Ming, and Liu Qian. "Naturally-induced Early Aviation Bearing Fault Test and Early Bearing Fault Detection". In: *2021 Global Reliability and Prognostics and Health Management (PHM-Nanjing)*. 2021, pp. 1–6. DOI:10.1109/PHM-Nanjing52125.2021.9612980.

[2] Tatjana Lazovic, Mileta Ristivojevic, and Radivoje Mitrovic. "Mathematical Model of Load Distribution in Rolling Bearing". In: *FME Transactions* 36 (Jan. 2008).

[3] Chetan Chaudhari et al. "A STUDY OF BEARING AND ITS TYPES". In: Apr.2015.

[4] Yuanyang Cai. "BEARING FAULT DIAGNOSIS USING DEEP LEARNING NEURAL NETWORKS WITH INPUT PROCESSING". In: (Dec. 2021). DOI: 10.25394/PGS.17162480.v1. URL: https://hammer.purdue.edu/articles/thesis/BEARING_FAULT_DIAGNOSIS_USING_DEEP_LEARNING_NEURAL_NETWORKS_WITH_INPUT_PROCESSING/ 17162480.

[5] Adnan Althubaiti, Faris Elasha, and Joao Amaral Teixeira. "Fault diagnosis and health management of bearings in rotating equipment based on vibration analysis– a review". In: *Journal of Vibroengineering* 24.1 (Nov. 2021), pp. 46–74. DOI: 10.21595/jve.2021.22100. URL: https://doi.org/10.21595%2Fjve.2021.22100.

[6] S Vishwakarma et al. "Analyzing Vibrations through Fast Fourier Transform (FFT) for Machine Health Monitoring: A Review of Fundamentals and Applied Methods". In: (Jan. 2022).

[7] N.T. van der Merwe and A.J. Hoffman. "A modified cepstrum analysis applied to vibrational signals". In: *2002 14th International Conference on*

Digital Signal Processing Proceedings. DSP 2002 (Cat. No.02TH8628). Vol. 2. 2002, 873–876 vol.2. DOI: 10.1109/ICDSP.2002.1028229.

[8] G.G. Yen and K.-C. Lin. "Wavelet packet feature extraction for vibration monitoring". In: *IEEE Transactions on Industrial Electronics* 47.3 (2000), pp. 650– 667. DOI: 10.1109/41.847906.

[9] Qiuhua Du and Shunian Yang. "Application of the EMD method in the vibration analysis of ball bearings". In: *Mechanical Systems and Signal Processing* 21 (Aug. 2007), pp. 2634–2644. DOI: 10.1016/j.ymssp.2007.01.006.

[10] M. Safizadeh, A. Lakis, and Marc Thomas. "USING SHORT-TIME FOURIER TRANSFORM IN MACHINERY DIAGNOSIS". In: 3 (Jan. 2005).

[11] Jing Tian et al. "Motor Bearing Fault Detection Using Spectral Kurtosis-Based Feature Extraction Coupled With K-Nearest Neighbor Distance Analysis". In: *IEEE Transactions on Industrial Electronics* 63.3 (2016), pp. 1793–1803. DOI: 10.1109/TIE.2015.2509913.

[12] Farzin Piltan and Jong-Myon Kim. "Bearing Fault Identification Using Machine Learning and Adaptive Cascade Fault Observer". In: *Applied Sciences* 10.17 (Aug. 2020), p. 5827. ISSN: 2076-3417. DOI: 10.3390/app10175827. URL: http://dx.doi.org/10.3390/app10175827.

[13] Shuang Lu, Fujin Yu, and Jing Liu. "Bearing Fault Diagnosis Based on K-L Transform and Support Vector Machine". In: *Third International Conference on Natural Computation (ICNC 2007)*. Vol. 1. 2007, pp. 522–527. DOI: 10.1109/ ICNC.2007.282.

[14] Haidong Shao et al. "Electric Locomotive Bearing Fault Diagnosis Using a Novel Convolutional Deep Belief Network". In: *IEEE Transactions on*

Industrial Electronics 65.3 (2018), pp. 2727–2736. DOI: 10.1109/TIE.2017.2745473.

[15] Emanuele Principi et al. “Unsupervised electric motor fault detection by using deep autoencoders”. In: *IEEE/CAA Journal of Automatica Sinica* 6.2 (2019), pp. 441–451. DOI: 10.1109/JAS.2019.1911393.

[16] OpenAI. *Fault Diagnosis in a Ball Bearing*. AI language model. 2023.URL: https://openai.com.

[17] Jinane Harmouche, Claude Delpha, and Demba Diallo. “Improved Fault Diagnosis of Ball Bearings Based on the Global Spectrum of Vibration Signals”. In: *IEEE Transactions on Energy Conversion* 30.1 (2015), pp. 376–383. DOI: 10.1109/TEC.2014.2341620.

[18] Basim Al-Najjar. “Condition-based maintenance : Selection and improvement of a cost-effective vibration-based policy in rolling element bearings”. In: 1997.

[19] Dubravko Miljkovic. “Detecting Aircraft Ball Problems by Analysis of´ Engine Parameters”. In: *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2019, pp. 937–942. DOI: 10.23919/MIPRO.2019.8757079.

