# Advanced Pre-processing Techniques for Cloud-based Degradation Detection Using Artificial Intelligence (AI)

# Advanced Pre-processing Techniques for Cloud-based Degradation Detection Using Artificial Intelligence (AI)

By

ESSAM SEDDIK, MSc (AI), MSc (Mech), BSc

A Thesis Submitted to the School of Graduate Studies in Partial Fulfilment of the Requirements for the Degree of Doctor of Philosophy

McMaster University

# *Permission to Use*

In presenting this thesis in partial fulfillment of the requirements for a Postgraduate degree from McMaster University, I agree that the Libraries of this University may make it freely available for inspection. I further agree that the permission for copying this thesis in any manner, in whole or in part for scholarly purposes, may be granted by the professors who supervised my thesis work or, in their absence, by the Head of the department or the Faculty Dean in which my thesis work was conducted. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and McMaster University in any scholarly use which may be made of any material in my thesis. Requests for permission to copy or to make other use of material in this thesis, in whole or in part, should be addressed to:

Head of the Department of Mechanical Engineering

McMaster University Faculty of Engineering

1280 Main Street West

Hamilton, Ontario L8S 4L6, Canada.

DOCTOR OF PHILOSOPHY (2021),

McMaster University, Hamilton, Ontario(Mechanical Engineering)

| | |
|---|---|
| TITLE: | Advanced Pre-processing Techniques for Cloud-based Degradation Detection Using Artificial Intelligence (AI) |
| AUTHOR: | Essam Seddik, M. Sc, B.Sc. |
| SUPERVISORS: | Prof. Saeid Habibi and Prof. Alan Wassyng |
| NUMBER OF PAGES: | xxi, 211 |

# *Abstract*

Predictive maintenance is extremely important to fleet owners. On-duty automobile engine failures add cost of extra towing, gas and labor expenses which can add up to millions of dollars every year. Early knowledge of upcoming failures helps reduce these expenses. Thus, companies invest considerably in fault detection and diagnosis (FDD) systems to reduce unnecessary costs. Artificial Intelligence (AI) is getting increasingly used in the data driven signal based FDD industry because it requires less labor and equipment. It also results in higher productivity since it can operate continuously. This research offers Artificial Intelligence based solutions to detect and diagnose the degradation of three Internal Combustion Engine (ICE) parts which may cause on-duty failures: lead-acid accessory battery, spark plugs, and Exhaust Gas Recirculation (EGR) valve. Since the goal behind most FDD systems is cost reduction, it is important to reduce the cost of the FDD test. Therefore, all the FDD solutions proposed in this research are based on three types of built-in sensors: battery voltage sensor, knock sensors and speed sensor. Furthermore, the engine database, the Machine Learning (ML) and Deep Learning (DL) models, and the virtual operating machines were all stored and operated in the cloud.

In this research, eight Machine Learning (ML) and Deep Learning (DL) models are proposed to detect degradations in three vehicle parts mentioned above. Additionally, novel advanced pre-processing approaches were designed to enhance the performance of the models. All the developed models showed excellent detection accuracies while classifying engine data obtained under artificially and physically induced fault conditions. Since some variant data samples could not be detected due to experimental flaws, defective sensors and changes in temperature and humidity, novel pre-processing methods were proposed for Long Short-Term Memory Networks (LSTM-RNN) and Convolutional Neural Networks (CNN) which solved the data variability problem and outperformed the previous ML/DL models.

## *Acknowledgment*

I would like to start my acknowledgments by deeply thanking my dear supervisors, Dr. Saeid Habibi and Dr. Alan Wassyng for their enormous guidance and support which helped me achieve my research goals in this PhD work.

Prof. Habibi has helped me land a very interesting research topic and provided me with all the resources needed, especially experimental and financial resources. Also, I believe a good boss creates positive vibes in their workplace. That is why I would like to thank him for creating such a healthy environment in his lab by building a great research group of amazing people and highly qualified students.

Prof. Wassyng, my co-supervisor, has been very helpful within the Artificial Intelligence, data management and software development areas. He has been always there for me and provided me with many great ideas and suggestions along the way.

I would also like to thank Cam Fisher and all my colleagues in the Centre for Mechatronics and Hybrid Technologies (CMHT) for their continuous help and their integrative teamwork which made my PhD journey a joyful one.

I would like to express my very deep gratefulness to my family in Canada: Gamal, Mona, Farid, and Nabila who have been supporting me since the moment I landed in Canada. They have been a great family to me and treated me like their son. Unfortunately, I was not fast enough to make Uncle Farid happy with my degree before he passed away, but I really hope he rests in peace.

The most special gratitude of all goes to my family in Egypt: Hamdy, Naglaa and Maram, for their mental support throughout my academic career. They have always believed in me and encouraged me to achieve my goals. I am also grateful to them for giving me the freedom of moving to another country and being away for years.

# *Table of Contents*

## *List of Figures*

# *List of Tables*

# Chapter 1

## Introduction

# Chapter 1 – Introduction

## 1.1 Overview

Artificial Intelligence (AI) is getting increasingly used in the Fault Detection and Diagnosis (FDD) industry. With the availability of both Big Data and much-improved computation capability, the use of AI for fault detection can be more feasible nowadays. Artificial Intelligence tools turn Big Data into learnable knowledge which can help us find solutions to industrial problems. In this research, both Big Data and Artificial Intelligence are used in Fault Detection and Diagnosis (FDD) of lead-acid accessory batteries and Internal Combustion Engines.

Fleet owners across every industry suffer on-duty emergency calls because of failures of starting batteries or engine parts as shown in Figure 1.1. Towing broken vehicles to their original hub creates undesired expenses. Sending back-up vehicles may result in late deliveries and doubling gas and labor costs. Loss of reputation is also an additional bill. This problem can be a nightmare for large fleet owners such as shipping companies and product suppliers.

*Figure 1.1 On-duty vehicle fails on the road*

Early knowledge about upcoming vehicle failures would help fleet owners save hundreds of thousands of dollars every year. This can be achieved by using fault prognosis that can help detect progression of faults in battery signals and engine parts. The cost of the fault prognosis methods typically comes from labor and equipment. Since the main goal behind fault prognosis is cost reduction, the solution must not be expensive. In this research, eight AI-based classification models were developed to detect the degradation of three vehicle parts in an engine using existing sensors only to keep operating cost to minimum. The vehicle parts are:

- ➢ Lead-acid accessory battery.
- ➢ Spark plugs.
- ➢ Exhaust Gas Recirculation (EGR) valve.

In this application, AI fault classification models require large amounts of engine data for training. The training data must include both healthy and faulty conditions from all the 3 vehicle parts mentioned above. This kind of data was not available in the public domain and needed to be generated. Therefore, it was necessary to collect data and build a new clean and reliable engine database which contained hundreds of thousands of engine cycles. This engine database was then migrated to the cloud to allow its sharing with other researchers and institutions. All the AI programming was implemented in Python and MATLAB. The Python-based and MATLAB-based classification models were also migrated to cloud virtual machines (VM) for the same reason. Figure 1.2 shows the workflow of the cloud-based model.

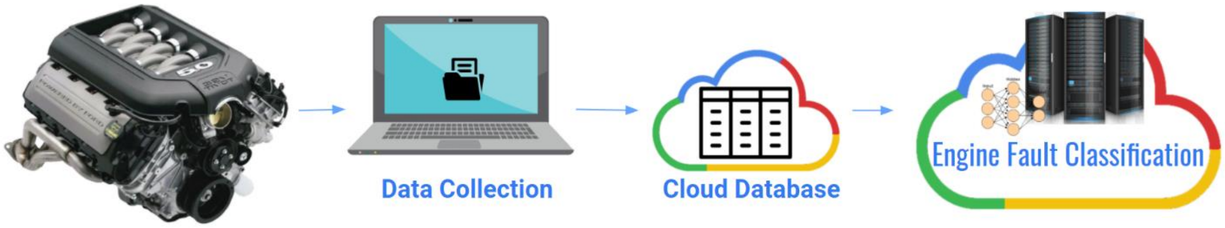


*Figure 1.2 Data pipeline from engine to cloud FDD*

### 1.1.1 Lead-acid Accessory Battery

The battery degradation detection element of this research involved a collaboration between McMaster's Center for Mechatronics and Hybrid Technology (CMHT) and Geotab Inc. The collaboration was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC). Geotab is a global leader in telematics which provides open platform fleet management solutions to businesses of all sizes. Many of Geotab's customers suffered from on-duty failures of 12v accessory batteries of their commercial vehicles and were looking for a solution to reduce the unnecessary costs of this problem. As a research institution, the CMHT's role was to develop a low-cost solution to detect progression of faults in the 12v accessory battery used in the powertrain to avoid on-duty failures. Since the main objective behind this project was cost reduction, only standard cranking voltage sensors were used in this research. All the battery data were collected by Geotab and cannot be shared in this thesis due to confidentiality limitations.

