DE-42 (DC & SE)

NUST COLLEGE OF
ELECTRICAL AND MECHANICAL ENGINEERING

Edge-enabled Fault Prediction of Motor Bearings

A PROJECT REPORT

DE-42 (DC & SE)

Submitted by

NS AAZAIN UMRANI

NS MUHAMMAD ADEEL INTIZAR

NS MUHAMMAD USMAN

NS ABDUL REHMAN

BACHELORS

IN

COMPUTER ENGINEERING

YEAR

2024

Project Supervisor

DR. SAJID GUL KHAWAJA

YEAR 2024

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,
ISLAMABAD, PAKISTAN

Certification

This is to certify that Aazain Umrani 350163, Muhammad Adeel Intizar 353152, Abdul Rehman 334937 and Muhammad Usman 333855 have successfully completed the final project edge-enabled fault prediction of motor bearings, at the NUST CoEME, to fulfill the partial requirement of the degree Computer Engineering.

Sajid
Signature of Project Supervisor
Dr. Sajid Gul Khwaja
Associate Professor

Sustainable Development Goals (SDGs)

SDG No	Description of SDG	SDG No	Description of SDG
SDG 1	No Poverty	**SDG 9**	**Industry,Innovation, and Infrastructure**
SDG 2	Zero Hunger	SDG 10	Reduced Inequalities
SDG 3	Good Health and Well Being	SDG 11	Sustainable Cities and Communities
SDG 4	Quality Education	SDG 12	Responsible Consumption and Production
SDG 5	Gender Equality	SDG 13	Climate Change
SDG 6	Clean Water and Sanitation	SDG 14	Life Below Water
SDG 7	Affordable and Clean Energy	SDG 15	Life on Land
SDG 8	Decent Work and Economic Growth	SDG 16	Peace, Justice and Strong Institutions
		SDG 17	Partnerships for the Goals

1 NO POVERTY
2 ZERO HUNGER
3 GOOD HEALTH AND WELL-BEING
4 QUALITY EDUCATION
5 GENDER EQUALITY
6 CLEAN WATER AND SANITATION
7 AFFORDABLE AND CLEAN ENERGY
8 DECENT WORK AND ECONOMIC GROWTH
9 INDUSTRY, INNOVATION AND INFRASTRUCTURE
10 REDUCED INEQUALITIES
11 SUSTAINABLE CITIES AND COMMUNITIES
12 RESPONSIBLE CONSUMPTION AND PRODUCTION
13 CLIMATE ACTION
14 LIFE BELOW WATER
15 LIFE ON LAND
16 PEACE, JUSTICE AND STRONG INSTITUTIONS
17 PARTNERSHIPS FOR THE GOALS
SUSTAINABLE DEVELOPMENT GOALS

Sustainable Development Goals

Complex Engineering Problem

Range of Complex Problem Solving

	Attribute	**Complex Problem**	
1	Range of conflicting requirements	Involve wide-ranging or conflicting technical, engineering and other issues.	X
2	Depth of analysis required	Have no obvious solution and require abstract thinking, originality in analysis to formulate suitable models.	X
3	Depth of knowledge required	Requires research-based knowledge much of which is at, or informed by, the forefront of the professional discipline and which allows a fundamentals-based, first principles analytical approach.	X
4	Familiarity of issues	Involve infrequently encountered issues	
5	Extent of applicable codes	Are outside problems encompassed by standards and codes of practice for professional engineering.	X
6	Extent of stakeholder involvement and level of conflicting requirements	Involve diverse groups of stakeholders with widely varying needs.	
7	Consequences	Have significant consequences in a range of contexts.	
8	Interdependence	Are high level problems including many component parts or sub-problems	X

Range of Complex Problem Activities

	Attribute	**Complex Activities**	
1	Range of resources	Involve the use of diverse resources (and for this purpose, resources include people, money, equipment, materials, information and technologies).	X
2	Level of interaction	Require resolution of significant problems arising from interactions between wide ranging and conflicting technical, engineering or other issues.	
3	Innovation	Involve creative use of engineering principles and research-based knowledge in novel ways.	X
4	Consequences to society and the environment	Have significant consequences in a range of contexts, characterized by difficulty of prediction and mitigation.	
5	Familiarity	Can extend beyond previous experiences by applying principles-based approaches.	X

Dedicated to all faculty members of Department of Computer and Software Engineering for their endless support.

Acknowledgment

First of all, Praise be to Allah for granting us the courage and faith to persevere and for supporting us at every turn. Our FYP has now been completed, and we are grateful to Him for this.

Second, we would like to express our sincere gratitude to Sajid Gul Khawaja, our supervisors, for his. invaluable assistance with every single problem. His support and direction gave us the willpower to pursue our goals. We are grateful for the significant role you have played in our life, sirs. We will always remember it.

Finally, we would like to express our gratitude to our parents and friends, without whose unwavering support and unceasing inspiration we might not have been able to finish our senior project. They were invaluable to us on our journey, and we will always be grateful to them. Their unwavering encouragement spurred us on to greater heights and gave us hope when we had none left in us.

Abstract

Our project focuses on implementing a comprehensive solution to reduce the risk of major breakdowns in electric motors caused by bearing faults. At its core, we developed an innovative edge cog device that uses vibration sensors for early prediction of different bearing faults in motors. The collected vibration signal data can give us a better knowledge of the various faults happening in mechanical systems. In this work, we analyze vibration signal data by integrating several signal processing methods and connecting them with machine learning approaches to categorize various types of bearing failures, therefore avoiding considerable downtime and costly repairs. The procedure begins with these sensors gathering real-time vibration data, which is then analyzed at the edge node using various signal processing algorithms to detect bearing faults. This data is then transferred to the cloud, where a deep learning neural network made up of 1-D convolution, pooling, and dense layers, and trained on a variety of datasets containing both healthy and defective bearings is examined to precisely determine the fault type. Furthermore, we provide a user-friendly dashboard that enables real-time monitoring and notifications, allowing maintenance teams to resolve any irregularities as soon as they are noticed. By combining these advanced techniques, our edge device not only prevents catastrophic failures but also promotes timely maintenance plans and operational efficiency in general.

Keywords: Motor Bearings, Fault Predictions, CWRU,Predictive maintenance, Artificial Intelligence, Edge Computing

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The rationale for this project can be regarded as a response to the necessity of increasing the dependability of industrial processes by solving the problem of bearing faults in motors. Issues arising from bearing faults are one of the leading sources of motor destruction, which in turn results in hefty losses and system downtime. Logically, bearing faults are regarded as one of the primary causes of this type of loss, which, overall, amounts to approximately \$50 billion globally annually[1] . These failures are found to be very costly and therefore we need to minimize these kinds of costs in the industries by building a solution to predict and fix them before they happen. Consequences of bearing faults may result in excessive time that is wasted and expensive repairs that hinder production flow. Our project aims at reducing these interferences since the quicker the discovery of a faulty line, the less the interruption that affects operations and pushes up productivity. This is because the early detection and prevention of bearing faults are very important to the enhancement of industrial safety and the reduction of environmental impacts on health. The arrival of Industry 4. 0 has introduced improved sensors and data acquisition systems on the production floors creating huge data sets. However, the difficulty is to efficiently organize this information to obtain valuable and useful knowledge [2]. That is why, the proposed project applies state-of-the-art approaches including edge computing and deep learning to utilize the potential of this data for real-time diagnostics and decision-making. The typical centralized processing of faults is ineffective due to temporal latency and/or bandwidth limitations. That is why there is a need to use decentralized solutions that process data at the place where they are collected, which allows to significantly increase the speed of fault detection. To mitigate these challenges, hence, the proposed edge computing will offer real-time data analysis and insights. The applications that are typically found in the industrial setting demand solutions that are both efficient in terms of scalability and reasonably priced. While high-performance computing platforms are quite useful, they are quite costly and thus impractical for universal use. Expenses are also an essential factor to consider when it comes to solving a 2 complex problem because it is imperative that the machine-learning model we propose can be made accessible to a wide variety of industries. The use of predictive maintenance strategies is vital in the current industrial environments because it offers an opportunity to avoid equipment failures. Through the utilization of specialty diagnostics and continuous data acquisition, our project contributes to the shift from corrective to preventive maintenance and thus, improves the overall maintenance approach. In other words, the rationale for undertaking this project is centered on loss avoidance, productivity gains, safety and reliability increments, utilization of innovative techniques, processing of data, scalability and affordability, and maintenance prognostication. Addressing these difficulties, it is our mission to change the approach to fault detection and maintenance in industrial environments to help make it more dependable and effective.

1.2 Problem Statement

Motor bearings are critical components found in all rotating electrical machinery and they cause catastrophic motor failures in industrial applications thus; higher maintenance costs and loss of time. A survey into the causes of unplanned downtime shows that the study industry loses $50 billion every year, and equipment failure accounts for 42% of all cases[1].

This type of failure is common, and in most of the cases, it leads to permanent failure if the root cause, such as bearing faults, is not identified on time. One of the main challenges with bearings is the inability to access them during the operation and make preliminary fault diagnosis. However, if the fault advances, the motor as well as related equipment can be severely damaged and in most cases costly repairs, or replacements may be necessary. Operation and maintenance costs rise sharply when grave faults are evident, deepening the problems of high costs and low efficiency.

Spectral features of vibration signal data have been found to be significant and more credible to diagnose the health condition of machines. However, the issue emanates from the way that this kind of data can be analyzed in a way that early faults can be detected. Acoustic signals are always cluttered and hidden by noise or related operational changes. In many cases traditional methods of fault detection may not be able to pick the subtle symptoms of bearing faults and the data generated by industries is too large that these techniques are not designed for processing large amounts of data which is characteristic for the contemporary industrial processes. Furthermore, there is a demand for solutions that are not only effective but also cost-effective and scalable.

In conclusion, the key obstacles in preventing catastrophic motor failures owing to bearing issues are:

1. Bearing Inaccessibility: Difficulty diagnosing issues early because bearings are inaccessible during operation.

2. Vibration Signal Complexity: Effective defect identification using complicated and noisy vibration signal data.

3. Centralized Processing Delays: Delays in fault detection caused by centralized data processing and response delays.

4. Cost and Scalability: There is a need for cost-effective and scalable solutions that can be extensively implemented in industrial situations.