[20] Donald Gray et al. “A neural network based approach for the detection of faults in the brushless excitation of a synchronous motor”. In: *2009 IEEE International Conference on Electro/Information Technology*. 2009, pp. 423–428. DOI: 10.1109/EIT.2009.5189654.

[21] Pavan Kankar, Satish Chandra Sharma, and SURAJ HARSHA. “Fault diagnosis of ball bearings using machine learning methods”. In: *Expert Syst. Appl.* 38 (Mar.2011), pp. 1876–1886. DOI: 10.1016/j.eswa.2010.07.119.

[22] Wang Fuan et al. “An adaptive deep convolutional neural network for rolling bearing fault diagnosis”. In: *Measurement Science and Technology* 28.9

(Aug.2017), p. 095005. DOI: 10.1088/1361-6501/aa6e22. URL: https://doi.org/10.1088/1361-6501/aa6e22.

[23] Lane Maria Rabelo Baccarini et al. “SVM practical industrial application for mechanical faults diagnostic”. In: *Expert Systems with Applications* 38.6 (2011), pp. 6980–6984. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2010.12.017.URL:https://www.sciencedirect.com/science/article/pii/S0957417410013801.

[24] Amin Khorram, Mohammad Khalooei, and Mansoor Rezghi. “End-to-end CNN + LSTM deep learning approach for bearing fault diagnosis”. In: *Applied Intelligence* 51 (Feb. 2021), pp. 1–16. DOI: 10.1007/s10489-020-01859-1.

[25] Pengfei Liang et al. “Intelligent Fault Diagnosis of Rolling Element Bearing Based on Convolutional Neural Network and Frequency Spectrograms”. In: *2019 IEEE International Conference on Prognostics and Health Management (ICPHM)*. 2019, pp. 1–5. DOI: 10.1109/ICPHM.2019.8819444.

[26] Alpha-Quantum, "Long Short-Term Memory (LSTM) with Python," AlphaQuantum Blog, 2021. [Online]. Available: https://www.alphaquantum.com/blog/long-short-term-memory-lstm-with-python/. [Accessed: Mar. 11, 2023].

[27] GeeksforGeeks, "Long Short-Term Memory Networks - Explanation," GeeksforGeeks, 2021. [Online]. Available: https://www.geeksforgeeks.org/long-short-term-memory-networksexplanation/. [Accessed: Mar. 11, 2023].

APPENDIX 1

Python Codes for FFT Plot

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import openpyxl

# Set file location
file_location = r'C:\Desktop\py files\mikesh\excel files'

# Import csv file
df = pd.read_excel(file_location + r'\normal2.xlsx',  header=None, names=['acc_x', 'acc_y'], skiprows=1)

# Select specific section of data
start = 20000
end = start + 4096
df_section = df[start:end]

# Plot data
plt.figure(figsize=(12,4))
plt.plot(df_section.index, df_section['acc_x'], label='Acc_X')
plt.plot(df_section.index, df_section['acc_y'], label='Acc_Y')
plt.xlabel('Time')
plt.ylabel('Acceleration')
plt.legend()
plt.show()

# FFT
# Number of sample points
N = len(df_section)
# Sampling frequency of signal
fs = 1000
# Time interval between samples
T = 1 / fs
# Perform FFT on signal
yf_x = np.fft.fft(df_section['acc_x'])
yf_y = np.fft.fft(df_section['acc_y'])
# Create new x-axis: frequency from signal
xf = np.linspace(0.0, fs/2, N//2)

# Plot results for entire frequency range
plt.figure(figsize=(12,4))
plt.plot(xf, 2.0/N * np.abs(yf_x[:N//2]), label='Acc_X')
plt.plot(xf, 2.0/N * np.abs(yf_y[:N//2]), label='Acc_Y')
plt.xlabel('Frequency')
```

```
plt.ylabel('Amplitude')
plt.legend()
plt.grid()
plt.show()

# Plot specific frequency range
lower = 50
medium = 250
upper = 500
plt.figure(figsize=(12,4))
plt.plot(xf, 2.0/N * np.abs(yf_x[:N//2]), label='Acc_X')
plt.plot(xf, 2.0/N * np.abs(yf_y[:N//2]), label='Acc_Y')
plt.xlim(lower, upper)
plt.ylim(0,0.1)
plt.xlabel('Frequency')
plt.ylabel('Amplitude')
plt.legend()
plt.grid()
plt.show()
```

Python Codes for Bandpass Filter

```
import numpy as np
import pandas as pd
from scipy import signal
import matplotlib.pyplot as plt
import openpyxl

# Set file location
file_location = r'C:\Desktop\py files\mikesh\excel files'

# Import excel file
# Import excel file, skipping first row
df = pd.read_excel(file_location + r'\normal2.xlsx', header=None, skiprows=1,
names=['acc_x', 'acc_y'])

# Sampling frequency of signal
fs = 1000
# Time interval between samples
T = 1 / fs

# Filter design
fc = [20, 500] # Band pass cutoff frequency range
b, a = signal.butter(4, [f / (fs/2) for f in fc], 'bandpass') # 4th order Butterworth
bandpass filter