At the time of this project, Geotab collected up to 1.5 billion data points every day from 2.2 million active vehicles through their own communication devices referred to as "GO device". The GO device shown in Figure 1.3 is a small yet extremely powerful telematics measurement tool which collects engine and GPS data. Geotab is complemented by their own "MyGeotab" visualization and reporting software. Many of the largest fleets in North America and Europe are part of their roster of customers. The company has been logging voltage data for over 8 years, including battery voltage, and cranking voltage. The battery data indicate a noticeable number of battery breakdowns in some of their customer fleets while the technical reports show that many of the breakdowns happen on duty. The affected vehicles are towed to maintenance workshops and fail to complete their assignments.

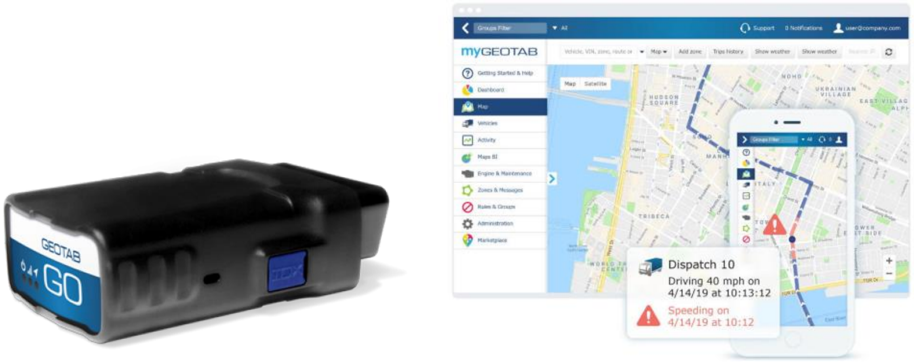


**Figure 1.3 Geotab's GO device**

The breakdowns result in extra costs and reduce productivity. Early detection of battery degradation would significantly reduce the expenses caused by unexpected battery failures. A fault prognosis tool would be a potential solution for maintaining a high and sustainable performance. In this thesis, the use of Machine Learning (ML) techniques will be considered as a prognostics tool to augment the functional architecture of telematics systems. Machine learning learns from historical data of the batteries that have already failed by finding patterns. The ability of the ML model to differentiate between healthy and failing batteries means that the model can detect new samples which are about to fail. This would be an early indication of upcoming battery failure, which would save large fleet owners millions of dollars in road call expenses every year. Battery voltage, cranking voltage, engine coolant temperature and other measurements would be used as input features to the machine learning model.

Due to Geotab's new data privacy policy, customers' data were not allowed to be shared anymore with third parties, like McMaster's CMHT. As a result, other applications were considered and the project was then expanded to detect degradation of other Internal Combustion engine (ICE) parts, namely Spark plugs and Exhaust Gas Recirculation (EGR) valve.

### *1.1.2 Spark Plugs*

One of the common causes of spark plug faults is the change in the gap size. This can happen when the spark plug is incorrectly installed or when the spark plug is degrading. If the spark plug is incorrectly installed, the gap may become too narrow over time and a shorter spark would be generated. This may result in early or weak ignition, which may create vibrations inside the engine. Another fault is when the spark plug degrades because of electrode wear. In this case, the gap becomes larger, and a wider spark would be generated which may result in late ignition or no ignition at all. In both cases, these faults may result in low engine performance, low efficiency, and high fuel consumption. Since the main objective of this research is cost reduction, only standard built-in knock sensors were used to capture engine vibration and knocking due to spark plug faults.

In this study, advanced AI-based fault classification models were built to detect faults in spark plugs (see Figure 1.4) using standard knock sensors only. ML and DL models require large amounts of training data to extract information from historical examples and make accurate classification decisions accordingly. Spark plug data were collected in the Center for Mechatronics and Hybrid Technology (CMHT). The center owns ICE engines, spark plugs, sensors, an engine dynamometer, and visualization software that are all available to use to collect the required training data.

**Figure 1.4 Ignition spark**

Keeping the cost of data collection as well as testing equipment low was very important to keep this model economically effective. Therefore, only standard built-in sensors in the ICE were used to collect data. The input data to the ML model must include information that reflect the health status of spark plugs. Knowing what sensors could indicate spark plug faults was a challenge. It is common knowledge that defective spark plugs affect the ignition system, which result in a drop in the engine performance. One of the popular symptoms of bad or failing spark plugs is knocking. Therefore, knock sensors were selected to be the primary sensors for data collection.

### *1.1.3 EGR Valve*

The exhaust gas recirculation (EGR) (see Figure 1.5) valve is an engine management component. It recirculates finely metered quantities of exhaust gas to the engine intake system to increase engine efficiency. This also reduces the fuel consumption rate and lowers NOx emissions. Since the world is increasingly caring more about reducing emissions, the EGR valve plays a very important role.

*Figure 1.5 EGR valve*

The atmosphere consists of nearly 80% nitrogen. However, when it is exposed to extremely high temperatures in the combustion chamber, the normally inert gas becomes reactive. The gas creates harmful oxides of nitrogen (i.e., NOx), which are then passed through the exhaust system into the atmosphere. Therefore, the EGR valve only allows a precise quantity of exhaust gas to re-enter the intake the system. This effectively changes the chemical mixture of the air entering the engine. With less oxygen, the diluted mixture burns slower which result in lowering the temperature inside the combustion chamber. This also helps reduce the production of NOx to keep the exhaust clean and more efficient. The EGR valve is normally closed while the engine is starting up. At idle and low speeds when only little power and oxygen is needed, the valve gradually opens. The valve can be up to 90% open at idle. As more torque, and therefore more power is required, the valve tends to close. During full acceleration, the EGR valve completely closes to ensure that the needed oxygen enters the cylinder. Thus, EGR valves can be used in engines to improve both combustion efficiency and knock tolerance as well as reduce pumping losses. In diesel engines, they can also help reduce knocking at idle. Older vehicles used to have vacuum-operated valves, while newer vehicles are now electronically controlled.

The hostile environment in which EGR valves operate results in wear and tear over time. However, the most common cause of EGR failure is the accumulation of carbon particles from the exhaust gases along the EGR valve and intake system passages. This results in clogged tubes and exhaust gas channels and possibly the plunger mechanism of the valve. This can force the valve to either remain open or closed. A rupture or leak in the valve diaphragm can also cause failure of the valve.

The symptoms associated with EGR valve failure are like those of many other engine management components, which makes it not easy for many technicians to diagnose. However, there are a few signs of EGR failure:

- ***Check engine light*:** An EGR valve fault may trigger the check engine light. Having the check engine light on makes the driver feel insecure since it can be on because of other major engine problems. The driver not knowing the reason why the check engine light is on might end up in stopping the vehicle on the road and asking for emergency help.
- ***Engine performance issues*:** if the valve remains open, the air-to-fuel ratio (AFR) will be disrupted causing engine performance issues such as power reduction, poor acceleration and rough idle. It may also produce turbo boost pressure leaks, which may cause difficulties to the turbo to work.
- ***Increased NOx emissions*:** On the other hand, if the EGR valve remains closed, the temperature will increase inside the combustion chamber which will leave a lot of unburned fuel in the exhaust. Therefore, NOx emissions will increase, and the fuel efficiency will decrease.
- ***Engine knocking*:** the higher temperatures and NOx emissions, the higher detonation or knocking in the engine. This symptom was the reason why engine knock sensors are used in this research to detect EGR faults. Not only that knock sensors are meant to sense knocking, but also, they already exist in all ICEs, which meets the objectives of this research. The level of knocking

in the engine is therefore used in this thesis as an indirect form of measurement that avoids use of additional instrumentation to diagnose the level of failure of the EGR valve.

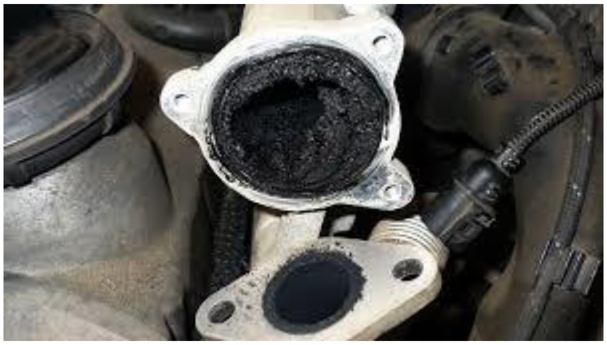


***Figure 1.6 Clogged EGR valve***

## *1.2 Research Motivation*

Early recognition of upcoming engine and vehicle battery failures can help save hundreds of thousands of dollars for large fleet owners. This research aims for early Fault Detection and Diagnosis (FDD) of 12v accessory batteries, spark plugs and Exhaust Gas Recirculation (EGR) valves through AI-based classification models to avoid on-duty failures, reduce maintenance costs and avoid major damages to the engine. Furthermore, no external sensors or transducers are required to help keep the cost of the solution low. Only built-in engine sensors are used for data collection and fault detection.