Addressing these difficulties is crucial for decreasing unexpected downtime, saving maintenance costs, and enhancing the overall dependability and efficiency of industrial processes. [2]

1.3 Scope

The scope of this project is to create, build, and validate an enhanced failure detection system for motor bearings using edge computing and deep learning technologies. The goal is to improve the reliability and efficiency of industrial processes by focusing on essential components of fault detection, data processing, and maintenance techniques. Major components and deliverables include the development of a small, cost-effective edge computing device using powerful vibration sensors, integrating data acquisition systems for continuous monitoring and recording of vibration signal data from motor bearings. Signal processing and data analysis will involve preparing vibration data at the edge node and using empirical mode decomposition and zero crossings to extract significant characteristics that will be sent to Azure cloud. A deep learning model will be developed and trained on Azure cloud, featuring 1-dimensional convolutional layers, pooling processes, and densely coupled layers, using extensive datasets of both normal and faulty bearings for precise fault classification. The model will incorporate strategies to enhance flexibility and accuracy under various operational conditions and noise levels. A user-friendly web application will be developed for real-time tracking of motor health, with alert systems to notify maintenance crews of abnormalities for timely intervention. Finally, predictive maintenance algorithms will be created to detect and prevent equipment faults before they occur, integrating the defect detection system with existing maintenance management systems to enhance overall maintenance strategy.

1.4 Aims and Objectives

The primary aim of this project is to create an enhanced fault detection and maintenance system for motor bearings by utilizing edge computing and deep learning technologies. This technology aims to improve the reliability, safety, and efficiency of industrial processes by allowing for early detection and proactive maintenance of bearing defects. The final objectives to achieve are stated below:

- Design a test rig for bearing fault data collection.
- Design a edge node for edge processing .
- Develop a cloud and edge-enabled solution for bearing fault detection.
- Design and implement algorithms for signal processing and machine learning to detect and classify faults.

1.5 Outcomes

Following are the outcomes achieved in our project:

1. A edge node that will predict the faults of motor bearings in real time.
2. Developing a test rig for data collection of both faulty and normal bearings.
3. A deep learning model trained on both data sets faulty and normal for classification of faults
4. A user friendly web application that can be used for monitoring and proactive maintenance

1.6 Report Organization

The organization of the thesis is as follows:

- Chapter 2, it mainly explains the faults in motor bearings and the related approcahes that are used to tackle failures.
- Chapter 3, it deals with the design and development of test rig and edge cog device and the tools and components that are used.
- Chapter 4, it deals with the methodology that we are using to detect the faults including both the edge processing and Machine Learning models.
- Chapter 5, it deals with deployment of our project on cloud and web application. It also states the the final results and accuracies.
- Chapter 6, It comprises of concluding the report and examining future prospects and directions for the project.

Chapter 2

Background & Related Work

2.1 Overview

This chapters covers the related works have been done related to faults prediction of motor bearings.It starts with descriptions of different types of errors, such as normal , inner race faults, ball bearing faults, and outer race faults, along with their causes, and prediction procedures. Following that, the chapter describes related research by several researchers who have used signal processing and deep learning approaches to forecast and detect motor bearing faults.

2.2 Error Types

2.2.1 Normal

When a bearing is operating at its best, there is little to no friction as the balls or rollers move between the inner and outer races, and there are no wear or damage indicators on the bearing. The proper amount of lubrication ensures that there is no contamination or misalignment and that excessive heat buildup is prevented. The bearing supports the motor shaft and enables it to rotate with the least amount of resistance while functioning silently and effectively. Vibration analysis and thermography are two predictive maintenance methods that can be used to track the state of a bearing and stop possible problems.

2.2.2 Inner Race

Inner race faults arise from defects such as spalling, pitting, or cracking in the inner race of the bearing, which is in direct contact with the motor shaft. These malfunctions are frequently brought on by heavy loads, shoddy installation, or inadequate lubrication. If left unchecked, the inner race's degradation can result in greater vibration and noise, decreased efficiency, and possibly catastrophic failure. The balls or rollers may roll unevenly as a result of surface damage on the inner race, which could further deteriorate the bearing and its surrounding parts. Inner race problems can be found early with the aid of routine surface irregularity inspection and abnormal vibration pattern monitoring. Early detection of spalling or subsurface cracks can be achieved with sophisticated diagnostic instruments like as ultrasonic testing.

2.2.3 Ball Bearing

Ball bearing defects are mostly related to the balls inside the bearing, which may experience wear, distortion, or pitting. These malfunctions are frequently caused by wear, contamination, or inadequate lubrication. Damage to the balls prevents them from rolling smoothly between the races, which increases noise, vibration, and friction. Ball bearing malfunctions have the potential to raise operating temperatures and hasten the wear of the balls and races over time. If this is not corrected right away, it may impair the bearing's ability to support loads and eventually result in motor failure. Ball bearing failures can only be avoided by routine maintenance and inspections, which also involve making sure that the lubricants are clean and that the loading conditions are right.

2.2.4 Outer Race

The stationary portion of the bearing that contacts with the motor housing is called the outer race, and damage to it means an outer race defect. Excessive loading, contamination, or insufficient lubrication can all lead to outer race faults, same as inner race problems. The presence of strange noises, elevated vibration, and maybe elevated operating temperature are some of the symptoms. A further worsening of the issue might result from misalignment of the bearing and uneven wear patterns caused by outer race problems. Motor failure must be avoided by early fault detection using vibration analysis and routine maintenance. It is possible to identify impurities that may be causing outer race wear using methods like oil analysis. Outer race fault risk can be reduced by maintaining a clean working environment and using appropriate bearing installation.

2.3 Related Work

Many academics have argued that signal processing and deep learning techniques should be used to forecast and identify different kinds of defects in motor bearings using motor vibrations in order to avoid the arduous nature of standard fault detection techniques. Thomazella et al. used digital signal processing techniques such as short-time Fourier transform (STFT) and the ratio of power (ROP) to extract features from vibrations signals captured to monitor chatter phenomenon during the grinding process [3]. Zoltan et al., have demonstrated that signal processing techniques such as Discrete Wavelet and Wavelet Packet Transform are effective in extracting features from the frequency domain for fault detection [4]. Sun W et al. extract the distinctive spectrum of rolling bearing vibration data by combining envelope analysis and discrete wavelet transformations. The various rolling bearing working conditions are then determined by applying a spectrum cross correlation coefficient[5]. The outcomes show that these deep learning methods are useful for investigating the long-term trends of the observed signals in addition to being able to forecast problems.

A few of the most accurate deep learning architectures, as well as several bearing failure classification datasets and signal feature extraction methods, are covered by Neupane et al. [6]. Among deep learning approaches, convolutional neural networks (CNNs) are most widely used architectures. A hierarchical adaptive deep convolution network is suggested by Guo et al. for fault size prediction. In their study, they employ CNNs to distinguish between several types of bearing errors after converting the signal data into a 32x32 array[7]. In order to capitalize on the signal data, Pan et al. used one dimensional convolutional neural network (1D CNN) and long short term memory (LSTM). In their article, they merged 1DCNN and LSTM into a single, unified structure by using the CNNs' output as the LSTM's input to identify the different types of bearing faults.[8]. Additionally, they examined how well nine distinct featurization methods performed in combination with various conventional machine learning algorithms. Pan et al. did not, however, use stacked median and mean channels in their investigation. Rather, they combined Long Short-Term Memory (LSTM) networks with Convolutional Neural Networks (CNN) to create a more computationally demanding framework.
Another method combining CNN and evidence fusion based on the Dempster-Shafer theory was put forth by Li et al. The study showcased the method's flexibility in handling different load scenarios and revealed an impressive accuracy rate of 98.92%.[9]. The CWRU dataset was used in a study by Hendriks et al. [10] to benchmark bearing fault diagnostics. To solve domain shift issues, they put forth a novel convolutional neural network (CNN)-based benchmarking technique. Using this method, independent sets of bearings were created for the training and testing datasets. Their method's outcomes were contrasted with those of other cutting-edge deep CNN-based techniques. An automatic feature-learning neural network for bearing defect diagnostics was proposed by Chen et al. [6]. In order to process the input data, this approach used two Convolutional Neural Networks (CNN) with various kernel sizes. They used a hierarchical Long Short-Term Memory (LSTM) network, which can extract intrinsic properties from vibrational signals, for the final fault classification. Prior to being fed into the CNNs, the signals were

downscaled to improve computational efficiency. By operating at distinct receptive fields, the two CNN layers enabled a more thorough examination of the input data. With an accuracy of 98.46%, our strategy outperformed a number of cutting-edge techniques. A physics-based deep learning method that combines a threshold model with convolutional neural networks (CNN) for fault identification was presented by Shen et al. [11]. By using the physics that underlie bearing faults, the threshold model evaluates the state of the bearings. After that, automatic feature extraction and fault categorization are carried out by the CNN. The implementation of a customized loss function for the CNN, intended to selectively magnify the influence of the physical insights acquired from the threshold model, is a crucial novelty in their methodology. Combining these two improves the model's capacity to recognize and categorize bearing issues with accuracy.

2.4 Conclusion

This chapter described different types of faults in the motor bearings like inner race, outer race, ball faults as well as how these faults occur in bearings.Furthermore, the chapter examined major related work in the field, demonstrating how signal processing and deep learning approaches were used to improve faults detection. The chapter discussed the successful use of these technologies in forecasting and diagnosing bearing faults.

Chapter 3

Material & Component

3.1 Overview

This chapter covers the components and technologies used in this project. It begins by covering the specifications of motor and its parameters. The chapter then discusses the function, features, and components of ball bearings. It then covers the piezoelectric vibration sensor, including its operating principles and specs, before moving on to an overview of the esp32 and its characteristics. The chapter also discusses the Case Western Reserve University (CWRU) dataset used to validate the algorithms, as well as the software tools utilized.

3.2 Hardware

The hardware component in this device consists of designing a rig that will bind all the components. This involves mounting the motor, bearings and vibration assembly in the rig and aligning the motor and bearings and any other assembly parts in order to make the rig functional. The rig is intricately designed and engineered in such a way that all the internal components of the equipment are enclosed and assembled in a way that allows the parts proper functioning.

3.2.1 Rig Design

At one side of the base there is a motor with an extended rotating shaft. The shaft is mounted with three ball bearings. These ball bearings are placed within clamps which tend to keep them fixed so that there can be no movement as the shaft rotate. Each bearing is attached with sensors to obtain information about the ball bearing vibrations. Figure 3.1 shows the test rig implemented for this project.

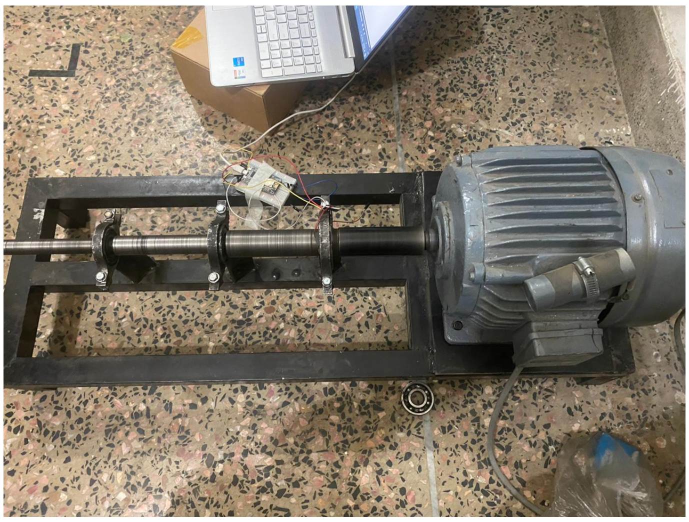


Figure 3.1: Test Rig with motor on it having extended shaft on which three bearings of different diameters are fixed through clamps

3.2.2 Motor Design

3.2.2.1 Function

These motors fall under the category of AC motors, specifically induction or asynchronous motors. They take power to the rotor through electromagnetic induction and thus do not require a direct connection. These motors are quite robust and this makes it ideal to use in industries and commercial building. Well, they do not have brushes it is much easier to maintain them. They are managed in velocity with an element known as a variable frequency drive.