# Apply filters
```

```
df['acc_x_bandpass'] = signal.filtfilt(b, a, df['acc_x'])
df['acc_y_bandpass'] = signal.filtfilt(b, a, df['acc_y'])

# Reset index to a single level
df.reset_index(drop=True, inplace=True)

# Plot filtered data
fig, axs = plt.subplots(2, 1, figsize=(8, 6))
axs[0].plot(df.index*T, df['acc_x_bandpass'])
axs[0].set_xlabel('Time (s)')
axs[0].set_ylabel('Acceleration (m/s^2)')
axs[0].set_title('X-axis filtered data')
axs[1].plot(df.index*T, df['acc_y_bandpass'])
axs[1].set_xlabel('Time (s)')
axs[1].set_ylabel('Acceleration (m/s^2)')
axs[1].set_title('Y-axis filtered data')
plt.tight_layout()
plt.show()

# Write filtered data to excel file
df.to_excel(file_location + r'\filtered_normal_2_20hz_500hz.xlsx', index=False)

# FFT plot of filtered data
n = len(df)  # length of the signal
f = np.linspace(0, fs, n)  # frequency range
xf = np.fft.fft(df['acc_x_bandpass'])/n  # fft of x-axis filtered data
yf = np.fft.fft(df['acc_y_bandpass'])/n  # fft of y-axis filtered data

fig, axs = plt.subplots(2, 1, figsize=(8, 6))
axs[0].plot(f[:n//2], 2*np.abs(xf[:n//2]))
axs[0].set_xlabel('Frequency (Hz)')
axs[0].set_ylabel('Magnitude')
axs[0].set_title('X-axis filtered data')
axs[1].plot(f[:n//2], 2*np.abs(yf[:n//2]))
axs[1].set_xlabel('Frequency (Hz)')
axs[1].set_ylabel('Magnitude')
axs[1].set_title('Y-axis filtered data')
plt.tight_layout()
plt.show()
```

Matlab Program For Feature Extraction

```
% Load the data from the TotalData.mat file

load('M:\Python\Classification_of_bearing_faults_using_ML-main\BearingData_CaseWestern\TotalData.mat');


% Define the number of features you want to extract

num_features = 6;


% Define the number of data points you want per column

num_data_points = 201;


% Reshape the data into a matrix with the desired number of columns

data_matrix = reshape(TotalData, num_data_points, [])';


% Extract the features from the data

max_val = max(data_matrix, [], 2);

min_val = min(data_matrix, [], 2);

mean_val = mean(data_matrix, 2);

var_val = var(data_matrix, [], 2);

std_val = std(data_matrix, [], 2);

rms_val = rms(data_matrix,[], 2);

skew_val = skewness(data_matrix, [], 2);

crest_val = crestFactor(data_matrix, 1);

kurt_val = kurtosis(data_matrix, [], 2);

amp_val = max(abs(data_matrix), [], 2);


% Combine the features into a single matrix

feature_matrix = [max_val, min_val, mean_val, var_val, std_val, rms_val, skew_val, crest_val, kurt_val, amp_val];
```

```
% Transpose the feature matrix to get the desired shape

feature_matrix = feature_matrix';


% Display the feature matrix

disp(feature_matrix);


% Combine the features into a matrix

data_matrix = [max_val; min_val; peak_to_peak; mean_val; var_val; std_val;
rms_val; skew_val; crest_val; kurt_val; amp_val]';


% Display the matrix

disp(data_matrix);
```

APPENDIX 2

Program for 'Fault Diagnosis on a Ball Bearing Using Vibration Analysis'

```
# Importation of all the required Libraries:
import scipy.io
import numpy as np
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import confusion_matrix
import tensorflow as tf
from keras import layers, models
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, Dropout
from keras.utils import to_categorical
from sklearn.metrics import confusion_matrix, classification_report
from keras.models import Sequential
import numpy as np
import scipy.io
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout
from keras.callbacks import EarlyStopping
```

Data Import

```
# Importing the required dataset in matlab/excel format:

def ImportData():
    X111_normal1_007 =
scipy.io.loadmat('BearingData_CaseWestern/NORMAL1.mat')['X099_DE_time']
    X176_InnerRace_014 =
scipy.io.loadmat('BearingData_CaseWestern/INNER.mat')['X111_DE_time']
    X203_Outer_014 =
scipy.io.loadmat('BearingData_CaseWestern/OUTER.mat')['X203_DE_time']
    X191_Ball_014 =
scipy.io.loadmat('BearingData_CaseWestern/BALL.mat')['X191_DE_time']
    return
```

[X176_InnerRace_014,X191_Ball_014,X203_Outer_014,X111_normal1_007]

Similarly other datas can be imported as per requirements. Here we have taken the major dataset containg 1,106,000 datas.
 # The data contains mainly normal,inner faults, outer faults and ball fault.