Due to experimental issues and equipment flaws, some of the collected data often have variant characteristics, such as different frequency or voltage amplitudes compared to the normal range. These variants are harder to detect by AI-based algorithms. Thus, this research provides novel pre-processing techniques which facilitated handling and, in some cases, use of certain forms of variant data.

## *1.3 Research Objectives*

The main research objective is to use Artificial Intelligence (AI) to perform Internal Combustion Engine (ICE) fault classification using standard engine sensors only. This was achieved by:

- building a labeled engine database of both healthy and faulty conditions, provided by McMaster's test engineers.
- Developing reliable Machine Learning (ML) and Deep Learning (DL) classification models which learns from engine data to detect and diagnose faults.
- Developing novel pre-processing techniques for Deep Learning algorithms to enhance the classification performance of variant engine data due to experimental issues and sensor flaws.

## *1.4 Datasets*

Three major datasets were collected in this research:

1. Battery cranking voltage dataset, which contains the cranking voltage history of vehicles with healthy batteries as well as vehicles with previous battery breakdowns. The dataset was collected through Geotab's GO devices which were connected to their customer's vehicle OBD II ports, as explained in section 1.1.1. This dataset was owned and stored by Geotab Inc. and cannot be shared in this thesis due to data confidentiality.
2. Spark Plug dataset, which contains physically simulated Original Equipment Manufacturer (OEM) spark plugs with three different gap sizes. The spark plugs were used to physically simulate fault while engine data were being collected through standard knock and speed sensors. The smallest gap size (i.e., 0.020") represents incorrectly installed spark plugs (a.k.a. fault 1), while the largest gap size (i.e., 0.080") represents degrading spark plugs (a.k.a. fault 2) due to

electrode wear. The spark plug with the medium gap size (i.e., 0.050") represents a standard and healthy spark plug. The entire spark plug dataset was collected at McMaster's CMHT lab.

Due to experimental issues and sensor flaws, about 40% of the data samples were considered as be corrupted and variant from the normal range with notably different characteristics (e.g., different voltage amplitudes and frequencies). Figure 1.7 shows an example of a regular knock sensor signal against a variant data sample due to a loose sensor cord. Note that in this signal, some of frequency content information may have been preserved, while the amplitude of the signal is clearly compromised. Therefore, one of the major novelties in this research was to invent a new method which enables classification by use of variant data that have partial information content.

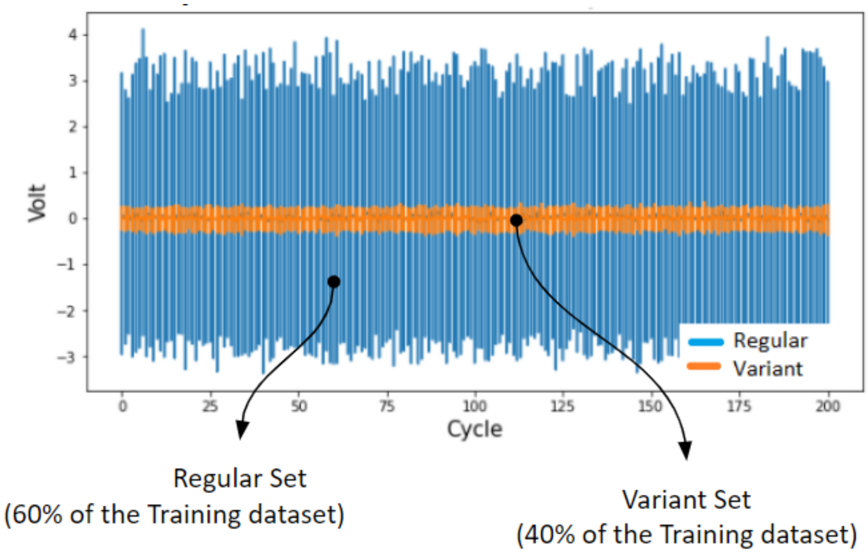


*Figure 1.7 Regular vs variant data*

3. Exhaust Gas Recirculation (EGR) valve dataset, which contains three physically simulated EGR valve faults. The faults contain 0% CO2 dilution (a.k.a. healthy), 5% CO2 dilution (a.k.a. fault

1), and 10% CO2 dilution (a.k.a. fault 2). The entire EGR valve dataset was collected at McMaster's CMHT lab.

## *1.5 Proposed Solutions*

The proposed solution meets the research objectives through eight AI-based fault classification models, as shown in Figure 1.8, which cover three common faults: degradation of 12v accessory batteries (i.e., Models B1 and B2), spark plugs (i.e., Models S1 to S4) and EGR valves (i.e., Models E1 and E2). All the proposed models were trained on their corresponding datasets discussed in section 1.4. The following naming convention of the models is used throughout the thesis:

- Model B1 was developed to detect 12v battery breakdowns in unlabeled cranking voltage signals using Artificial Neural Networks (ANN). The purpose of this model was to label battery data into healthy and breakdown batteries, which was used as training data in Model B2.
- Model B2 detects degrading 12v battery signals using cranking voltage data, which was labeled by Model B1, using the Random Forest (RF) classifier. The model suggests a new set of features which describe the signal curve instead of using raw data to enhance the detection rate of the RF classifier.
- Model S1 detects and diagnoses spark plug faults through standard knock sensors only. The model consists of McMaster CMHT's own Extended Multi-Scale Principal Component Analysis (EMSPCA), which generates fault signatures as input data to an Artificial Neural Network (ANN).

- Model S2 detects and diagnoses spark plug faults through standard knock sensors only. The model turns the raw data into sequential features as input data to a deep Long Short-Term Memory Recurrent Neural Network (LSTM-RNN). Model S2 and S3 were developed for 2 reasons: a) because the running time of Model S1 was high at the time of this research (not anymore), and b) to achieve higher accuracy.
- Model S3 detects and diagnoses spark plug faults through standard knock sensors only. The model turns the raw data into Mel-frequency Cepstral Coefficient (MFCC) heatmaps as input data to a deep Convolutional Neural Network (CNN).
- Model S4 uses a single engine speed sensor only and the Random Forest (RF) classifier to detect and diagnose spark plug faults.
- Model E1 detects and diagnoses Exhaust Gas Recirculation (EGR) valve faults through standard knock sensors only. The model consists of McMaster CMHT’s own Extended Multi-Scale Principal Component Analysis (EMSPCA), which generates fault signatures as input data to an Artificial Neural Network (ANN).
- Model E2 detects and diagnoses Exhaust Gas Recirculation (EGR) valve faults through standard knock sensors only. The model consists of Fast Fourier Transform (FFT) along with Principal Component Analysis (PCA). This model was developed to validate the results of Model E2.

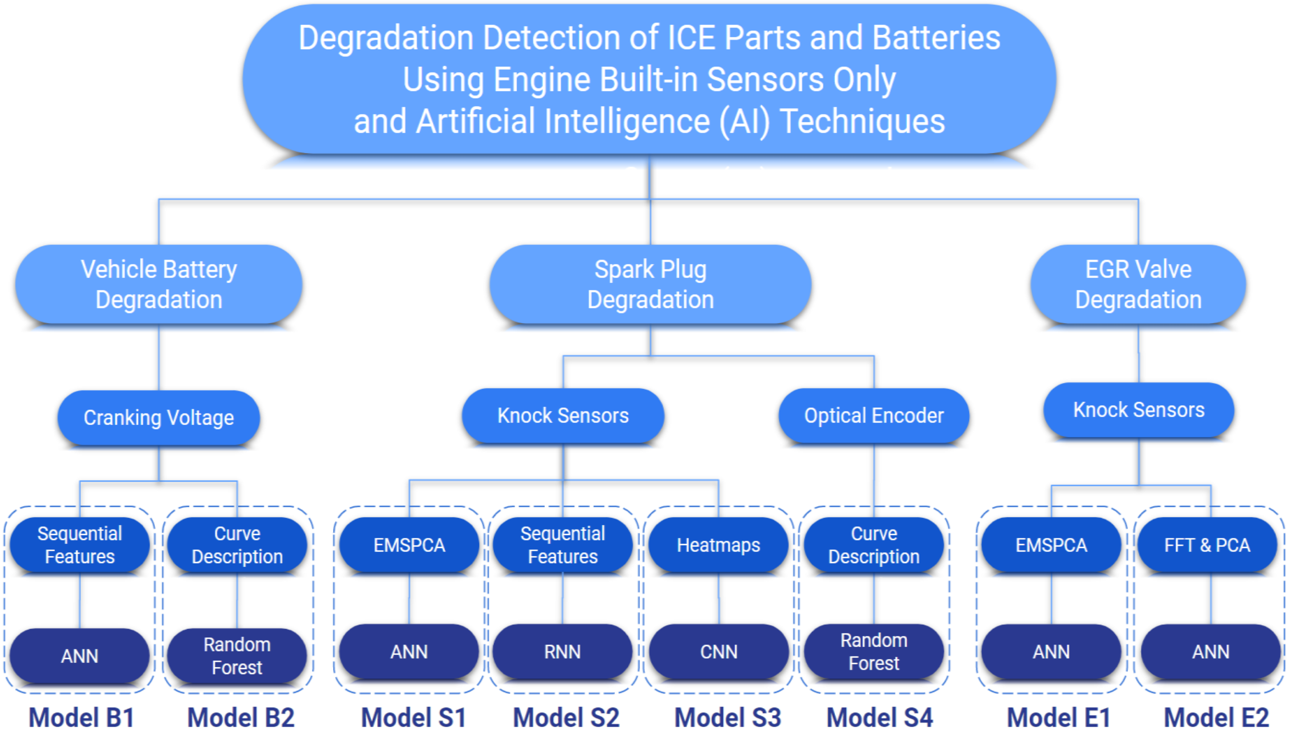


**Figure 1.8 Hierarchy of AI-based engine fault classification models**

## *1.6 Research Contributions and Novelties*

Novel pre-processing techniques were proposed in this research to enhance the classification accuracy of engine data using well-established Machine Learning (ML) and Deep Learning (DL) algorithms:

- **[CD-RF]:** A curve description (CD) pre-processing technique was proposed for Random Forest (RF) classifiers to perform time-series classification of engine data at almost 90% accuracy.
- **[MFCC-CNN]:** An image pre-processing strategy to time-series data was proposed for Convolutional Neural Networks (CNN) to achieve more than 97% detection rate.
- **[EMSPCA-Mosaic-CNN]:** A novel image pre-processing technique was invented for Convolutional Neural Networks (CNN) to perform time-series classification of engine data through EMSPCA fault signatures. This new technique enabled the conversion of spark plug

fault signatures to mosaic-like images which can be classified by the CNN with 96% accuracy, with respect to variance in data due to experimental issues and sensor flaws.

## *1.7 Implementations*

In this research, the following implementations were achieved:

- An easy-to-share, structured, and labeled spark plug and EGR condition database was generated by McMaster's CMHT test engineers for this research. It contains billions of engine sensory data points of both healthy conditions as well as physically simulated Spark Plug and Exhaust Gas Recirculation (EGR) fault conditions. The database was then migrated to the cloud to provide useable data to automotive researchers.
- Eight AI-based fault classification models were proposed to detect and diagnose faults in 3 vehicle parts:12v battery (models B1 and B2), spark plugs (models S1 to S4), and EGR valve (models E1 and E2) using standard engine sensors only. The models were able to achieve 95%, 96% and 100% detection rates of 12v batteries, spark plugs and EGR valve faults, respectively, during the tests.
- Extension and refinement of McMaster CMHT's own FDD algorithm, referred to as the Extended Multi-Scale Principal Component Analysis (EMSPCA), to perform classification of Internal Combustion Engines (ICE). The processing time of the EMSPCA algorithm was reduced by 99% and full documentation for the algorithm was created.

## *1.8 Thesis Structure*

This section describes the structure of the thesis, which is organized as follows:

Chapter 1 provides an overview about the problem solved in this research as well as the proposed solutions. The research projects conducted in this thesis were described. Brief descriptions of the research objectives, motivation, contributions, and novelties are also provided.

In chapter 2, a literature review on the current FDD methodologies used in the industry as well as research publications about the most recent FDD approaches, specifically signal-based methods. The Machine Learning (ML) and Deep Learning (DL) Techniques used in this research were also discussed along with the research limitations and problems to be solved.

Chapter 3 exhibits the battery data collection process, which was performed by McMaster's industry partner, Geotab Inc. The data cleansing and filtration process was also explained in this chapter along with many challenging data issues which were solved throughout the process.

Chapter 4 discusses the results of the battery degradation detection models developed in this research. Two Machine Learning (ML) models were built to detect battery replacements and progression of battery faults.

Chapter 5 discusses the experimental setup, and the equipment used in this research. The engine data collection process is discussed, which included 3 spark plug and 3 EGR valve fault conditions. Data analysis strategies for the collected database were also explained. A cloud-based structured database was generated from this research and discussed in this chapter.

Chapter 6 is where all the AI models developed for spark plug and EGR degradation detection were discussed in detail and fed with the collected engine data. The chapter also mentions the extension and refinement stages which were performed to the Extended Multi-Scale Principal Component Analysis

(EMSCPA) algorithm to handle ICE data classification since it was originally designed for electric motors. A

Chapter 7 proposes novel pre-processing techniques to the deep Convolutional Neural Network (CNN) to handle classification of EMSPCA fault signatures. These techniques helped solve the data variability problem due to experimental flaws and temperature change (see section 1.4). This model outperformed all the ML and DL models developed in this research which attempted to classify variant data.

Chapter 8 discusses the results of the engine data classification models and compares their performances in each problem. It contains the results of both spark plug degradation problem, and EGR valve degradation problem. A summary of the results of the classification models developed for both problems was also provided.

Chapter 9 provides a conclusion of the entire research and recommendations for future work.

# Chapter 2

# Literature Review

# *Chapter 2 - Literature Review*

Fault diagnosis and detection of ICE can help avoid major damages in the engine [1], [2]. There are two main categories of fault detection methods: signal-based and model-based [3], [4]. Many of the current FDD systems in industry [5] are signal-based and involve either conducting rule-based checks of raw measurements or using traditional statistical methods [6]. It is a common practice for FDD systems in rotating systems like ICE to conduct frequency analysis techniques, such as Fast Fourier transform (FFT), over engine speed signals [6]. However, FFT may not be the best approach with non-stationary vibration signals since frequency components change in time, which was shown by Rai and Mohanty [7] in their experimental research on faulty bearings. On the other hand, frequency components do not change in time with stationary vibration signals, which makes FFT a good tool for the latter case.

## *2.1 Fault Detection and Diagnosis of ICE Using Artificial Intelligence.*

With the increasing acceptance of Artificial Intelligence (AI) in the automotive industry, AI based FDD methods are being considered more often, especially when using multiple sensors [8]. One of the most popular AI algorithms is the Artificial Neural Networks (ANN) [9]. ANNs are data-driven classifiers that are currently categorized as signal-based Fault Detection and Diagnosis (FDD) methods and are getting increasingly used in many FDD applications [10]. ANNs can be used in Machine Learning such as for classification and regression.

More advanced versions of Artificial Neural Networks (ANNs), referred to as Deep Neural Networks (DNNs), are currently being used in the most advanced and complex technologies in the world such as self-driving cars [11], [12]. These networks require large amounts of data, which can be available through collecting many samples from each fault condition, multiple fault conditions, and multiple features, which are likely collected through sensors [13]. The main objective of this research is to utilize both Machine Learning (ML) and Deep Learning (DL) techniques in the detection of engine parts degradation, as well as vehicle batteries.

DL algorithms are not only deeper networks than ML algorithms, but also have more complex structure. ML algorithms can be based on neural networks like Artificial Neural Networks (ANN) or other concepts like Random Forest (RF) and Support Vector Machines (SVM). However, DL algorithms are only based on neural networks. Unlike ANNs, some Deep Neural Networks (DNN) were particularly designed to learn from images, such as Convolutional Neural Networks (CNN), while others were designed for time-series data like Recurrent Neural Networks (RNN). The ANN consists of simple hidden layers where the weights are calculated regardless of the type of data. The CNN contains convolutional and max pooling layers which splits images into patches.

Recent surveys [14] show that machine learning and deep learning techniques are being increasingly used in predictive maintenance applications. Figure 2.1 shows a trend line that describes the growth in the number of articles on predictive maintenance that were published between 2009 and 2018. Although the trend line is rising significantly, the predictive maintenance of spark plugs has never been studied using Deep Learning techniques. This paper describes our successful approach to diagnose and detect engine faults related to spark plugs using two of the most powerful deep learning algorithms with existing engine sensors.

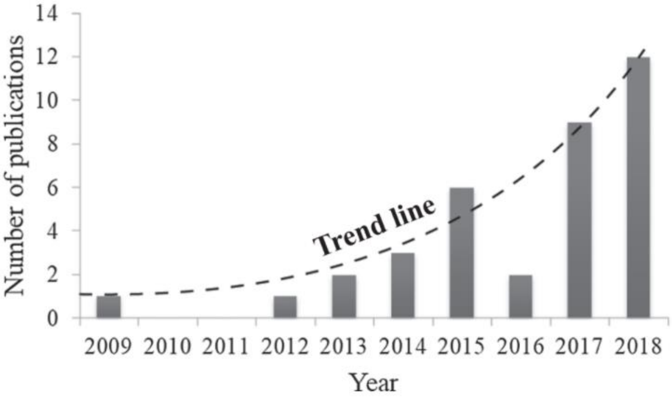


*Figure 2.1 Number of papers per year (with a trend line) [14]*

In a successful attempt of detecting gear faults, Samantha [14] reached a classification rate of nearly 100% using ANNs as well as Support Vector Machines (SVMs). Samantha used sensory data acquired from seven accelerometers to train both models. Laftah, Rafil et al. [15] developed an ANN-based model that detected different engine faults by analyzing exhaust gas data. The faults introduced to the engine included spark plug faults, valves, air filter, piston rings and carburetor. In this research, a different set of sensors will be used to detect Exhaust Gas Recirculation (EGR) valve [16] failure using Machine Learning (ML) models. In 2019, Carvalho et al. [17] published a systematic literature review of machine learning (ML) methods applied to predictive maintenance (Pd.M.). The survey included FDD models of different applications such as fuel cells [18], automobile gearbox [19], wind and gas turbines [20], air compressors [21], fans [22] and pumps [23]. Random Forest [24], SVM [25], ANN [26] and K-means [27] were all used in these FDD models as the acting classifiers.