3.2.2.2 Motor Construction

Motor comes fitted with a cast iron housing that serve to lengthen its life span, and reduce on chances of a breakdown. On the inside, one has a rotor as well as a stator which are made of electrically coated silicon steel sheets. It includes ball bearings at the corners of the motor, they are quality bearings which are greased as the motor is going to be used. The rotor is made to run on the balanced system and the outer cover of the motor is very beautifully constructed and painted with a special type of paint so that they can never be subjected to rusting. The fan-cooling system also proves that the machines are enclosed and can be used in numerous applications. Figure 3.2 shows the water pump motor used in this project with required specifications given in Table 3.1.

Figure 3.2: Water pump motor

Specification	Value
HP	1
KW	0.74
V	400 440
Phase	3
A	1.75
RPM	1500
Hz	50
Ambient Temp	-15°C – +40°C
Relative Humidity	60% (@ +40°C)
Altitude	Less than 1000 meters above sea level

Table 3.1: Specifications of water pump motor

3.2.3 Ball Bearings

3.2.3.1 Purpose

The balls bear their essential role to minimize friction in the course of rotation and carry radial loading and some amount of the axial loading, if necessary. This it does by establishing at least two circular structures known as races, to contain the balls. Some balls take the load and make the parts to move freely. Traditionally, one race is stationary while the other is rotating around a shaft, hub or cylinder-like core. Switching the moving race makes the balls roll When the moving race gets exchanged. Just like when two flat surfaces move over each other, the rolling balls produce much lesser friction than any other object.

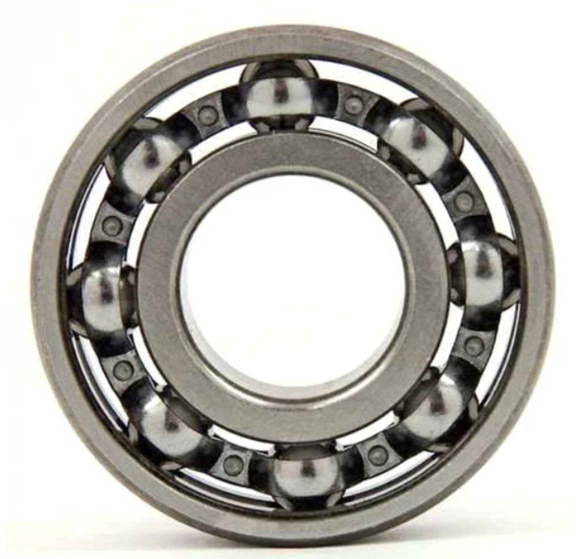


Figure 3.3: Ball Bearing

3.2.3.2 Characteristics

Ball bearings normally have a lower load carrying capacity than other type of bearings of similar dimensions as the effective load bearing area is only between the balls and the races. Nevertheless, it is possible to endure minor angular mismatch between the inside and outside loci of the thrust wedge. This makes them very usable in many systems where it’s important to have steady and very precise movement.

3.2.3.3 Parts

Ball

The ball is the rolling element inside the bearing that reduces friction and carries the load. It moves between the inner and outer races, allowing smooth rotation.

Inner Race

The inner race is the inner ring of the bearing that usually attaches to the rotating part, like a shaft. It has a smooth, curved surface for the balls to roll on.

Outer Race

The outer race is the outer ring of the bearing that stays stationary or is attached to the housing. It also has a smooth, curved surface for the balls to roll on.

Figure 3.4 shows all the inner parts of ball bearing;

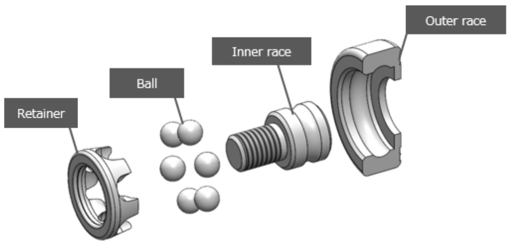


Figure 3.4: Bearing parts (a)-Reatiner, (b)-Ball, (c)-Inner Race and (d)-Outer Race

3.2.4 Piezoelectric Vibration Sensor

Piezoelectric Vibration Sensor is used to monitor the vibrations. By connecting to the A0 terminal of the controller, the piezoelectric vibration sensor module enables the controller to monitor the vibration values from the serial port. The sensor is perfect for interactive projects like electronic drums since it works with specific expansion boards and uses analog ports to detect even weak electrical signals. A potentiometer is a small dial that is used to control the sensitivity of the sensor. It increases in response to higher vibrations. The sensor can provide useful information for a variety of applications by interpreting the vibrations' frequency and strength, guaranteeing precise and responsive interaction based on the results.Figure 3.5 shows the Piezoelectric vibration sensor module and Table 3.2 shows the specifications, inputs and outputs of this sensor.

Specification	**Value**
Voltage	5.0V (DC)
Interface Type	AD/DO
AO	Analog Signal Output
DO	TTL Level Output
Temperature	-10 °C +70 °C
Size	Approx. 20 x 20mm / 0.8 x 0.8in
Weight	4.5 g / 0.2 oz (approx.)
Test Method and Wiring:	
G (GND)	Power ground
V (VCC)	Operating voltage 5V (DC)
ADO	Analog signal output
Frequency	33 Hz

Table 3.2: Specifications of Piezoelectric vibration sensor module

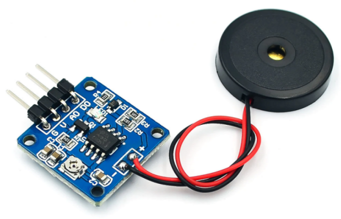


Figure 3.5: Piezoelectric vibration sensor module

3.2.5 esp32

A family of low-cost, energy-efficient microcontrollers with integrated Bluetooth and Wi-Fi is called esp32. These microcontrollers can perform a variety of jobs, much like miniature computers. They come in several versions and have a main CPU that can be single or dual-core. In order to keep everything functioning properly, the esp32 contains additional functions including power management tools, amplifiers for stronger signals, and antenna switches. Shanghai, China-based Espressif Systems is the firm that designed the esp32. TSMC uses cutting-edge technologies to make it. The esp32 is a refined variant of the esp8266, an earlier microcontroller.

Table 3.3 shows the pins and features of esp32 and Figure 3.6 shows the microcontroller esp32.

Features	**Value**
Memory	
RAM	520 KB
ROM	448 KB
Wireless Connectivity	
Wi-Fi	802.11 b/g/n
Bluetooth	v4.2 BR/EDR and BLE
Peripheral Interfaces	
GPIOs	34 × programmable
SAR ADC	12-bit up to 18 channels
DACs	2 × 8-bit
Touch Sensors	10 × (capacitive sensing GPIOs)

Table 3.3: Features of esp32- pins, memory, interfaces and connectivity

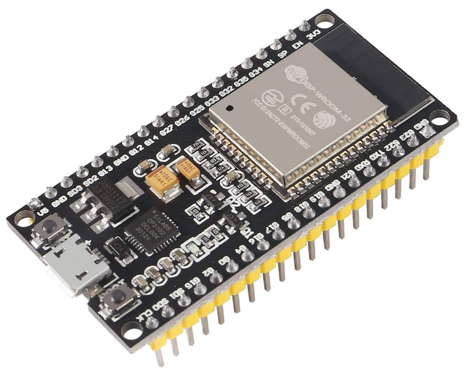


Figure 3.6: Espressif32

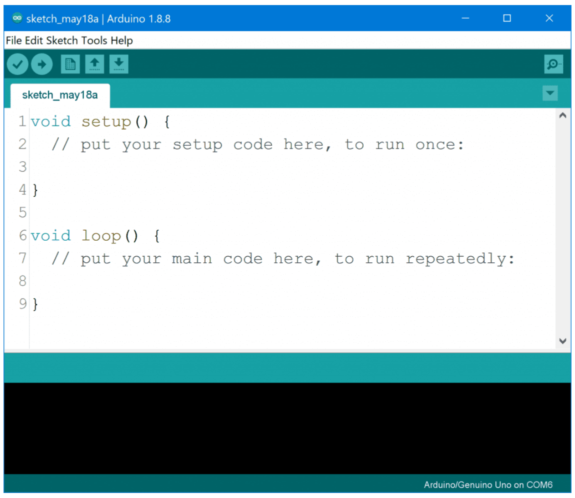
3.3 Dataset

For validation of our algorithm that we will run on our edge device and the models that we will deploy on cloud, the bearing dataset of Case Western Reserve University **(CWRU)** was used. This data is contain normal and faulty signals along with the subclasses of each faults. The faults were Inner Race, Ball Bearing and Outer Race and each fault has subclass of fault of depth of 0.007, 0.014 and 0.021 inches.[12].

3.4 Software

3.4.1 Arduino IDE

The Arduino IDE is a user-friendly tool for developing and uploading code to Arduino boards. It is perfect for novices and supports both C and C++. It has a code editor, messaging area, and a prewritten code library to make adding new features easier. The Figure 3.7 shows the homepage of arduino IDE.


sketch_may18a | Arduino 1.8.8
File Edit Sketch Tools Help
sketch_may18a
Arduino/Genuino Uno on COM6

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Figure 3.7: Homepage of Arduino IDE

3.4.2 Matlab

MathWorks' MATLAB is a powerful tool for data analysis, algorithms, and visualizations. It includes many built-in functions, is excellent at matrix math, and provides toolboxes with specific applications for many professions. It requires a large amount of resources and an expensive license, but it interfaces with Python and Arduino. Logo of MATLAB is shown in the Figure 3.8.

Figure 3.8: Logo of MathWorks' MATLAB

3.4.3 Python

Python is a well-liked and simple-to-learn programming language that may be used for automation, scientific computing, data analysis, AI, and web development. It features a large library, community assistance, and support for several programming languages. Logo of Python is shown in the Figure 3.9.

Figure 3.9: logo of python

3.4.4 Tensorflow

TensorFlow, by Google Brain, is an open-source machine learning framework. It provides low-level operations and high-level API support together with model creation tools and libraries. TensorFlow operates on platforms ranging from mobile phones to huge systems, making it perfect for research and production. Logo of Tensorflow is shown in the Figure 3.10.