```
def Sampling(Data, interval_length, samples_per_block):
    # Calculate the number of blocks that can be sampled based on the interval length
    No_of_blocks = (round(len(Data)/interval_length) -
round(samples_per_block/interval_length)-1)
    SplitData = np.zeros([No_of_blocks, samples_per_block])
    for i in range(No_of_blocks):
        SplitData[i,:] =
(Data[i*interval_length:(i*interval_length)+samples_per_block]).T
    return SplitData

def DataPreparation(Data, interval_length, samples_per_block):
    for count,i in enumerate(Data):
        SplitData = Sampling(i, interval_length, samples_per_block)
        y = np.zeros([len(SplitData),10])
        y[:,count] = 1
        y1 = np.zeros([len(SplitData),1])
        y1[:,0] = count
        # Stack up and label the data
       if count==0:
            X = SplitData
            LabelPositional = y
            Label = y1
        else:
            X = np.append(X, SplitData, axis=0)
            LabelPositional = np.append(LabelPositional,y,axis=0)
            Label = np.append(Label,y1,axis=0)
    return X, LabelPositional, Label
```

Here, the data interval length is set as per requirement of different training models.An example:
 # For 1D CNN and 2D CNN the samples per block= 1936 so that the two dimensional data can be in [44,44] form.
 # Similarly for LSTM which requires higher data number, the sample per block= 5184
 # For SVM the data samples per block is kept= 1444

```
Data = ImportData()
 interval_length = 200
samples_per_block = 1936
```

Y_CNN is of shape (n, 10) representing 10 classes as 10 columns. In each sample, the data has been divided in these columns for easier computation,

```
# the corresponding column value is marked 1 and the rest as 0, facilitating Softmax implementation in CNN
# Y is of shape (m, 1) where column values are between 0 and 4 representing the classes directly. - 1-hot encoding

X, Y_CNN, Y = DataPreparation(Data, interval_length, samples_per_block)

print('Shape of Input Data =', X.shape)
 print('Shape of Label Y_CNN =', Y_CNN.shape)
 print('Shape of Label Y =', Y.shape)
```

1D-CNN

```
# Saving the provided data for future use
 XX = {'X':X}
 scipy.io.savemat('Data.mat', XX)

# Applying k-fold cross validation for easier computation of CNN
 kSplits = 5
 kfold = KFold(n_splits=5, random_state=101, shuffle=True)

# Reshaping the given data into - 1 dimensional feed
Input_1D = X.reshape([-1,1936,1])

# Splitting of Testing data-Training data for 1D CNN
X_1D_train, X_1D_test, y_1D_train, y_1D_test = train_test_split(Input_1D, Y_CNN, train_size=0.7,test_size=0.3, random_state=101)
 print(type(X_1D_test))

# Defination of the CNN Classification model for 1D CNN
 class CNN_1D():
   def __init__(self):
     self.model = self.CreateModel()

  def CreateModel(self):
     model = models.Sequential([
        layers.Conv1D(filters=16, kernel_size=3, strides=2, activation='relu'),
        layers.MaxPool1D(pool_size=2),
        layers.Conv1D(filters=32, kernel_size=3, strides=2, activation='relu'),
        layers.MaxPool1D(pool_size=2),
        layers.Conv1D(filters=64, kernel_size=3, strides=2, activation='relu'),
        layers.MaxPool1D(pool_size=2),
        layers.Conv1D(filters=128, kernel_size=3, strides=2, activation='relu'),
        layers.MaxPool1D(pool_size=2),
        layers.Flatten(),
        layers.InputLayer(),
        layers.Dense(100,activation='relu'),
        layers.Dense(50,activation='relu'),
        layers.Dense(10),
        layers.Softmax()
```

```
        ])
    model.compile(optimizer='adam',
            loss=tf.keras.losses.CategoricalCrossentropy(),
            metrics=['accuracy'])
    return model

accuracy_1D = []

# Training the 1D CNN model acoording to required conditions
for train, test in kfold.split(X_1D_train,y_1D_train):
   Classification_1D = CNN_1D()
   history = Classification_1D.model.fit(X_1D_train[train], y_1D_train[train],
verbose=1, epochs=15)

  # Evaluation of accuracy of the model on the training set
  kf_loss, kf_accuracy = Classification_1D.model.evaluate(X_1D_train[test],
y_1D_train[test])
  accuracy_1D.append(kf_accuracy)

CNN_1D_train_accuracy = np.average(accuracy_1D)*100
 print('CNN 1D train accuracy =', CNN_1D_train_accuracy)

# Evaluation of accuracy of the 1D-CNN model on the test set
 CNN_1D_test_loss, CNN_1D_test_accuracy =
Classification_1D.model.evaluate(X_1D_test, y_1D_test)
 CNN_1D_test_accuracy*=100
 print('CNN 1D test accuracy =', CNN_1D_test_accuracy)

# Saving the given model for future use
 Classification_1D.model.save('Classification_1D.model.h5')

# Defination of the confusion matrix for future plotting
def ConfusionMatrix(Model, X, y):
      y_pred = np.argmax(Model.model.predict(X), axis=1)
   ConfusionMat = confusion_matrix(np.argmax(y, axis=1), y_pred)
   return ConfusionMat

# Plotting Confusion matrix for 1D CNN. Train, Test. Also plotting bar graph for
Training and Testing accuracies
 plt.figure(1)
 plt.title('Confusion Matrix - CNN 1D Train')
sns.heatmap(ConfusionMatrix(Classification_1D, X_1D_train, y_1D_train) ,
annot=True, fmt='d',annot_kws={"fontsize":8},cmap="YlGnBu")
 plt.show()

plt.figure(2)
 plt.title('Confusion Matrix - CNN 1D Test')
sns.heatmap(ConfusionMatrix(Classification_1D, X_1D_test, y_1D_test) ,
annot=True, fmt='d',annot_kws={"fontsize":8},cmap="YlGnBu")
```