## 2.2 Artificial Neural Networks

In cognitive science, we humans have neural networks [28] which are the central nervous systems of our brains. In machine learning, Artificial Neural Networks (ANNs) are models inspired by our neural networks used to estimate functions given historical data [29]. ANNs consist of interconnected layers with each layer containing neurons with numeric tunable weights which make the network capable of learning. Since traditional rule-based programming was not able to solve a wide variety of tasks, neural networks were invented to tackle more complex problems such as object detection and computer vision. The smallest unit of a typical neural network is referred to as the perceptron [30].

There are three major types of learning: supervised, unsupervised and reinforcement learning. Some other types can fall in between such as semi-supervised learning. Supervised learning is occurred using labeled data. The term label means information about the target being predicted. If information about the predicted target is available along with the dataset, then this is a supervised learning problem. In this case, the ML algorithm creates rules between the data itself and its labels (i.e., targets) based on the examples provided. Unsupervised learning works differently since no information about the target is available. In this case, the ML algorithm blindly assigns data points to groups based on measurements. Clustering methods such as K-means and Kernel Nearest Neighbor (KNN) are among the most popular unsupervised learning algorithms where data points are classified based on distances. In reinforcement learning, no data is given. Instead, environment is given and an agent who is supposed to navigate in this environment and finds its goal using its own experience that is gained gradually.

### *2.2.1 The Perceptron*

The perceptron is the basic neural network building block, which was the earliest supervised learning algorithm [31]. The perceptron takes historical data samples of a certain label, as well as samples of a different label (e.g., “car” and “not car”). The goal was to assign a test sample to the proper label. One way to do that was to assign the test sample to the closest neighbor as shown in Figure 2.2. The algorithm draws a separating line, referred to as the separator, and splits the data into two classes. Any point on one side of the line would be a car while any data point on the other side would not be a car. This is how a linear classifier works [32].

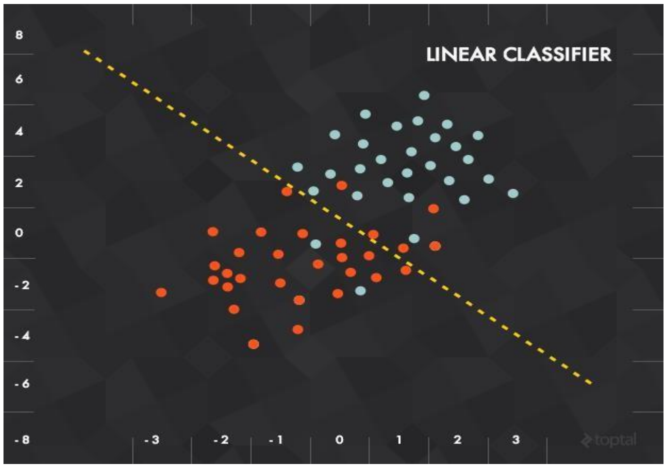


*Figure 2.2 Linear classifier [32]*

To represent this line mathematically, a weighted sum transfer function f(x) is formed [33]. The transfer function is equal to the sum of the input vector (x), multiplied by its weight (w), and a bias (b):

$$f(x) = x.w + b \tag{2.1}$$

The transfer function represents the threshold which is then fed into an activation function h(x), like in Figure 2.3. The activation function works as a threshold cut-off [34]. If the result exceeds a certain value, it labels it as “1” (i.e., car), otherwise, “0” (i.e., not car).

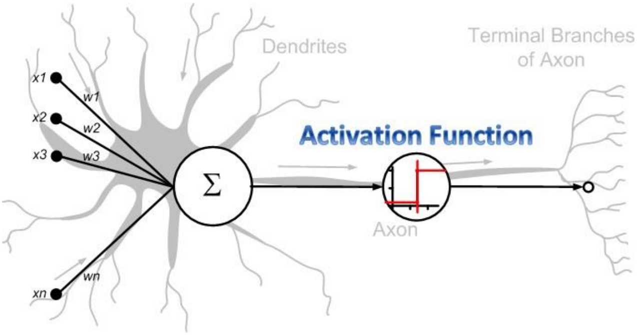


*Figure 2.3 Activation function [34]*

$$h(\mathbf{x}) = \begin{cases} 1 : & \text{if } f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b > 0 \\ 0 : & \text{otherwise} \end{cases} \qquad (2.2)$$

Weights (w) eliminate the output error, which is the difference between the desired and the actual output. Mean Square Error and Least Square Error are examples of the error functions which can be used in a transfer function. Single perceptron is simple and easy to apply, however, it can only learn linearly separable functions [35]. This is a major drawback because some simple functions, like XOR, cannot be classified as shown in Figure 2.4. That is why the multi-layered perceptron is used more often.

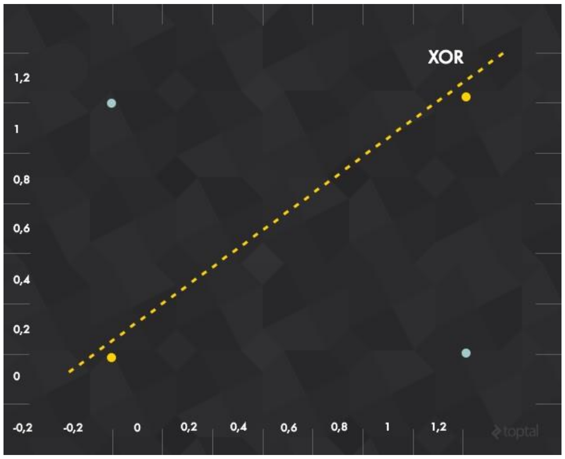


*Figure 2.4 XOR problem [32]*

### 2.2.2 Structure of Neural Networks

A neural network consists of layers, each layer consists of perceptron (i.e., “neuron” or “unit”). The first layer is always called the input layer and it is always a single layer [37]. The last layer is called the output layer, which is always a single layer as well, and this is where the target is generated. Between the input and the output layer, there are hidden layers, which can be a single layer or as many as needed. Figure 2.5 shows an example of a neural network with a 3-unit input layer, a 4-unit hidden layer and a 2-unit output layer. Each unit in a layer (n) is typically connected to every unit of the previous layer (n-1). These connections are where the weights are calculated and can be disconnected by setting their weights to zero. F(x) and h(x) represents the transfer function and the activation function, respectively.

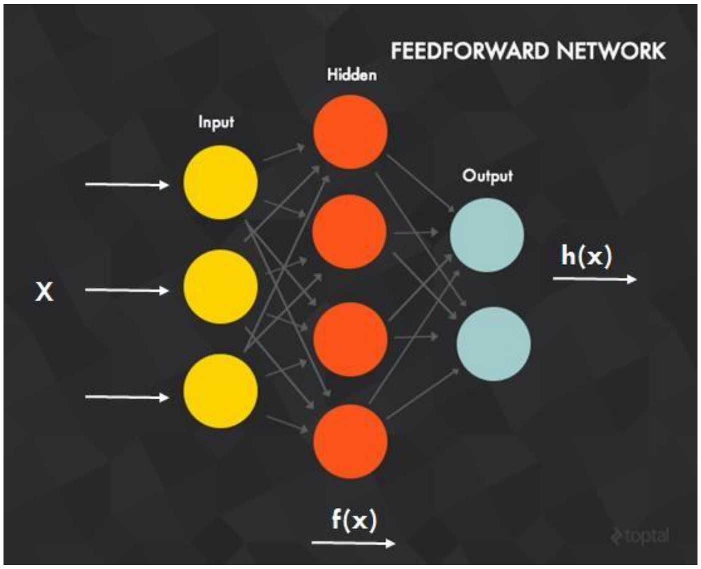


*Figure 2.5 Feedforward neural network [32]*

The network starts functioning when an input vector is fed to the input layer. The input vector is propagated forward to the hidden layers where the output of each layer becomes the input to the next.

The output of the last layer (i.e., output layer) becomes the output of the entire network [34]. Using linear activation functions makes the feedforward neural network not much more powerful than the perceptron, even if it contains multiple layers. That is the reason why non-linear activation functions are used in most neural networks. Networks with a single hidden layer are not powerful enough to learn functions. Thus, it is common practice to build neural networks with multiple hidden layers [38]. Figure 2.6 shows the different activation functions that can be used in neural networks.