TensorFlow

Figure 3.10: Logo of TensorFlow

3.4.5 Google Colab

You can run Python code on your web browser using Google Colab, a free cloud-based Jupyter notebook. It is perfect for data science and machine learning because it works with frameworks like PyTorch and TensorFlow, supports GPUs and TPUs, and connects with Google Drive. Logo of Google Colab is shown in the Figure 3.11.

Google Colaboratory

Figure 3.11: logo of Google Colab

3.4.6 Visual Studio Code

VS Code is a text editor that supports multiple languages, code compilation, debugging, and Git integration. Logo of Visual Studio Code is shown in the Figure 3.12.

Figure 3.12: Visual Studio Code

Chapter 4

Methodology

4.1 Overview

This chapter covers the methods that we have used to classify the faults found in motor bearings, including inner race faults, outer race faults, and ball bearing defects. We have integrated piezoelectric sensor with each bearing on extended shaft, to monitor vibration patterns. The sensors collect the data that is initially processed on a local edge device, which performs initial analysis to evaluate the condition of the bearings. If the bearings are non-faulty their status is directly displayed on the dashboard in real time. However, If a fault is detected, the data is sent to the Azure SQL database on the cloud for further fault classification. Here our machine learning algorithm analyzes the data to precisely determine the type and severity of the fault. The type is displayed on the dashboard and then maintenance can be scheduled. As depicted in Figure 4.1.Furthermore, The details are explained below.

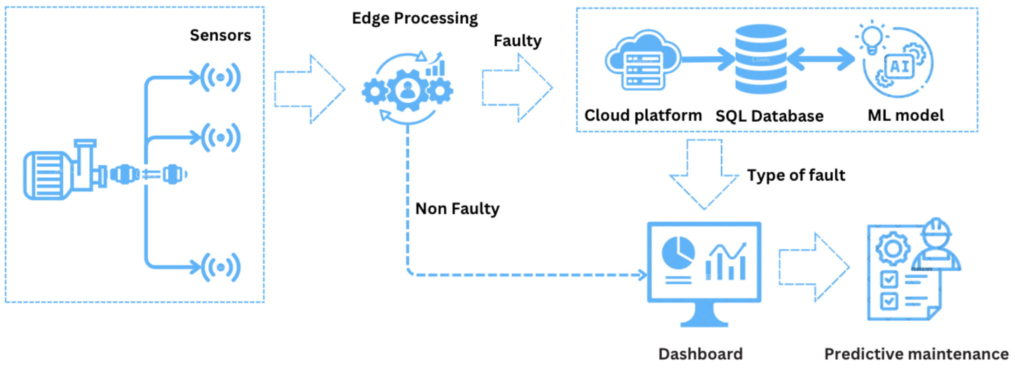


Figure 4.1: System level diagram explaining the workflow

4.2 Edge Processing

Initially, the vibrations that are collected from bearings through piezoelectric sensor are processed on esp32. A window of 2000 samples are made in which the vibrations data is stored. This window acts as a batch of data that will be processed at one time.

Once the window is filled with 2000 samples, the data is passed through a filter function. This filter removes any unwanted noise or frequencies above 15Hz, which helps in obtaining a cleaner signal. After filtering, the data is normalized to a range between -1 and 1. Normalization adjusts the values so they are within a consistent range, making it easier to analyze the data accurately.

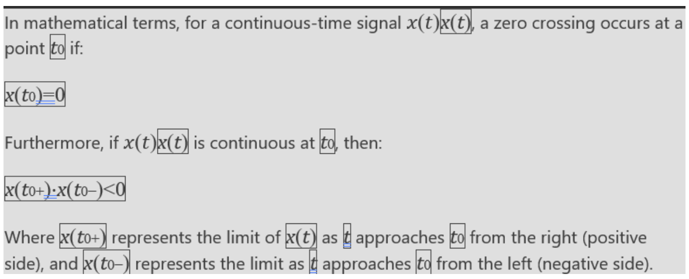
Next, the normalized data undergoes a zero-crossing check. This involves counting how many times the data values cross from negative to positive or from positive to negative. This count is crucial as it helps determine the condition of the bearing. A predefined threshold is set, and if the zero-crossing count exceeds this threshold, it indicates that the bearing is faulty. The result of this zero-crossing analysis is then sent to the cloud for further monitoring and analysis.

After processing, the window of 2000 samples shifts to the left. The first 1000 samples are discarded, and the next 1000 samples move to the initial position. This allows space for 1000 new samples to be recorded. The process then repeats, with the new data being collected and analyzed in the same manner. This continuous cycle ensures that the vibration data is constantly monitored, providing real-time information about the condition of the bearing.

4.3 Zero Crossing

In signal processing and analysis, zero crossing is a widely utilized concept, especially when analyzing periodic or oscillatory signals. It describes the locations in a signal where the signal's value crosses the axis of zero amplitude. To put it another way, it represents the times that the signal flips from positive to negative or vice versa.
Zero crossing is a concept commonly used in signal processing and analysis, particularly in the analysis of periodic or oscillatory signals. It refers to the points in a signal where the value of the signal crosses the zero amplitude axis. In simpler terms, it indicates the instances where the signal changes polarity, transitioning from positive to negative or vice versa.

The Figure 4.2 shows the steps of zero crossing


In mathematical terms, for a continuous-time signal $x(t)$$\boxed{x(t)}$, a zero crossing occurs at a point $\boxed{t_0}$ if:

$$\boxed{x(t_0)=0}$$

Furthermore, if $x(t)$$\boxed{x(t)}$ is continuous at $\boxed{t_0}$, then:

$$\boxed{x(t_{0+})\cdot x(t_{0-})<0}$$

Where $\boxed{x(t_{0+})}$ represents the limit of $\boxed{x(t)}$ as $\boxed{t}$ approaches $\boxed{t_0}$ from the right (positive side), and $\boxed{x(t_{0-})}$ represents the limit as $\boxed{t}$ approaches $\boxed{t_0}$ from the left (negative side).

Figure 4.2: Equations of zero crossing

Similar methods are used to identify zero crossings in discrete-time signals, such as digital signals sampled at discrete intervals. When neighboring samples of a signal have opposite signs, this is known as a zero crossover.

4.4 Empirical mode decomposition (EMD)

A data-driven, adaptive signal processing method called empirical mode decomposition (EMD) is used to analyze non-stationary, non-linear time series data. It is especially helpful for breaking down signals into a finite collection of intrinsic mode functions (IMFs), each of which stands for a distinct oscillatory mode or signal component.
The following are the main steps in the EMD process:

4.4.1 Finding Extrema

Finding the signal's local maxima and minima is the first stage in EMD. The upper and lower envelopes of the signal are constructed using these extrema points as a foundation.
Equation:

$$\text{Extrema Points:} \quad x_{\text{max}} \quad \text{and} \quad x_{\text{min}} \tag{4.1}$$

4.4.2 Interpolating Envelopes

The upper and lower envelopes of the signal are constructed using cubic spline interpolation, which is applied after the extrema have been determined. These envelopes provide the variance in the signal a continuous, smooth representation.
Equation:
Upper Envelope:

$$U(t) \tag{4.2}$$

Lower Envelope:

$$L(t) \tag{4.3}$$

4.4.3 Finding the Mean

The mean envelope is obtained by computing the mean of the upper and lower envelopes.
Equation:

Mean Envelope:

$$M(t) = \frac{U(t) + L(t)}{2} \tag{4.4}$$

4.4.4 First IMF (Intrinsic Mode Function) extraction

The mean envelope is subtracted from the original signal to get the first IMF. The signal's high-frequency oscillatory component is represented by this IMF.
Equation:
First IMF:

$$\text{IMF}_1(t) = x(t) - M(t) \tag{4.5}$$

4.4.5 Repeat Process

The generated IMF is subjected to steps 1 through 4 repeatedly until a set of convergence requirements is satisfied. The procedure of finding extrema, interpolating envelopes, calculating the mean, and deducting it from the signal is repeated at each iteration in order to extract a new IMF.

4.4.6 Residual Component

The low-frequency or trend component of the initial signal is represented by the residual component, which is the component left over after all IMFs have been extracted.
EMD has a number of benefits, such as its applicability for time-frequency analysis, flexibility in responding to the local characteristics of the signal, and capacity to manage non-stationary and non-linear data.

4.5 Discrete wavelet transformation (DWT)

For the analysis and breakdown of signals into their component frequency parts, the Discrete Wavelet Transform (DWT) is a very effective signal processing method. It allows for both time and frequency localization by offering a multi-resolution analysis of signals. Here is the overview of DWT:

4.5.1 Decomposition Process

At various resolution levels or scales, the DWT breaks down a signal into approximation (low-frequency) and detail (high-frequency) components. A number of filtering and downsampling steps are used to accomplish this decomposition.

4.5.2 Filter Bank

The DWT employs two filters: a high-pass filter known as the wavelet or detail filter, and a low-pass filter known as the scaling or approximation filter. These filters pick up various signal frequency components.

4.5.3 Downsampling

The signal is downsampled by a factor of two to cut its length in half after filtering. Using this downsampling technique, the approximate and detail coefficients at various scales can be obtained.

4.5.4 Hierarchical Decomposition

The DWT generates a number of detail and approximation coefficients at various resolution levels or scales through a hierarchical operation. Every stage of decomposition gathers data about the signal at various frequency ranges.
Equation:

$$\text{DWT}(x) = \sum_k c_{j,k}\phi_{j,k}(t) + \sum_k d_{j,k}\psi_{j,k}(t) \tag{4.6}$$

Where:
- x is the input signal.

- $c_{j,k}$ are the approximation coefficients at level j and position k.

- $d_{j,k}$ are the detail coefficients at level j and position k.

- $\phi_{j,k}(t)$ and $\psi_{j,k}(t)$ are the scaling (approximation) and wavelet (detail) functions, respectively, at leve j and position k.

4.6 Discrete Fourier Transform (DFT

A popular mathematical technique for examining the frequency content of discrete-time signals or sequences is the Discrete Fourier Transform (DF T). It converts a series of real or complex values into a series of complex numbers that represent the frequency spectrum of the signal.

Here is the overview of DFT:

The DFT of a sequence $x[n]$ of length N is given by: **Discrete Fourier Transform (DFT)**:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn} \tag{4.7}$$

4.6.1 Frequency Components

The DFT generates a complex-valued spectrum $X[k]$ that represents the input signal's frequency components. Each frequency component's amplitude and phase shift are represented by the magnitude and phase of $X[k]$, respectively.