```
 plt.show()

plt.figure(3)
 plt.title('Train - Accuracy - CNN 1D')
 plt.bar(np.arange(1,kSplits+1),[i*100 for i in accuracy_1D])
 plt.ylabel('accuracy')
 plt.xlabel('folds')
 plt.ylim([50,100])
 plt.show()

plt.figure(4)
 plt.title('Train vs Test Accuracy - CNN 1D')
 plt.bar([1,2],[CNN_1D_train_accuracy,CNN_1D_test_accuracy])
 plt.ylabel('accuracy')
 plt.xlabel('folds')
 plt.xticks([1,2],['Train', 'Test'])
 plt.ylim([50,100])
 plt.show()

# Calculation and printing the average accuracy and standard deviation
 CNN_1D_train_accuracy = np.average(accuracy_1D)*100
 CNN_1D_train_std = np.std(accuracy_1D)*100
 print('CNN 1D train accuracy =', CNN_1D_train_accuracy)
 print('CNN 1D train std deviation =', CNN_1D_train_std)

# Evaluation of the accuracy of the model on the test set
 CNN_1D_test_loss, CNN_1D_test_accuracy =
Classification_1D.model.evaluate(X_1D_test, y_1D_test)
 CNN_1D_test_accuracy *= 100
 print('CNN 1D test accuracy =', CNN_1D_test_accuracy)

# Checking the data for overfitting
 if CNN_1D_test_accuracy < CNN_1D_train_accuracy:
    print('Model is overfitting the data')
 else:
    print('Model is not overfitting the data')

# Training the model 1D CNN
history = Classification_1D.model.fit(X_1D_train, y_1D_train, validation_split=0.3,
verbose=1, epochs=10)

# Plotting the training and validation accuracy/loss curves
 plt.figure(figsize=(10,5))
 plt.subplot(1,2,1)
 plt.plot(history.history['accuracy'], label='Training Accuracy')
 plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
 plt.title('Accuracy')
 plt.xlabel('Epoch')
 plt.ylabel('Accuracy')
```