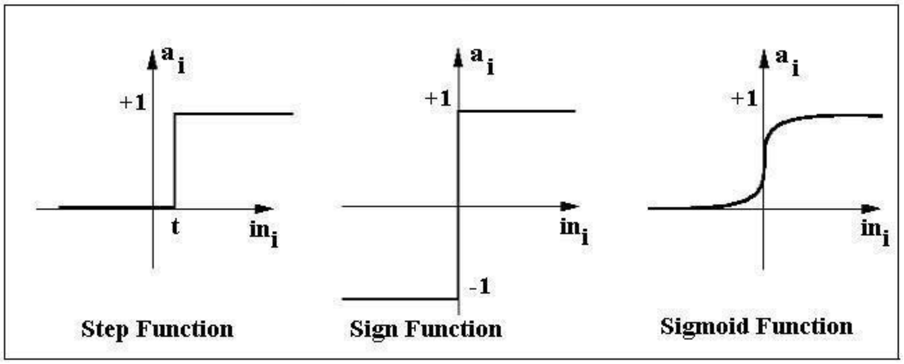


***Figure 2.6 Activation functions [31]***

Like humans, feed forward neural networks do not get the best results from the first time. More attempts do not mean only repetition, but also corrections to the weights throughout the training process. This can be done using backpropagation.

### *2.2.3 Backpropagation*

Backpropagation is the most common algorithm for supervised learning in Machine Learning (ML). It is a simple, yet very effective algorithm.

After publishing the seminal book “Parallel Distributed Processing” by Rumelhart and McClelland in 1986 [41], backpropagation started to get popular. Every backpropagation algorithm consists of two phases: forward phase and backward phase [42]. In the forward phase, the weights do not change, and the input data is propagated through the network all the way to the output layer. This is when the activation functions and output of each neuron are being calculated. The backward phase starts by calculating the output error, which is the difference between the actual and the desired target. The error is propagated back through all the layers in the opposite direction to update the weights accordingly. The correction $\Delta w_{ji}(n)$ that is applied to the weight connecting neuron *(i)* to neuron *(j)* is defined in the delta rule [42]:

$$\begin{pmatrix} \textit{Weight} \\ \textit{correction} \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} \textit{learning-} \\ \textit{rate parameter} \\ \eta \end{pmatrix} \times \begin{pmatrix} \textit{local} \\ \textit{gradient} \\ \delta_j(n) \end{pmatrix} \times \begin{pmatrix} \textit{input signal} \\ \textit{of neuron } j, \\ y_i(n) \end{pmatrix} \qquad (2.3)$$

The local gradient $\delta_j(n)$ depends on whether neuron *(j)* is an output or hidden node. If it is an output node, $\delta_j(n)$ is equal to the product of the derivative $\varphi'_j(v_j(n))$ and the error $e_j(n)$, which are both associated to the neuron *j* as follows:

$$\delta_j(n) = e_j(n)\ \varphi'_j(v_j(n)) \qquad (2.4)$$

If neuron *j* is a hidden node, $\delta_j(n)$ *is* equal to the product of the derivative $\varphi'_j(v_j(n))$ and the weighted sum of $\delta s$ that is computed for the neurons in the next output or hidden layer that are connected to neuron *j*, where neuron k is an output node [38]:

$$\delta_j(n) = \varphi'_j(v_j(n))\ \Sigma_k\ \delta_k(n)\ w_{kj}(n) \qquad (2.5)$$

The stochastic gradient method is the most preferred method of online implementation of backpropagation. Figure 2.7 represents this method graphically. The optimal value for a weight is when the error achieves the global minimum. Weights are typically updated in small steps in a continuous attempt to reach the global minimum. However, it is practically a difficult task as it often ends up in a local minimum [42].

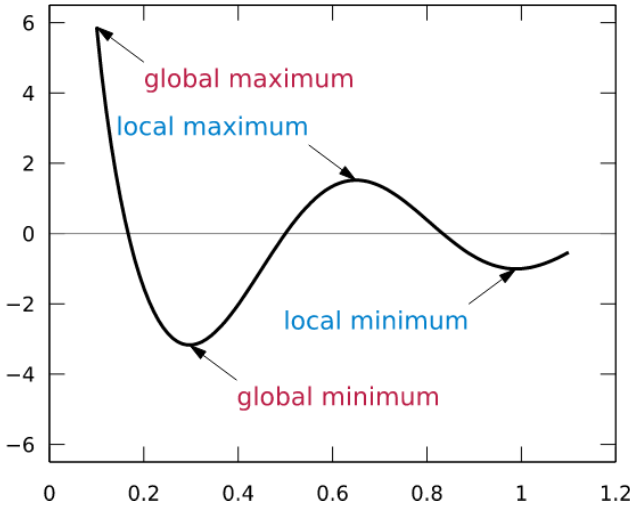


***Figure 2.7 Stochastic gradient descent [32]***

### *2.2.4 Cost Function*

A Feedforward Neural Network typically consists of multiple layers. The input propagates through the network until an output vector is returned. The output vector is called $a_j^i$ the activation of the $j^{th}$ neuron in the $i^{th}$ layer, where $a^{\mathrm{i}}$ is the $j^{th}$ element of the input vector. Then the input of the next layer is related to its previous via the following relation [43]:

$$a_j^i = \sigma \left( \Sigma_k \left( w_{jk}^i \,.\, a_k^{i-1} \right) + b_j^i \right) \tag{2.6}$$

where σ is the activation function, $w^i$ is the weight from the $k^{th}$ neuron in the $(i-1)^{th}$ layer to the $j^{th}$ neuron in the $i^{th}$ layer. $b_j^i$ is the bias of the $j^{th}$ neuron in the $i^{th}$ layer while $a_j^i$ represents the activation value of the $j^{th}$ neuron in the $i^{th}$ layer.

The output from a neural network is the predicted label, which assigns the test sample to one of the possible classes of the problem (e.g., Healthy/Faulty). The label is predicted through probabilities, where the sum adds up to 100%. The classifier compares the test sample with the training samples looking for similarities and calculates the probabilities for each class accordingly. The higher similarity between the sample and the class, the higher probability. The class with the highest probability becomes the final label to the test sample.

## *2.3 Random Forest*

Random forest is an accurate, easy to use, non-neural network, Machine Learning (ML) algorithm which has showed great results [44] over the past few years. It is a supervised learning algorithm which consists of decision trees, which form a “forest” (see Figure 2.8) and are trained using the bagging method [56]. The bagging method is where a combination of learning models increases the overall result. Each tree makes an individual decision, and the forest merges all decisions together to get a more accurate and stable prediction. Random forest can be used for classification or regression problems, which form most machine learning methods.

***Figure 2.8 Random Forest***

Random Forest has only been working well with classification of statistical data, or regression of time-series data [45, 46]. Thus, a new pre-processing approach was proposed in this research for the Random Forest (RF) algorithm. This new pre-processing approach enhances the performance of the Random Forest algorithm in classification of time-series data. An example of classification of statistical data using Random Forest is provided by S. Bernard and S. Adam who used Random Forest (RF) to classify the popular handwritten digits dataset "MNIST" [47]. This dataset does not contain time-series data, but rather consists of 10 handwritten digits, dealt with as images. The dataset has a black background, and the digits are written in white. The dataset contains various fonts and way of writing the digits to cover as many scenarios as possible. Random forest has successfully classified the 10 classes with a 98% detection rate. Another example is the popular IRIS dataset. N Azizah, L.S. Riza, and Y. Wirhardi used the IRIS dataset to test the Random Forest algorithm to test parallel computing performance using the R programming language [48]. Same with Wu Y., He J., and Ji Y., et al., who published enhanced models to increase the detection rate of the IRIS dataset [49]. The IRIS dataset is an image-based flower dataset which contains statistical values which describe each specie. Examples of these values are the sepal length, sepal width, petal length and petal width [50].

On the regression side, Random Forest was mostly used in forecasting. Mei J., He D., Harley R.G., et al. used RF to predict the electricity price in New York on a real-time basis [51]. Dudek G. forecasted short-term load using Random Forest [52], while Buchwitz B., Falkenberg A., and Kusters U. used Random Forest to forecast time-series events in consumer electronic markets [53]. The way time-series forecasting works, is that the Random Forest model takes the last few data points to predict the current value. For data samples with large number of data points, the algorithm keeps predicting one point at a time, and compares with the actual value. By the end of the training, the prediction accuracy increases, and the model can predict values in the near future. Based on the predict values, it predicts further points. The longer the predicted period, the less accuracy.

Time-series classification in Random Forest works differently. The model must receive all the data points of the sample to find similarities within each class. This means the number of input data points per sample can be very large in case of engines and motors which can generate millions of data points in a few minutes. This does not match with the nature of Random Forest. Rodriguez J., and Kuncheva L. [54] compared the performance of decision trees, SVM on time-series interval and DTW features. Random Forest was not the best approach for this application. Same with Goehry B., Yan H. and Goude Y., et al. who tried Random Forests for time-series [55].