To reconstruct original signal use following formula:

Inverse Discrete Fourier Transform (IDFT):

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j\frac{2\pi}{N}kn} \tag{4.8}$$

4.7 Edge Device Algorithm

We took a signal, passed it to DWT and EMD then we find zero crossing of that processed signal and based on value of that zero crossing respective fault class. To make the range for each class all the signals of that class were passed divided into multiple windows of size of 12000 for our CWRU dataset each window was through above steps and their zero crossing values were noted. Then the box plots of those recorded zero crossings were create the range was created from 25th and 75th percentile of the box plot. Since there was a very high overlap between ranges of zero crossings between error classes i.e. Inner Race, Outer Race and Ball Bearings the zero crossings obtained from EMD were only used to classify whether the signal is faulty or normal. In order to classify the error class i.e Inner Race, Outer Race and Ball Bearings and their subclasses i.e error at depth of 0.007, 0.014, 0.021 inches zero crossings obtained from DWT were used and the rule-based tree like algorithm was created. Following Figure 4.3 shows the flow diagram of our algorithm:

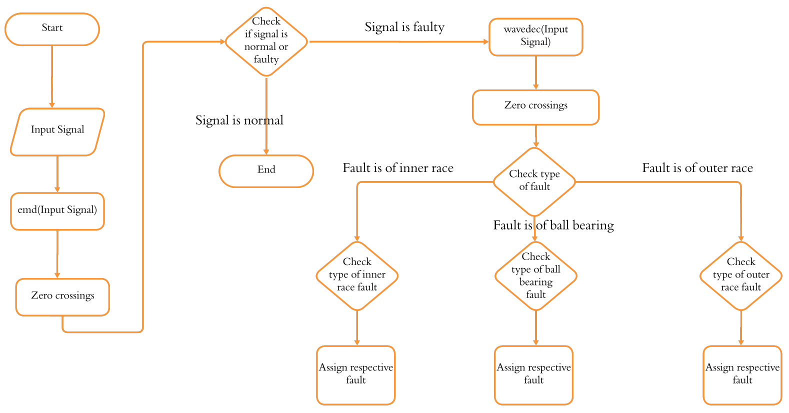


Figure 4.3: Complete flow graph of computation from taking input samples to classifications

4.8 Deep learning

To classify between error class i.e. Inner Race, Outer Race and Ball Bearings and sub-classes of each error class i.e. error at depth of 0.007, 0.014, 0.021 inches again we used the CWRU dataset for our model validation and for this purpose we used one dimensional convolutional neural networks (1DCNNs) and followings layers were includes in model architecture.

4.8.1 1D Convolutional Layer

For convolutional neural networks (CNNs) to process sequential data, including time series, audio signals, and other one-dimensional data arrays, 1D convolutional layers are an essential component. These layers create a feature map by sliding filters, or kernels, over the input data to pick up specific patterns and characteristics within the sequence. A 1D convolution's mathematical expression is as follows: **1D Convolution Equation**:

$$y[i] = (x * w)[i] = \sum_{j=0}^{k-1} x[i+j]w[j] \tag{4.9}$$

Where:

- x is the input signal.

- w is the filter (or kernel) of size k.

- $*$ denotes the convolution operation.

- i is the index of the output position.

- j is the index within the filter.

This equation describes how each output value $y[i]$ is computed by taking the dot product of the filter w with a segment of the input x, starting at position i.

4.8.2 Max Pooling Layer

In convolutional neural networks (CNNs), max pooling layers are crucial for downsampling the spatial dimensions of input feature maps. While maintaining the most crucial features, this procedure aids in lowering the network's computational load and parameter count. By swiping a window over the input feature map and calculating the maximum value inside the window at each step, max pooling accomplishes this. In addition to shrinking the feature map, this procedure adds a type of spatial invariance that strengthens the network's resistance to input translations. A Max Pooling layer's mathematical operation is represented as follows:

Max Pooling Equation:

$$y[i] = \max(x[j]) \quad \text{for} \quad j \in \text{window}(i) \tag{4.10}$$

Where:

- x is the input feature map.
- y is the output feature map.
- $\max$ denotes the maximum operation.
- window(i) represents the set of input values within the pooling window centered at position i.

This equation describes how each output value $y[i]$ is computed by taking the maximum value from the set of input values within the pooling window centered at position i.

4.8.3 Batch Normalization Layer

In contemporary deep learning architectures, batch normalization layers play a critical role in stabilizing and expediting neural network training. This layer helps reduce the internal covariate shift problem, which arises when the distribution of the inputs for each layer varies during training, by normalizing the inputs of each mini-batch to have a mean of zero and a variance of one. The Batch Normalization layer makes sure that the network performs better and trains more quickly by normalizing the input. The layer also performs a scaling and shifting transformation after normalization, which enables the network to discover the ideal mean and variance values. Mathematically speaking, batch normalization works as follows:

Batch Normalization Equations:

$$\hat{x}_i = \frac{x_i - \mu_{\text{batch}}}{\sqrt{\sigma^2_{\text{batch}} + \epsilon}} \tag{4.11}$$

$$y_i = \gamma \hat{x}_i + \beta \tag{4.12}$$

Where:

- x_i is the input feature.
- μ_{batch} is the mean of the mini-batch.
- σ^2_{batch} is the variance of the mini-batch.
- ϵ is a small constant added for numerical stability.
- $\hat{x}_i$ is the normalized input.
- γ and β are learnable parameters for scaling and shifting, respectively.
- y_i is the output feature.

4.8.4 Flatten Layer

In convolutional neural networks (CNNs), flatten layers are a basic but crucial part of the network that is used to prepare the data for the fully connected layers that come after the convolutional and pooling layers. Specifically, a flatten layer's job is to convert a multi-dimensional input tensor into a one-dimensional vector because fully connected layers require a one-dimensional input. For example, following the convolutional and pooling layers, which preserve the spatial structure of the data, the Flatten layer will transform the three-dimensional feature maps into a single long vector while maintaining the elements' order. This transformation enables the fully connected layers to process the data efficiently for tasks like classification or regression. Equation of flatten layer is as bellow:

Flatten Layer Equation:

$$y = \text{Flatten}(X) \tag{4.13}$$

Where:

- X is the input tensor of shape $(d_1, d_2, \ldots, d_n)$.
- y is the output vector of shape $(d_1 \times d_2 \times \ldots \times d_n)$.

4.8.5 Fully Connected or Dense Layer

Dense layers, sometimes referred to as fully connected (FC) layers, are an essential portion of neural networks, especially when convolutional neural networks (CNNs) are nearing their conclusion for tasks like regression and classification. A thorough integration of the extracted characteristics is made possible by the connections between every neuron in a Fully Connected layer and every neuron in the layer before it. The input data is first transformed linearly by this layer, and then a non-linear activation function is applied. A Fully Connected layer's mathematical operation is represented as follows:

Fully Connected Layer Equation:

$$y_i = \sigma\left(\sum_j w_{ij} x_j + b_i\right) \tag{4.14}$$

Where:

- x_j is the input to the FC layer.

- w_{ij} is the weight connecting input j to neuron i.

- b_i is the bias term for neuron i.

- σ is the activation function (e.g., ReLU, sigmoid).

- y_i is the output of neuron i.

4.8.6 Dropout Layer

Neural networks employ dropout layers as a regularization approach to avoid overfitting. At each update during training, Dropout arbitrarily sets a portion of the input units to zero, forcing the network to pick up redundant representations and strengthen itself. As a result, the network is less dependent on any one neuron, which encourages the emergence of independent characteristics and enhances generalization to fresh input. In actual use, dropout is applied to a layer's outputs either before or after activation, and throughout a single training iteration, the same units are dropped in each forward pass. It is possible to adjust the hyperparameter of the fraction of neurons to decline. The Dropout operation can be stated mathematically as follows:

Dropout Layer Equation:

$$y_i = \frac{d_i \cdot x_i}{1 - p} \tag{4.15}$$

Where:

- x_i is the input to the Dropout layer.

- d_i is a binary mask where each element is 0 with probability p and 1 with probability $1 - p$.

- p is the dropout rate, i.e., the fraction of the input units to drop.

- y_i is the output of the Dropout layer after scaling.

4.8.7 Architecture of our model

Our model takes a 1D signal of 2000 samples to predict its class and has the following architecture:

- Four 1D convolutional layers each having a kernel of size 5.
- A 1D max pooling layer after the convolutional layers. This layer has a kernel of size 2.
- A flatten layer after the max pooling layer.
- A batch normalization layer after the max pooling layer.
- Three dense layers alternating with dropout layers.
- Output layer.

The convolutional layers and dense layers use ‘relu’ as their activation functions. Our model has total 101146 parameters out of which 91226 are trainable parameters and 9920 are non-trainable parameters. Depiction of model along with its summary is explained in Figure 4.4 and 4.5.

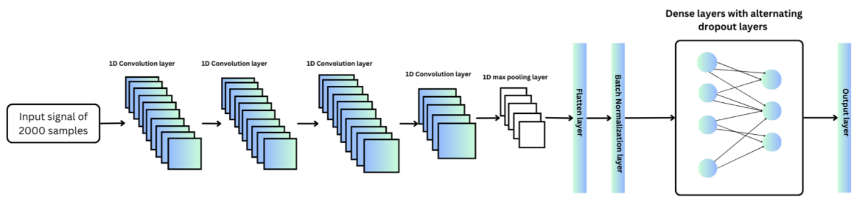
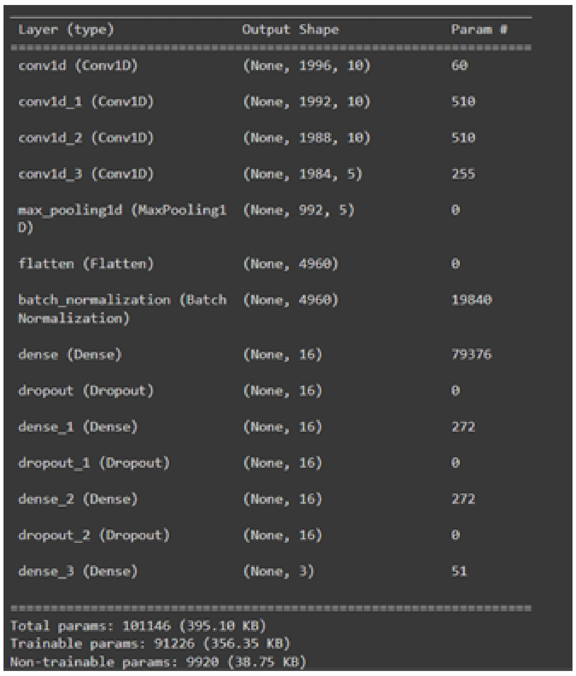


Figure 4.4: Model Summary- all the layers (CNN->Max Pooling->Normalization->Flatten->Dense->Dropout->Output



```
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 1996, 10)          60

 conv1d_1 (Conv1D)           (None, 1992, 10)          510

 conv1d_2 (Conv1D)           (None, 1988, 10)          510

 conv1d_3 (Conv1D)           (None, 1984, 5)           255

 max_pooling1d (MaxPooling1  (None, 992, 5)            0
 D)

 flatten (Flatten)           (None, 4960)              0

 batch_normalization (Batch  (None, 4960)              19840
 Normalization)

 dense (Dense)               (None, 16)                79376

 dropout (Dropout)           (None, 16)                0

 dense_1 (Dense)             (None, 16)                272

 dropout_1 (Dropout)         (None, 16)                0

 dense_2 (Dense)             (None, 16)                272

 dropout_2 (Dropout)         (None, 16)                0

 dense_3 (Dense)             (None, 3)                 51

=================================================================
Total params: 101146 (395.10 KB)
Trainable params: 91226 (356.35 KB)
Non-trainable params: 9920 (38.75 KB)
```

Figure 4.5: Summary of deep learning model's architecture

4.9 React Application

A React application was developed for visualisation of the results. Normal signals are classified on the edge-device and their status is uploaded to an API deployed on Microsoft Azure. Faulty signals are classified with our deep learning model. The status of faulty signals is uploaded to another API and the result of this classification is shown on the dashboard. This enabled us to display the status of both normal and faulty signals on the dashboard.