```
plt.legend()
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

2D-CNN

```
# Assuming data is stored in a variable named `X`For 2D CNN
X_2D = X.reshape((-1, 44, 44, 1))

# Printing the shape of the reshaped array
print(X_2D.shape)

# Reshaping the data for - 2 dimensional feed
Input_2D = X.reshape((-1, 44, 44, 1))

# Splitting of Testing data-Training data for 2D CNN
X_2D_train, X_2D_test, y_2D_train, y_2D_test = train_test_split(Input_2D, Y_CNN,
train_size=0.7,test_size=0.3, random_state=101)

# Defination of the 2D CNN Classification model
class CNN_2D():
  def __init__(self):
    self.model = self.CreateModel()

  def CreateModel(self):
    model = models.Sequential([
        layers.Conv2D(filters=16, kernel_size=(3,3), strides=(2,2), padding
='same',activation='relu'),
        layers.MaxPool2D(pool_size=(2,2), padding='same'),
        layers.Conv2D(filters=32, kernel_size=(3,3),strides=(2,2), padding
='same',activation='relu'),
        layers.MaxPool2D(pool_size=(2,2), padding='same'),
        layers.Conv2D(filters=64, kernel_size=(3,3),strides=(2,2),padding ='same',
activation='relu'),
        layers.MaxPool2D(pool_size=(2,2), padding='same'),
        layers.Conv2D(filters=128, kernel_size=(3,3),strides=(2,2),padding ='same',
activation='relu'),
        layers.MaxPool2D(pool_size=(2,2), padding='same'),
        layers.Flatten(),
        layers.InputLayer(),
        layers.Dense(100,activation='relu'),
        layers.Dense(50,activation='relu'),
        layers.Dense(10),
        layers.Softmax()
```

```
        ])
    model.compile(optimizer='adam',
            loss=tf.keras.losses.CategoricalCrossentropy(),
            metrics=['accuracy'])
    return model

# Training the model
accuracy_2D = []
 for train, test in kfold.split(X_2D_train,y_2D_train):
   Classification_2D = CNN_2D()
   history = Classification_2D.model.fit(X_2D_train[train], y_2D_train[train],
verbose=1, epochs=15, use_multiprocessing=True)

  # Evaluation of accuracy of the model on the training set
  CNN2D_loss, CNN2D_accuracy =
Classification_2D.model.evaluate(X_2D_train[test], y_2D_train[test])
  accuracy_2D.append(CNN2D_accuracy)

CNN_2D_train_accuracy = np.average(accuracy_2D)*100
 print('CNN 2D train accuracy =', CNN_2D_train_accuracy)


# Evaluation of the accuracy of the model on the test set
 CNN_2D_test_loss, CNN_2D_test_accuracy =
Classification_2D.model.evaluate(X_2D_test, y_2D_test)
 CNN_2D_test_accuracy*=100
 print('CNN 2D test accuracy =', CNN_2D_test_accuracy)

# Saving the model for future use
 Classification_2D.model.save('models/classification_2D.model.h5')

# Plotting Confusion matrix for 2D CNN. Train, Test. Also, plotting bar graph for
Training and Testing accuracies

plt.figure(5)
 plt.title('Confusion Matrix - CNN 2D Train')
sns.heatmap(ConfusionMatrix(Classification_2D, X_2D_train, y_2D_train) ,
annot=True, fmt='d',annot_kws={"fontsize":8},cmap="YlGnBu")
 plt.show()

plt.figure(6)
 plt.title('Confusion Matrix - CNN 2D Test')
sns.heatmap(ConfusionMatrix(Classification_2D, X_2D_test, y_2D_test) ,
annot=True, fmt='d',annot_kws={"fontsize":8},cmap="YlGnBu")
 plt.show()

plt.figure(7)
 plt.title('Train - Accuracy - CNN 2D')
 plt.bar(np.arange(1,kSplits+1),[i*100 for i in accuracy_2D])
```

```
 plt.ylabel('accuracy')
 plt.xlabel('folds')
 plt.ylim([50,100])
 plt.show()

plt.figure(8)
 plt.title('Train vs Test Accuracy - CNN 2D')
 plt.bar([1,2],[CNN_2D_train_accuracy,CNN_2D_test_accuracy])
 plt.ylabel('accuracy')
 plt.xlabel('folds')
 plt.xticks([1,2],['Train', 'Test'])
 plt.ylim([50,100])
 plt.show()

# Calculation and printing of the average accuracy and standard deviation
 CNN_2D_train_accuracy = np.average(accuracy_2D)*100
 CNN_2D_train_std = np.std(accuracy_2D)*100
 print('CNN 2D train accuracy =', CNN_2D_train_accuracy)
 print('CNN 2D train std deviation =', CNN_2D_train_std)

# Evaluation of the accuracy of the model on the test set
 CNN_2D_test_loss, CNN_2D_test_accuracy =
Classification_2D.model.evaluate(X_2D_test, y_2D_test)
 CNN_2D_test_accuracy *= 100
 print('CNN 2D test accuracy =', CNN_2D_test_accuracy)

# Checking for any overfitting
 if CNN_2D_test_accuracy < CNN_2D_train_accuracy:
    print('Model is overfitting the data')
 else:
    print('Model is not overfitting the data')

# Training the model
history = Classification_2D.model.fit(X_2D_train, y_2D_train, validation_split=0.3,
verbose=1, epochs=15)

# Ploting the training and validation accuracy/loss curves for 2D CNN
 plt.figure(figsize=(10,5))
 plt.subplot(1,2,1)
 plt.plot(history.history['accuracy'], label='Training Accuracy')
 plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
 plt.title('Accuracy')
 plt.xlabel('Epoch')
 plt.ylabel('Accuracy')
 plt.legend()
 plt.subplot(1,2,2)
 plt.plot(history.history['loss'], label='Training Loss')
 plt.plot(history.history['val_loss'], label='Validation Loss')
 plt.title('Loss')
```

```
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

LSTM Model

```
# Checking for any value error
if len(X.shape) == 2:
    num_samples, num_features = X.shape
elif len(X.shape) == 3:
    num_samples, num_rows, num_cols = X.shape
    num_features = num_rows * num_cols
else:
    raise ValueError("Unexpected shape of X")

# Reshaping the data -1 dimensional feed for LSTM
Input = X.reshape([-1,5184,1])

# Splitting of Testing data-Training data for LSTM
X_train, X_test, y_train, y_test = train_test_split(Input, Y_CNN,
train_size=0.7,test_size=0.3, random_state=101)

# Defining the LSTM Classification model
class LSTM_Model():
  def __init__(self):
    self.model = self.CreateModel()

  def CreateModel(self):
    model = models.Sequential([
        layers.LSTM(32, return_sequences=True),
        layers.Flatten(),
        layers.Dense(10),
        layers.Softmax()
        ])
    model.compile(optimizer='adam',
            loss=tf.keras.losses.CategoricalCrossentropy(),
            metrics=['accuracy'])
    return model

accuracy = []

# Training the model
for train, test in kfold.split(X_train,y_train):
  Classification = LSTM_Model()
  history = Classification.model.fit(X_train[train], y_train[train], verbose=1,
epochs=15, use_multiprocessing=True)

  # Evaluating the accuracy of the model on the training set
  kf_loss, kf_accuracy = Classification.model.evaluate(X_train[test], y_train[test])
```

```
  accuracy.append(kf_accuracy)

LSTM_train_accuracy = np.average(accuracy)*100
 print('LSTM train accuracy =', LSTM_train_accuracy)

# Evaluating the accuracy of the model on the test set
 LSTM_test_loss, LSTM_test_accuracy = Classification.model.evaluate(X_test,
y_test)
 LSTM_test_accuracy*=100
 print('LSTM test accuracy =', LSTM_test_accuracy)