A new pre-processing approach is proposed later in this research to solve the limitations above. The approach is referred to as the curve description technique. This technique provides statistical features of time-series data instead of fitting raw data points into the random forest algorithm. Not only that it opens a new path for time-series classification in Random Forest, but it also solves a very difficult Fault Detection and Diagnosis (FDD) problem of an Internal Combustion Engine (ICE). Random forest (Figure 2.9) is a fast, accurate and computationally light algorithm which competes with neural networks in providing data-driven solutions to easy problems. One of the reasons why this algorithm is popular is because it is simple and diverse and can be used for both classification and regression tasks.

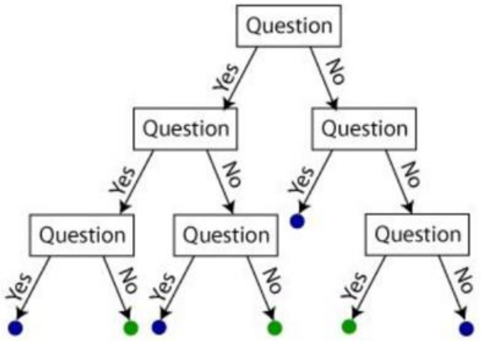


*Figure 2.9 Random Forest logic*

Random forest (RF) shares the same hyperparameters as traditional decision trees and bagging classifiers. Since the classifier-class of random forest can be used, it is not necessary to combine a decision tree with a bagging classifier. For regression tasks, the regressor of Random Forest can be used instead. While growing the trees, RF adds further randomness to the model by searching for the best feature among a random subset of features instead of searching for the most important feature while splitting a node. This increases the diversity of the model which affects the results positively. Thus, the algorithm takes only a random subset of the features into consideration for splitting a node. To add more randomness to the model, random thresholds can be used for each feature rather than searching for the best possible thresholds, which is what happens in a normal decision tree [57]. Figure 2.10 shows a random forest with two trees.

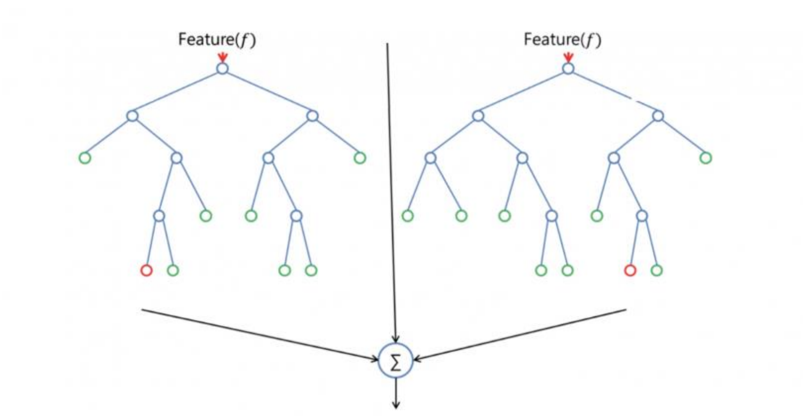
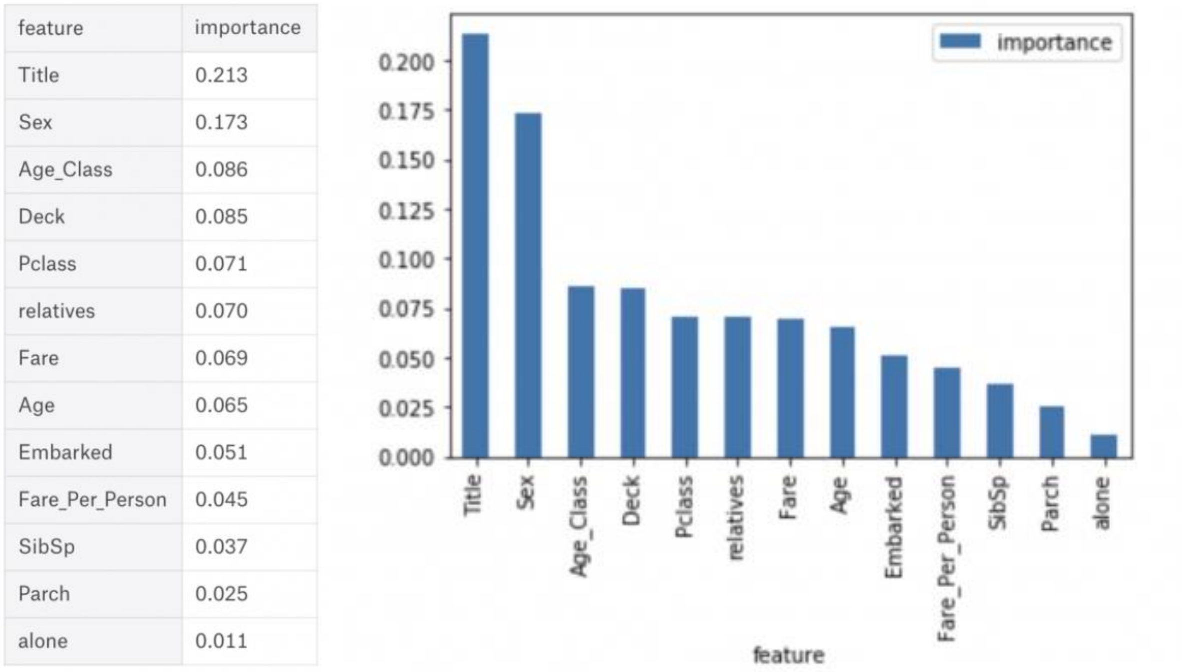


*Figure 2.10 Random Forest trees*

A great option in the random forest algorithm is that it provides importance ratings of each feature on the prediction. One of the most popular libraries which offer random forest models is Scikit-Learn (a.k.a. Sklearn) [58]. In Sklearn, a feature importance tool is provided which measures the importance of every feature by looking at how much it affected the results among all trees in the forest. This score is computed automatically for each feature after training and the results are scaled so that the sum of all ratings is equal to one.

Each node in a decision tree represents a prediction. Each branch represents the outcome of the prediction, and each leaf node represents the label. The label is the decision taken after calculating all attributes. The feature importance tool helps to decide about what features to keep and what features to drop if they do not contribute to the prediction. This tool is very important to avoid overfitting due to redundant features, as well as eliminate computations and the running time. Figure 2.11 shows a table and a chart with an example of the feature importance tool of 13 features from the famous Titanic dataset on Kaggle [59].



| feature | importance |
|---|---|
| Title | 0.213 |
| Sex | 0.173 |
| Age_Class | 0.086 |
| Deck | 0.085 |
| Pclass | 0.071 |
| relatives | 0.070 |
| Fare | 0.069 |
| Age | 0.065 |
| Embarked | 0.051 |
| Fare_Per_Person | 0.045 |
| SibSp | 0.037 |
| Parch | 0.025 |
| alone | 0.011 |

***Figure 2.11 Example of feature importance tool using Titanic dataset***

Although the Random Forest (RF) algorithm consists of decision trees, there are differences. Once a decision tree is trained on features and labels, it formulates a set of rules, which will be used to make predictions. For example, predicting whether a person will click on an online advertisement, the ads the person clicked on in the past are important features to train the tree on. Also, any features that describe their decision would be important features. Once trained on these features along with their corresponding labels, the decision tree will generate some rules that help predict whether the person will click on specific ads or not. On the other hand, the Random Forest algorithm randomly selects observations and features to build several decision trees where most decisions will take over the final label. However, deep decision trees often suffer from overfitting. Random forest avoids overfitting by creating random subsets of the features and building smaller trees using those subsets. Then, the subtrees are re-combined.

The hyperparameters in random forest are used to increase the performance of the model and to decrease the running time [60]. One important hyperparameter is the "n_estimators", which represents the number of decision trees in the algorithm builds. Generally, higher number of trees increases the performance of the model and makes more stable predictions. However, more decision trees also slow down the computation. The "max_features**"** hyperparameter is the maximum number of features which can be considered to split a node. Another important hyperparameter is **"**min_sample_leaf" which is the minimum number of leaves needed to split a node. The "n_jobs" hyperparameter determines the number of processors allowed to use. If n_jobs is equal to "1", only one processor will be used, while "-1" means there is no limit on processors. The "random_state" hyperparameter makes the output of the model reproducible, which means the model always produces the same results when "random_state" is set to a definite value, given the same hyperparameters and training data. Finally, the "oob_score" hyperparameter, a.k.a. oob sampling, is a cross-validation approach. When set on, almost one-third of the data is not used for training but rather used for validation. This is what is called "validation set", or the "out-of-bag" samples. This hyperparameter is very similar to the leave-one-out-cross-validation method with no additional computational.

Like any algorithm, the random forest algorithm has advantages and disadvantages. One big advantage is that it can be used for both regression and classification tasks, as well as the feature importance tool which sorts out the features based on their contribution to the generalized rules and results. Another important advantage is that the algorithm is normally set to the best settings based on previous experiments. This means that the algorithm requires minimum parameter tuning since the default hyperparameters often produce good prediction results. Furthermore, understanding the hyperparameters is straightforward. Although overfitting is a common problem in Machine Learning, this will not happen in random forest if there are enough trees in the forest.