4.9.1 Data Visualization

The data is collected from an IoT hub hosted on the Azure Cloud platform. This allows for real-time fetching of data. The API endpoint is integrated into the application using Axios for HTTP requests. The fetched data is stored in the application's state using React's `useState` hook. The `useEffect` hook is utilized for data fetching within a functional component. The application displays real-time data by updating the state at regular intervals, maintaining an index to slice the most recent data points for visualization. This is achieved by using the `useState` hook to manage the current index and the `useEffect` hook to update this index at set intervals.

4.9.2 Fault Detection Dashboard

The main functionality of the application is to detect faults in the bearings based on the vibration data and provide visual feedback to the user. The application provides the status of the bearing using dynamic UI components that change color and display messages based on the fault status. The `react-chartjs-2` library is utilized to render line charts that visualize the vibration data, showing both real-time data and historical trends.

4.10 Conclusion

This chapter discussed how the vibrations coming from motor bearings are processed on esp32 , how the results and data is sent to cloud and model description and deployment on cloud. It also covers the description of dashboard on which the results about bearings status is displayed.

Chapter 5

Deployment & Validation

float graphicx

5.1 Overview

This chapter provides a thorough discussion of the data processing and deep learning models used to classify motor bearing errors. The chapter describes the data flow on the Azure cloud, as well as the overall computing process, which includes input sampling and categorization. It then digs into the deep learning approach, explaining the construction and purpose of each layer in the 1D convolutional neural network used for error classification. The CWRU dataset is used to train and validate models, which includes preprocessing processes, loss functions, optimizers, and training parameters. Finally, the chapter displays the trained models' findings, which demonstrate their accuracy in classifying various types of bearing problems.

5.2 Azure Cloud

The project is deployed on the Azure cloud, with multiple Azure services used to guarantee effective data processing, storage, and machine learning integration. The project's detailed flow is explained in Figure 5.1 and the components use in defined below:

1. IoT Hub:

   The IoT Hub functions as a central communications hub, collecting data from our edge node devices. These devices continually monitor motor bearings and transmit real-time vibration data to the IoT Hub. The IoT Hub enables dependable and secure connectivity between edge nodes and cloud infrastructure.

2. Stream Analytics Jobs:

   The Stream Analytics task functions as an intermediate processing layer. It accepts real-time data from the IoT Hub, analyzes it to extract relevant characteristics like zero-crossing values, and then stores the processed data in the SQL Database. Stream Analytics guarantees that data is effectively streamed and converted for storage and subsequent analysis.

3. SQL Database:

   The SQL Database stores the zero-crossing values retrieved from the vibration data. These parameters are critical for defect identification and analysis. The database provides an organized and scalable storage solution, allowing for quick data access and retrieval for further processing by other system components.

4. Functions App:

   Our data processing pipeline relies heavily on the Functions App, a serverless computing service. It collects zero-crossing values from the SQL database and feeds them into the machine learning (ML) model. The ML model analyzes these numbers to discover any flaws and returns the results to the SQL database. In addition, the Functions App transmits zero-crossing information straight to the dashboard for real-time monitoring and visualization.

5. Azure Blob Storage:

Azure Blob Storage is used for storing machine learning models. Blob Storage offers scalable and safe storage for huge datasets and model files. The Functions App uses ML models stored in Blob Storage to perform fault detection and classification activities.

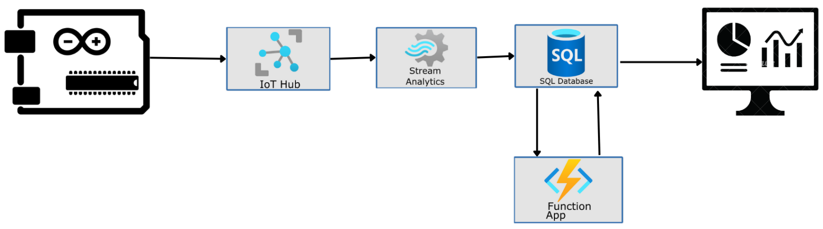


Figure 5.1: Data flow on Azure cloud

5.3 ML Validation

5.3.1 Signal is faulty or normal

Using the zero crossings obtained from EMD. We were easily able to classify between normal and faulty bearing with an accuracy of 100%. Following Figure 5.2 is the resulting confusion matrix:

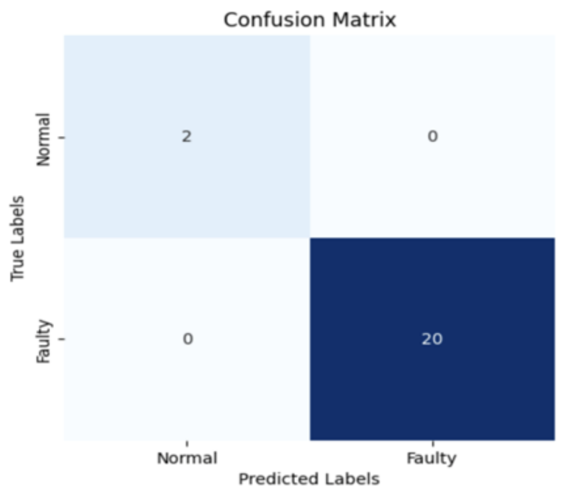


Figure 5.2: Confusion matrix for determining whether the signal is normal or faulty

5.3.2 Signal is faulty has error of Inner Race, Outer Race or Ball Bearing

Using the zero crossings obtained from DWT. We were easily not able to classify correctly between error class i.e. Inner Race, Outer Race and Ball Bearings. The signals of outer race class were classified as ball bearing error and the total accuracy of this was 65%. Following Figure 5.3 is the resulting confusion matrix:

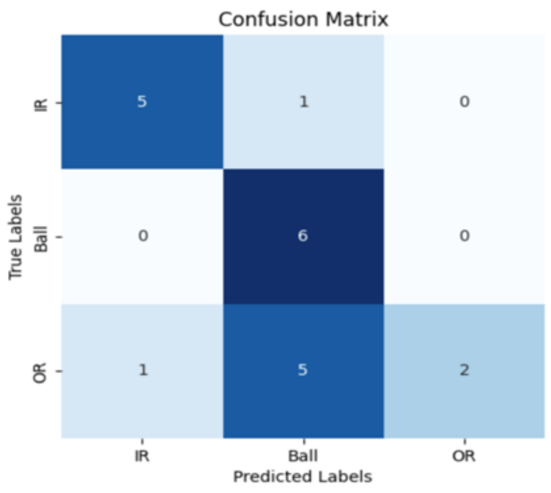


Figure 5.3: Confusion matrix for determining whether the signal is Inner Race, Ball bearing or outer race

5.3.3 Subclass of respective has error of signal

After the classifying the error class of signal, again using the zero crossings obtained from DWT. We classified subclasses of each fault i.e. error at depth of 0.007, 0.014, 0.021 inches. We were easily not able to classify correctly between subclasses. Again the signals of outer race with subclass of depth of 0.021 inches class were classified as ball bearing error with depth of 0.021 and the total accuracy of was 55%. The Table 5.1 shows the labels and assigned class to each label and the The resulting confusion matrix of our algorithm for subclasses of each fault type is shown in Figure 5.4 below::

Labels	Classes
1	Inner Race at 0.007
2	Inner Race at 0.014
3	Inner Race at 0.021
4	Ball Bearing at 0.007
5	Ball Bearing at 0.014
6	Ball Bearing at 0.021
7	Outer Race at 0.007
8	Outer Race at 0.014
9	Outer Race at 0.021

Table 5.1: Labels and Classes

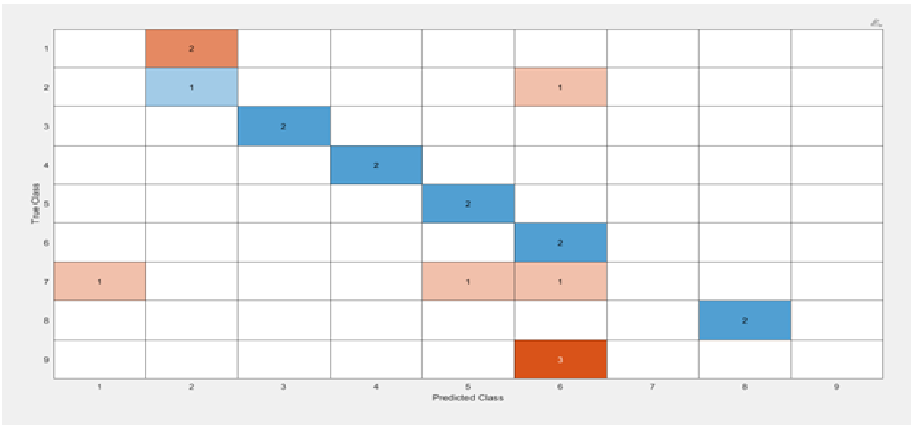


Figure 5.4: Confusion matrix for subclasses of each fault class

5.4 Conclusion

Based on the results and accuracies mentioned bellow in the Table 5.2 we decided to only classify whether the signal is faulty or normal on our edge device. But issue of memory arose in our edge device because EMD consumes a lot of memory so We applied low pass filter on multiple created windows of our signal. Then we normalized those windows ranging from -1 to 1. The zero crossing of those normalized windows were used to classify whether the signal is faulty or normal and again were able to achieve this correctly and for classifying the error classes and sub classes we decided to train deep learning models which will be deployed on cloud and the data will be uploaded to clouds results from each model will be obtained.