# Plotting Confusion matrix for LSTM. Train, Test. Also plotting bar graph for
Training and Testing accuracies

plt.figure(9)
 plt.title('Confusion Matrix - LSTM Train')
sns.heatmap(ConfusionMatrix(Classification, X_train, y_train) , annot=True,
fmt='d',annot_kws={"fontsize":8},cmap="YlGnBu")
 plt.show()

plt.figure(10)
 plt.title('Confusion Matrix - LSTM Test')
sns.heatmap(ConfusionMatrix(Classification, X_test, y_test) , annot=True,
fmt='d',annot_kws={"fontsize":8},cmap="YlGnBu")
 plt.show()

plt.figure(11)
 plt.title('Train - Accuracy - LSTM')
 plt.bar(np.arange(1,kSplits+1),[i*100 for i in accuracy])
 plt.ylabel('accuracy')
 plt.xlabel('folds')
 plt.ylim([50,100])
 plt.show()

plt.figure(12)
 plt.title('Train vs Test Accuracy - LSTM')
 plt.bar([1,2],[LSTM_train_accuracy,LSTM_test_accuracy])
 plt.ylabel('accuracy')
 plt.xlabel('folds')
 plt.xticks([1,2],['Train', 'Test'])
 plt.ylim([50,100])
 plt.show()

# Checking Model Loss on LSTM

X_train, X_test, y_train, y_test = train_test_split(Input, Y_CNN,
train_size=0.7,test_size=0.3, random_state=101)
 history = Classification.model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=15, verbose=1)
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()

# Training the model LSTM
history = Classification.model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=15, verbose=1)

# Plotting of training and validation accuracy/loss curves
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Calculation and printing the average accuracy and standard deviation
LSTM_train_accuracy = np.average(accuracy)*100
LSTM_train_std = np.std(accuracy)*100
print('LSTM train accuracy =',  LSTM_train_accuracy)
print('LSTM train std deviation =', LSTM_train_std)

# Evaluation of the accuracy of the model on the test set
LSTM_test_loss, LSTM_test_accuracy = Classification.model.evaluate(X_test,
y_test)
LSTM_test_accuracy*= 100
print('LSTM 2D test accuracy =', LSTM_test_accuracy)

# Checking for overfitting of the data
if LSTM_test_accuracy <  LSTM_train_accuracy:
    print('Model is overfitting the data')
else:
    print('Model is not overfitting the data')
```

SUPPORT VECTOR MACHINE MODEL

IMPORT FEATURE DATA INCLUDES: max value, min value, mean, variance,standard deviation,rms,skewness, ,kurtosis

```
# Importation of data for SVM
 X_Features =
scipy.io.loadmat('BearingData_CaseWestern/X_Features.mat')['Feature_Data']

# Feature data shape (no. of samples, no. of features)
 X_Features.shape

# Importation of nececcary libraries for SVM
 from sklearn.decomposition import PCA
 from sklearn.svm import SVC
 from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
 from sklearn.preprocessing import StandardScaler
 from sklearn.decomposition import PCA
 from tqdm import tqdm_notebook as tqdm
import warnings
warnings.filterwarnings('ignore')

# Fitting the given set of data
 X_Norm = StandardScaler().fit_transform(X_Features)

# Determination of shape for Y
 print(Y.shape)

# Application of PCA for smooth computation and dimensionality reduction
 pca = PCA(n_components=5)
 Input_SVM_np = pca.fit_transform(X_Norm)
 Input_SVM = pd.DataFrame(data = Input_SVM_np)
 Label_SVM = pd.DataFrame(Y, columns=['target'])

# Setup of required parameters for SVM
 parameters = {'kernel':('rbf','poly','sigmoid'),
               'C': [0.01, 1],
               'gamma' : [0.01, 1],
               'decision_function_shape' : ['ovo']}

# Support vector Machine
svm = SVC()

# Test-Train Split for SVM Model
 X_train_SVM, X_test_SVM, y_train_SVM, y_test_SVM =
train_test_split(Input_SVM_np, Y, train_size=0.7,test_size=0.3, random_state=101)

# Traing of the model to obtain the best parameters
 svm_cv = GridSearchCV(svm, parameters, cv=5)
 svm_cv.fit(X_train_SVM, y_train_SVM)

print("Best parameters = ",svm_cv.best_params_)
```

```
SVM_train_accuracy = svm_cv.best_score_*100
 print('SVM train accuracy =', SVM_train_accuracy)

# Evaluation of the accuracy of the model on the test set
 SVM_test_accuracy = svm_cv.score(X_test_SVM, y_test_SVM)
SVM_test_accuracy*=100
 print('SVM test accuracy =', SVM_test_accuracy)

# Definition of the confusion matrix for plotting
 def ConfusionMatrix_SVM(Model, X, y):
     y_pred = Model.predict(X)
  ConfusionMat = confusion_matrix(y, y_pred)
  return ConfusionMat

print(svm_cv.score(X_train_SVM, y_train_SVM))

# Plotting of different confusion matrix, bars for test,train accuracy, SVM
 plt.figure(13)
 plt.title('Confusion Matrix - SVM Train')
sns.heatmap(ConfusionMatrix_SVM(svm_cv, X_train_SVM, y_train_SVM) ,
annot=True, fmt='d',annot_kws={"fontsize":8},cmap="YlGnBu")
 plt.show()

plt.figure(14)
 plt.title('Confusion Matrix - SVM Test')
sns.heatmap(ConfusionMatrix_SVM(svm_cv, X_test_SVM, y_test_SVM) ,
annot=True, fmt='d',annot_kws={"fontsize":8},cmap="YlGnBu")
 plt.show()

plt.figure(16)
 plt.title('Train vs Test Accuracy - SVM')
 plt.bar([1,2],[SVM_train_accuracy,SVM_test_accuracy])
 plt.ylabel('accuracy')
 plt.xlabel('folds')
 plt.xticks([1,2],['Train', 'Test'])
 plt.ylim([40,100])
 plt.show()

# Plotting of Decision Boundary
 from mlxtend.plotting import plot_decision_regions
 value = 0
 width = 1
 plt.figure(17)
 plt.figure(figsize=(10.4,8.8))
 plt.title('Decision Boundary - SVM')
 plot_decision_regions(X_test_SVM, (y_test_SVM.astype(np.integer)).flatten(),
clf=svm_cv, legend=2,
                 feature_index=[0,1],
```

```
            filler_feature_values={2:value, 3:value, 4:value},
            filler_feature_ranges={2:width, 3:width, 4:width},)
```

Models Comparision

```
# Comparison of Models
 plt.figure(18)
 plt.title('Accuracy in Training data')
 plt.bar([1,2,3,4],[CNN_1D_train_accuracy, CNN_2D_train_accuracy,
LSTM_train_accuracy, SVM_train_accuracy])
 plt.ylabel('accuracy')
 plt.xlabel('folds')
 plt.xticks([1,2,3,4],['CNN-1D', 'CNN-2D' , 'LSTM', 'SVM'])
 plt.ylim([70,100])
 plt.show()


plt.figure(19)
 plt.title('Accuracy in Test data')
 plt.bar([1,2,3,4],[CNN_1D_test_accuracy, CNN_2D_test_accuracy,
LSTM_test_accuracy, SVM_test_accuracy])
 plt.ylabel('accuracy')
 plt.xlabel('folds')
 plt.xticks([1,2,3,4],['CNN-1D', 'CNN-2D' , 'LSTM', 'SVM'])
 plt.ylim([70,100])
 plt.show()
```

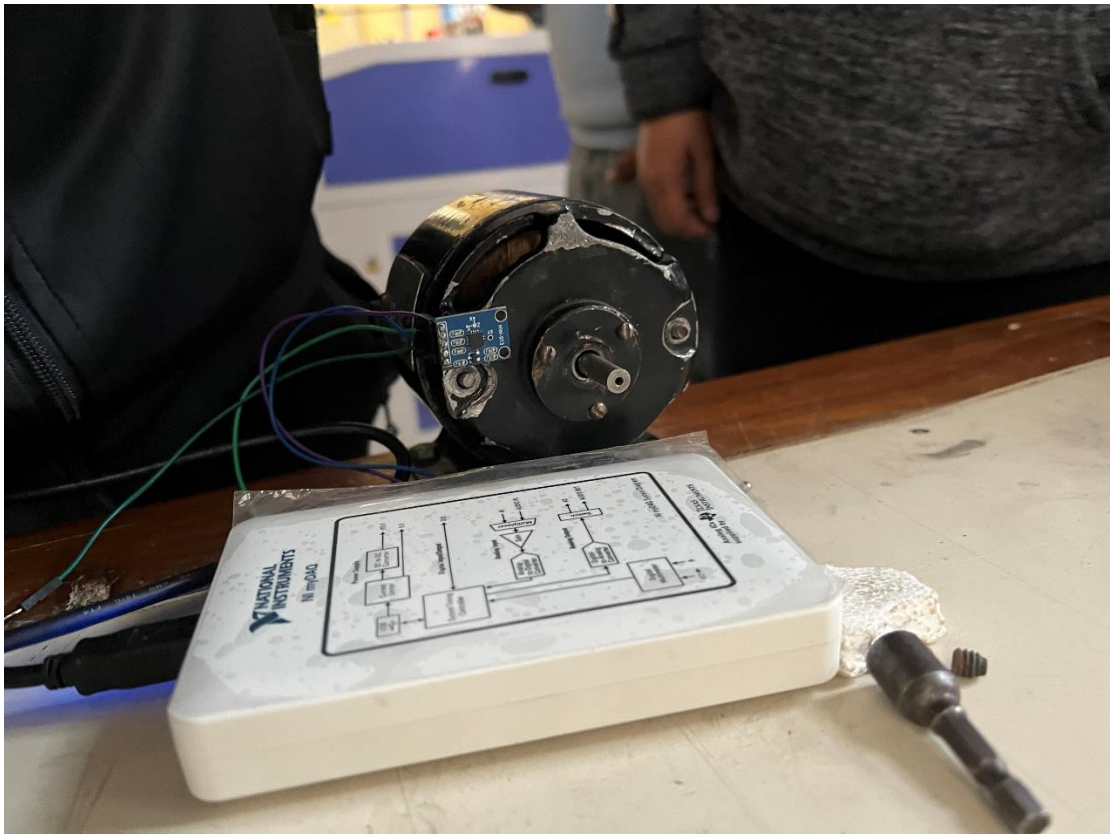

Figure A2. 1: Accelerometer, attached to the drive end connected with the DAQ system

Figure A2. 2: The Project Team and the Experimental Setup