Categorical Distribution	**Accuracy**
Between normal and fault signals	100%
Between inner race (IR), ball, and outer race (OR) faults	65%
Between subcategories of inner race (IR), ball, and outer race (OR) faults	55%

Table 5.2: Categorical Distribution and Accuracy

5.5 DL Validation

5.5.1 Model training and validation

Just like the tree liked algorithm was created to classify between normal and faulty bearings and their types and subtypes of faults. A similar tree like structure was also created for our deep learning models. The only difference is that this our deep learning models will predict whether the signal is faulty or normal. They will only predict the type of fault i.e. Inner Race, Outer Race and Ball Bearing and subclasses of each fault i.e. error depth of 0.007, 0.014, 0.021 inches. So, the total of four models were trained and they are:

- Classification of error classes
- Classification of subtypes of Inner Race error
- Classification of subtypes of Ball Bearing error
- Classification of subtypes of Outer Race error

Collectively they create a tree like structure as shown in Figure 5.5:

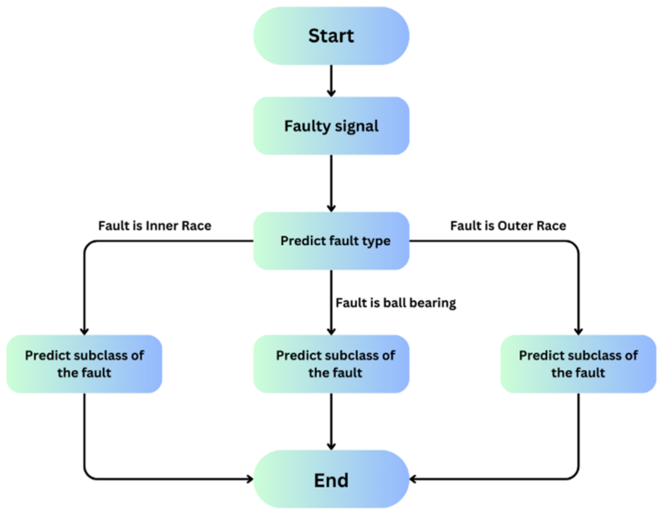


Figure 5.5: Flow Chart

5.5.2 Classification of error classes

To train this model we used CWRU bearing data and the whole faulty data was used. The faulty data was divided into three classes i.e. Inner Race, Ball Bearing and Outer Race and following labels mentioned in Table 5.3 were assigned to each class:

Labels	Classes
0	Inner Race
1	Ball Bearing
2	Outer Race

Table 5.3: Labels and Classes

Each signal in data was divided to multiple small windows each of 2000 samples with an overlap of 500 samples with previous window and their respective labels were assigned to those windows. In this way the preprocessing of data was done. The architecture that we defined earlier was used in this model.
The loss function we used was ‘sparce categorical crossentropy’. This loss function prevents the conversion of numeric labels into one hot encoded labels. Adam optimizer was used with a learning rate of 0.00001. Batch size was 32 and total epochs were 100.

5.5.3 Classification of subtypes of Inner Race error

To train this model we used CWRU bearing data and only inner race fault data was used. The data was divided into three classes i.e. Inner Race at 0.007, Inner Race at 0.014 and Inner Race at 0.021 inches and following labels mentioned in Table 5.4 were assigned to each class:

Labels	**Classes**
0	Inner Race at 0.007
1	Inner Race at 0.014
2	Inner Race at 0.021

Table 5.4: Labels and Classes

Each signal in data was divided to multiple small windows each of 2000 samples with an overlap of 500 samples with previous window and their respective labels were assigned to those windows. In this way the preprocessing of data was done. The architecture that we defined earlier was used in this model. The data was then divided into training and testing data and the training and testing split was of 20%.
The loss function we used was ‘sparce categorical crossentropy’. This loss function prevents the conversion of numeric labels into one hot encoded labels. Adam optimizer was used with a learning rate of 0.00001. Batch size was 32 and again total epochs were 100 for training.

5.5.4 Classification of subtypes of Ball bearing error

To train this model we used CWRU bearing data and only ball bearing fault data was used. The data was divided into three classes i.e. Ball Bearing at 0.007, Ball Bearing at 0.014 and Ball Bearing at 0.021 inches and following labels mentioned in Table 5.5 were assigned to each class:

Labels	**Classes**
0	Ball Bearing at 0.007
1	Ball Bearing at 0.014
2	Ball Bearing at 0.021

Table 5.5: Labels and Classes

Each signal in data was divided to multiple small windows each of 2000 samples with an overlap of 500 samples with previous window and their respective labels were assigned to those windows. In this way the preprocessing of data was done. The architecture that we defined earlier was used in this model. The data was then divided into training and testing data and the training and testing split was of 20

The loss function we used was 'sparce categorical crossentropy'. This loss function prevents the conversion of numeric labels into one hot encoded labels. Adam optimizer was used with a learning rate of 0.0001. Batch size was 32 and again total epochs were 100 for training.

5.5.5 Classification of subtypes of Outer Race error

To train this model we used CWRU bearing data and only outer race fault data was used. The data was divided into three classes i.e. Outer Race at 0.007, Outer Race at 0.014 and Outer Race at 0.021 inches and following labels mentioned in Table 5.6 were assigned to each class:

Each signal in data was divided to multiple small windows each of 2000 samples with an overlap of 500 samples with previous window and their respective labels were assigned to those windows. In this way the preprocessing of data was done. The architecture that we

Labels	Classes
0	Outer Race at 0.007
1	Outer Race at 0.014
2	Outer Race at 0.021

Table 5.6: Labels and Classes

defined earlier was used in this model. The data was then divided into training and testing data and the training and testing split was of 20
The loss function we used was 'sparce categorical crossentropy'. This loss function prevents the conversion of numeric labels into one hot encoded labels. Adam optimizer was used with a learning rate of 0.00001. Batch size was 32 and again total epochs were 100 for training.

5.5.6 Results of Models

Following models were trained:

- Model 1: Classification of error classes
- Model 2: Classification of subtypes of Inner Race error
- Model 3: Classification of subtypes of Ball Bearing error
- Model 4: Classification of subtypes of Outer Race error

Our model 1 has an accuracy of 94.81% and the resulting confusion matrix for our error classes of Inner Race, Outer Race and Ball Bearing is shown in Figure 5.6:

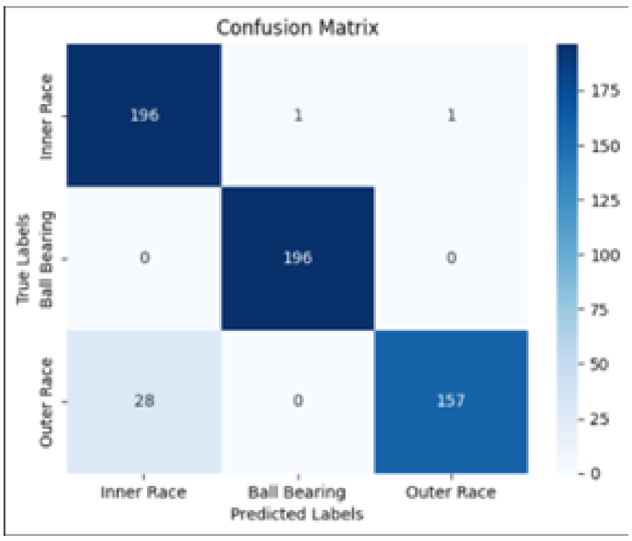


Figure 5.6: Classification of error classes

Our model 2 has an accuracy of 96.37% and the resulting confusion matrix for our types of inner race error i.e inner race at 0.007 inches depth, inner race at 0.014 inches depth, inner race at 0.021 inches depth is shown in Figure 5.7:

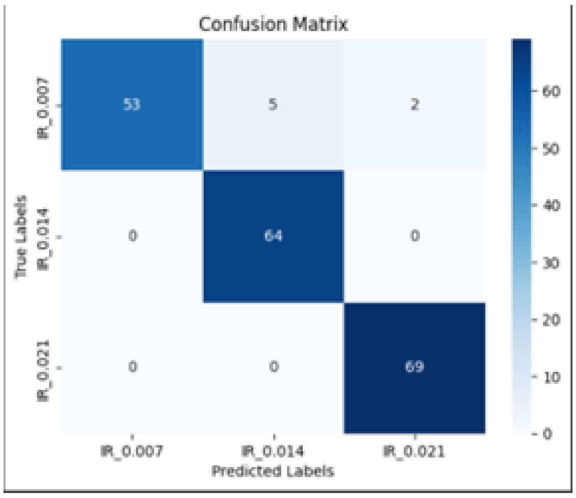


Figure 5.7: Classification of subtypes of Inner Race error

Our model 3 has an accuracy of 96.89% and the resulting confusion matrix for our types of ball bearing error i.e ball bearing at 0.007 inches depth, ball bearing at 0.014 inches depth, ball bearing at 0.021 inches depth is shown in Figure 5.8:

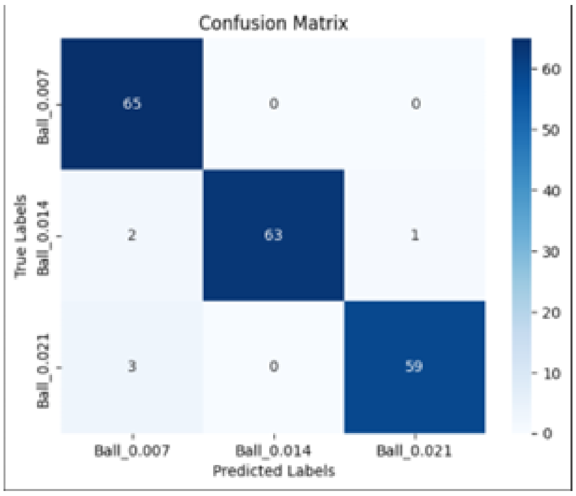


Figure 5.8: Classification of subtypes of Ball Bearing error

Our model 4 has an accuracy of 93.78% and the resulting confusion matrix for our types of outer race error i.e outer race at 0.007 inches depth, outer race at 0.014 inches depth, outer race at 0.021 inches depth is shown in Figure 5.10:

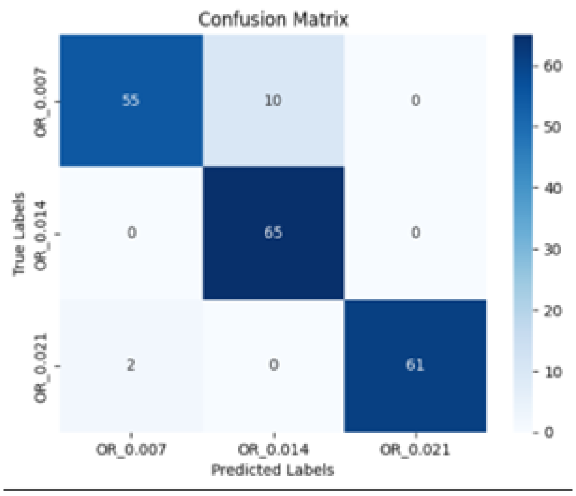


Figure 5.9: Classification of subtypes of Outer Race error

The accuracies of our trained models are shown in the Table 5.7.

Model Name	**Accuracy**
Model 1: Classification of error classes	94.81%
Model 2: Classification of subtypes of Inner Race error	96.37%
Model 3: Classification of subtypes of Ball Bearing error	96.89%
Model 4: Classification of subtypes of Outer Race error	93.78%

Table 5.7: Trained models and their accuracies

5.6 Fine tuning of Model 1 on our data

Since we our created dataset did not have excessive data for to classify between subtypes of inner race, outer race and ball bearing, we only fined tuned the model 1 which classifies between faulty types i.e. inner race, outer race and ball bearing on our own dataset.

For this purpose we kept the whole "Model 1" frozen except the weights of output layer. Then we trained this model on our dataset. The epoch were 100 which learning rate of 0.001 and batch size was 32. The loss function used was same as our validation models which is sparce categorical crossentropy and this fined tuned yielded us the accuracy of 94% on our test data. Following Figure 5.10 shows the resulting confusion matrix of our fined tuned model:

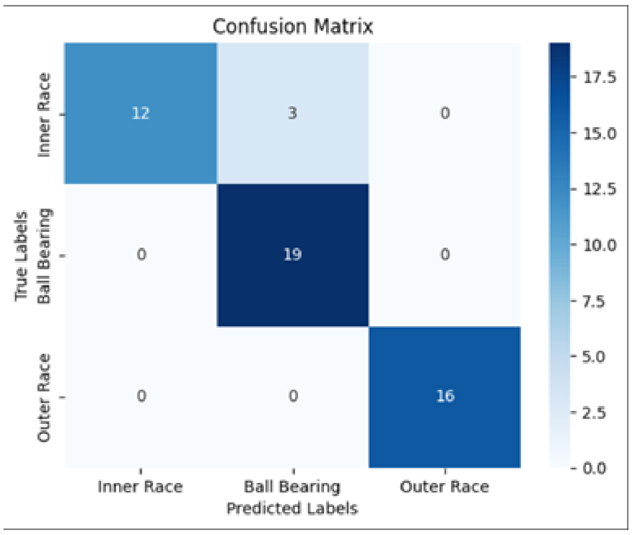


Figure 5.10: Confusion matrix of our fined tuned model

5.7 Front End Application

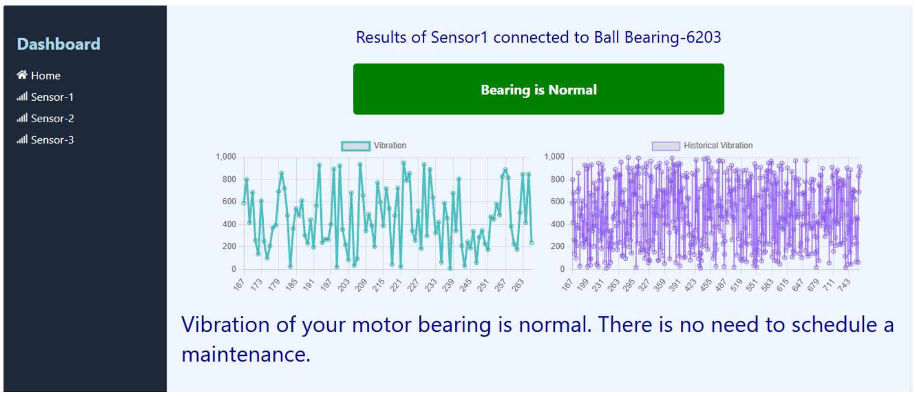
We developed the front end of our application using React JS. React router was used for client-side routing which enabled navigation between different pages of our application.

Charts JS was used because of the modern and clear UI elements it has for visualization. This allowed us to create an intuitive user experience for interaction with the application.

Every sensor is attached to a different sized bearing in our rig. Separate pages for every sensor is developed in our application for this purpose. Line graphs of zero crossings are plotted in our application which helps to decide whether to schedule a maintenance or not.

5.7.1 Normal Signal Results

The decision between normal and faulty signals was done both on the edge device and after passing it through deep learning models deployed on the cloud. The dashboard shows the results of both normal and faulty signals. After making an HTTP request using axios in React, the data is retrieved from the API in the form of a list. This is plotted on a line graph shown in the attached figure 5.11.



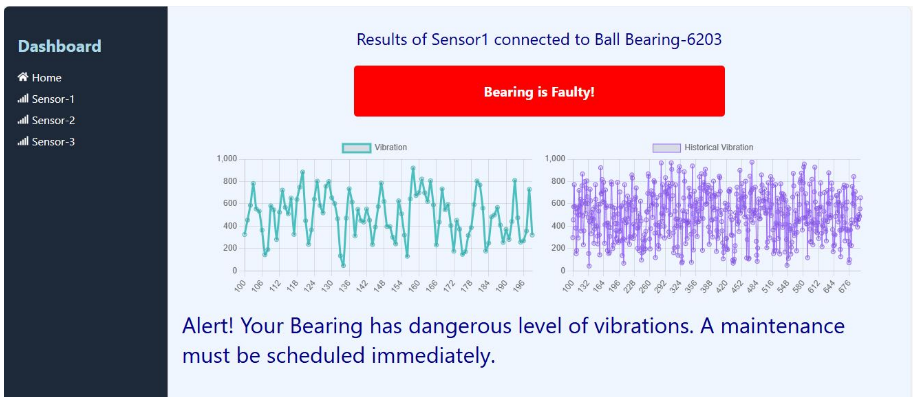
Results of Sensor1 connected to Ball Bearing-6203

Vibration of your motor bearing is normal. There is no need to schedule a maintenance.

Figure 5.11: Normal Signal

5.7.2 Faulty Signal Results

The faulty signals are passed through our deep learning model. The model deploys the results to another API we are using in our application This shows us the results of faulty bearings on our dashboard and gives us an alert that a maintenance has to be scheduled. The Historical Vibration plot gives us an insight to the history of different zero crossing values. The Figure 5.12 is the UI of our website for faulty bearing:



Results of Sensor1 connected to Ball Bearing-6203

Alert! Your Bearing has dangerous level of vibrations. A maintenance must be scheduled immediately.

Figure 5.12: Faulty Signal

5.8 Conclusion

This chapter shows the deployment of project on azure cloud, react application and the validation of deep learning models to classify motor bearing errors. It described the categorization procedure for several error types and subclasses, such as inner race faults, ball bearing defects, and outer race faults. The chapter describes the data preprocessing steps, model architecture, training procedures, and validation methodologies employed in the study and the processes used to deploy and display the results.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This project focused on developing an edge-based device for the early detection of motor bearing faults. This is a critical issue in industries leading to catastrophic failures and expensive repairs, both in terms of cost and time. We implemented a test rig to collect Real-time vibration data at the edge using an ESP32 microcontroller. Signal processing techniques were used to pre-process the vibration data, followed by the development of a deep learning model which is capable of distinguishing between healthy and faulty bearings. This model is deployed on Microsoft Azure, which enables real-time fault detection. A React application was developed to provide users with visualization of the results.

6.2 Future Prospects

The project has promising prospects for future development. A lot of implementations planned were not executed due to time constraints but with sufficient time and resources, this project has the potential to scale into a permanent solution for motor bearings fault issues. We hope that this project will be continued with an optimistic mindset and enthusiasm.

A key limitation of our approach is that the test rig operates on an extended shaft rather than directly on internal motor bearings. One improvement in future work would be to develop this solution for internal motor bearings. This would enhance the system's applicability in industrial IoT environments.

An approach which was not implemented due to time constraints involves developing models to estimate the remaining useful life (RUL) of motor bearings. The development of different Deep Learning models for RUL estimation would minimize unexpected downtimes and enable cost-effective maintenance scheduling.

One major improvement would be expanding the dataset by adding different type of signals such as acoustic signals and temperature data. The addition of these data streams would allow for a more robust fault detection system and a better monitoring system.

References

[1] L. Fu, K. Yan, Y. Zhang, R. Chen, Z. Ma, F. Xu, and T. Zhu, “Edgecog: A real-time bearing fault diagnosis system based on lightweight edge computing,” *IEEE Transactions on Instrumentation and Measurement*, vol. 72, pp. 1–11, 2023.

[2] R. Magar, L. Ghule, J. Li, Y. Zhao, and A. B. Farimani, “Faultnet: A deep convolutional neural network for bearing fault classification,” *IEEE Access*, vol. 9, pp. 25189–25199, 2021.

[3] R. Thomazella, W. N. Lopes, P. R. Aguiar, F. A. Alexandre, A. A. Fiocchi, and E. C. Bianchi, “Digital signal processing for self-vibration monitoring in grinding: A new approach based on the time-frequency analysis of vibration signals,” *Measurement*, vol. 145, pp. 71–83, 2019.

[4] Z. Germán-Salló and G. Strnad, “Signal processing methods in fault detection in manufacturing systems,” *Procedia Manufacturing*, vol. 22, pp. 613–620, 2018. 11th International Conference Interdisciplinarity in Engineering, INTER-ENG 2017, 5-6 October 2017, Tirgu Mures, Romania.

[5] Q. C. A. P. W. Sun, G. An Yang and K. Feng, “Fault diagnosis of rolling bearing based on wavelet transform and envelope spectrum correlation,” *Journal of Vibration and Control*, vol. 19, pp. 924–941, 8 2012.

[6] D. Neupane and J. Seok, “Bearing fault detection and diagnosis using case western

reserve university dataset with deep learning approaches: A review," *IEEE Access*, vol. 8, pp. 93155–93178, 2020.

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

[8] H. Pan, X. He, S. Tang, and F. Meng, "An improved bearing fault diagnosis method using one-dimensional cnn and lstm," *Strojniški vestnik - Journal of Mechanical Engineering*, vol. 64, no. 7-8, pp. 443–452, 2018.

[9] S. Li, G. Liu, X. Tang, J. Lu, and J. Hu, "An ensemble deep convolutional neural network model with improved d-s evidence fusion for bearing fault diagnosis," *Sensors*, vol. 17, no. 8, 2017.

[10] J. Hendriks, P. Dumond, and D. Knox, "Towards better benchmarking using the cwru bearing fault dataset," *Mechanical Systems and Signal Processing*, vol. 169, p. 108732, 2022.

[11] S. Anbalagan, B. Natarajan, and B. Srinivasan, "A physics based deep learning approach for cross domain bearing fault detection," in *2023 IEEE Kansas Power and Energy Conference (KPEC)*, pp. 1–4, 2023.

[12] D. Neupane and J. Seok, "Bearing fault detection and diagnosis using case western reserve university dataset with deep learning approaches: A review," *IEEE Access*, vol. 8, pp. 93155–93178, 2020.