The biggest disadvantage of the random forest algorithm is that the large number of trees may make the algorithm too slow for real-time predictions. The algorithm is fast to train; however, it is slow to create predictions once they are trained. The more trees, the more accurate predictions, the slower model. Thus, Random Forest would not be the best algorithm for real-world real-time applications and other methods would be preferred.  Also, RF is a predictive modeling tool and not a descriptive tool. This means it would not be the best approach when looking for a description of the relationships within the data. Random Forest is often used in a lot of fields, such as stock market predictions, banking, e-commerce and even medicine. In finance, it is used to detect customers who are likely to pay their debt on time, or use certain services more frequently, and detect fraudsters. Predicting the future behavior of the stock market is a very common application where RF is used. In healthcare, it can be used to identify the correct components in medicine and analyze the medical history of patients to diagnose diseases. In e-commerce, RF is used to determine whether a customer will like a certain product.

## *2.4 Extended Multi-Scale Principal Component Analysis (EMSPCA)*

The Extended Multi-Scale Principal Component Analysis (EMSPCA) is a feature extraction method which was developed previously at McMaster’s Center for Mechatronics and Hybrid Technology (CMHT) [61]. The purpose of this method is to generate fault signatures from test samples. A fault signature is a measurement of the deviation of the test sample from healthy baselines (i.e., references). To achieve this purpose, the input data goes through three main stages: normalization, Wavelet Pack Transform (WPT), and Principal Component Analysis (PCA). The resultant fault signatures are then used as input to an Artificial Neural Network (ANN) which is responsible for classification. Figure 2.12 shows the block diagram of the EMSPCA method.

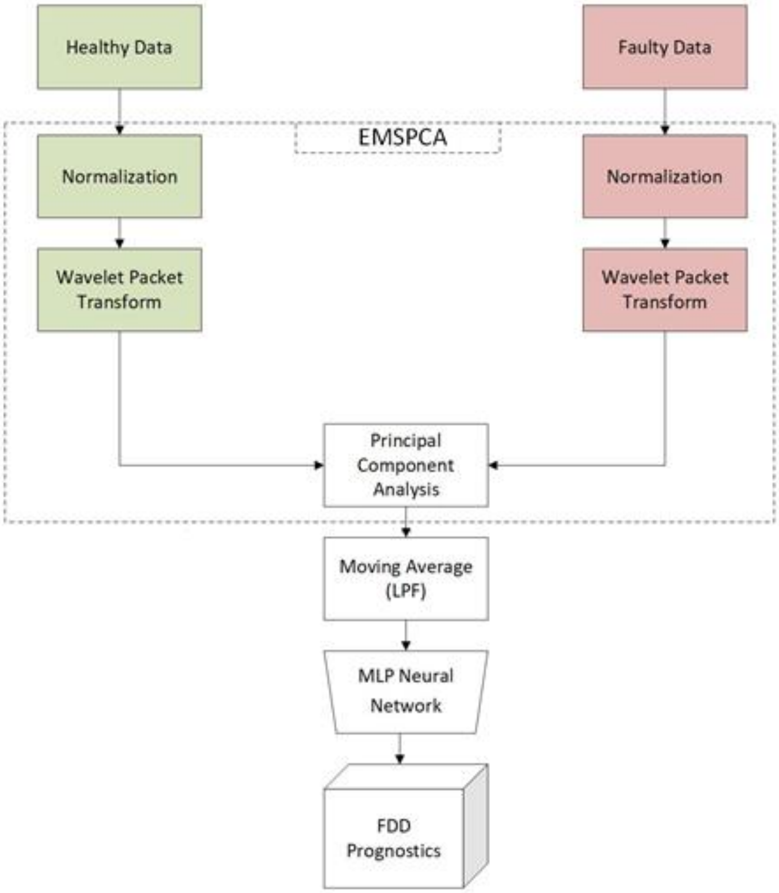


*Fig (2.12) EMSPCA Diagram*

The EMSPCA method starts with normalization of both healthy baselines and faulty samples. The goal of normalization is to change all data points in the dataset to a common scale. Normalization is strongly required when different set of features have different ranges. The motive behind this step is to help the EMSPCA algorithm detect faults accurately in the case of different measurement types. This normalization is applied as follows in Equation 2.1:

$$Normalized\ signal\ = \frac{raw\ signal\ -\ mean\ value\ (baseline\ center)}{Variance\ (baseline\ raw\ signal)} \qquad (2.1)$$

It is very important to use the baseline centerlines and variances in normalizing new observations, which is implemented in Equation 2.1. The next step was to apply Wavelet Packet Transform (WPT) to the normalized data. WPT converts data from its current domain (e.g., time or crank angle) to the frequency domain and splits the original signal into smaller portions, namely frequency components.

### 2.4.1 Wavelet Packet Transform (WPT)

The EMSPCA method was inspired by the Modified Multi-Scale Principal Component Analysis (Mod-MSPCA) [61], but the difference was in the wavelet stage. The Mod-MSPCA consists of Discrete Wavelet Transform (DWT) and Principal Component Analysis (PCA), while the EMSPCA consists of Wavelet Packet Transform (WPT) and PCA. Mod-MSPCA was developed to enable fault diagnosis through contribution charts in individual frequency levels. In DWT, the length of the signal is reduced at each level. Figure 2.13 shows a signal of 128 samples (i.e., components).

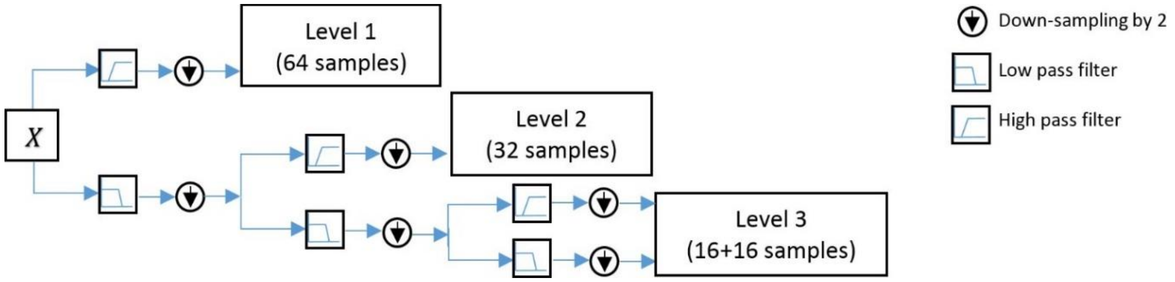


*Figure 2.13 DWT of a 128-samples signal [61]*

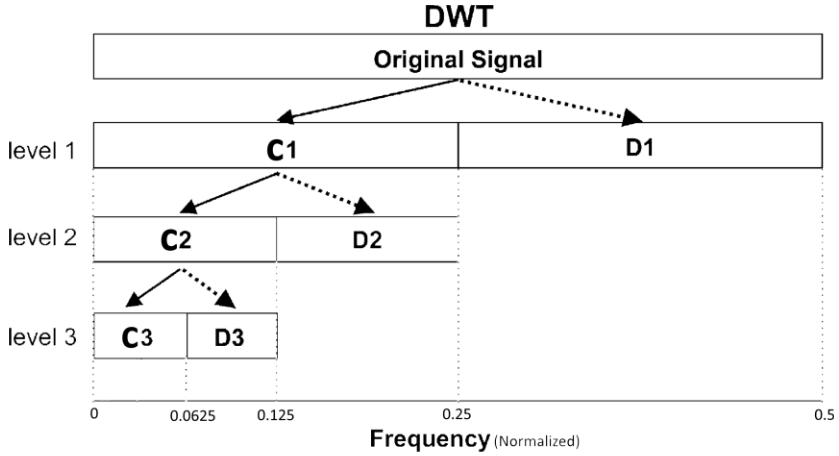


*Figure 2.14 Frequency contents of DWT coefficients [61]*

At each level, the signal is broken down into frequency levels with reduction in sample size. The full length of each level coefficients is recovered using wavelet reconstruction filters. This means that lower frequencies have narrower widths, which becomes wide again at higher frequencies as illustrated in Figure 2.14. For FDD applications where high frequency contents of the signal are rich in information, DWT strategy would result in loss of some of this information. Thus, another version of wavelet analysis, namely, Wavelet Packet Transformation (WPT) was used instead of DWT in the Extended Multi-Scale Principal Component Analysis (EMSPCA) method to avoid loss of information. The structure of WPT is shown in Figure 2.15.

In WPT, both the outputs of high and low pass filters are broken down to the next level, while in DWT, only the output of low pass filter is considered. Each level breaks down the frequency content into two frequency bands. The first is a high frequency band, which results from the high pass filter, while the second is a low frequency band, which is represented by approximation coefficients and results from the low pass filter. As shown in Figure 2.16, each frequency content splits in half in every level of wavelets. This process iterates multiple times depending on the assigned level of wavelets. The outputs of WPT of a test sample are smaller frequency bins which are then compared to those of healthy baselines [61]. Figure 2.15 shows the WPT method.

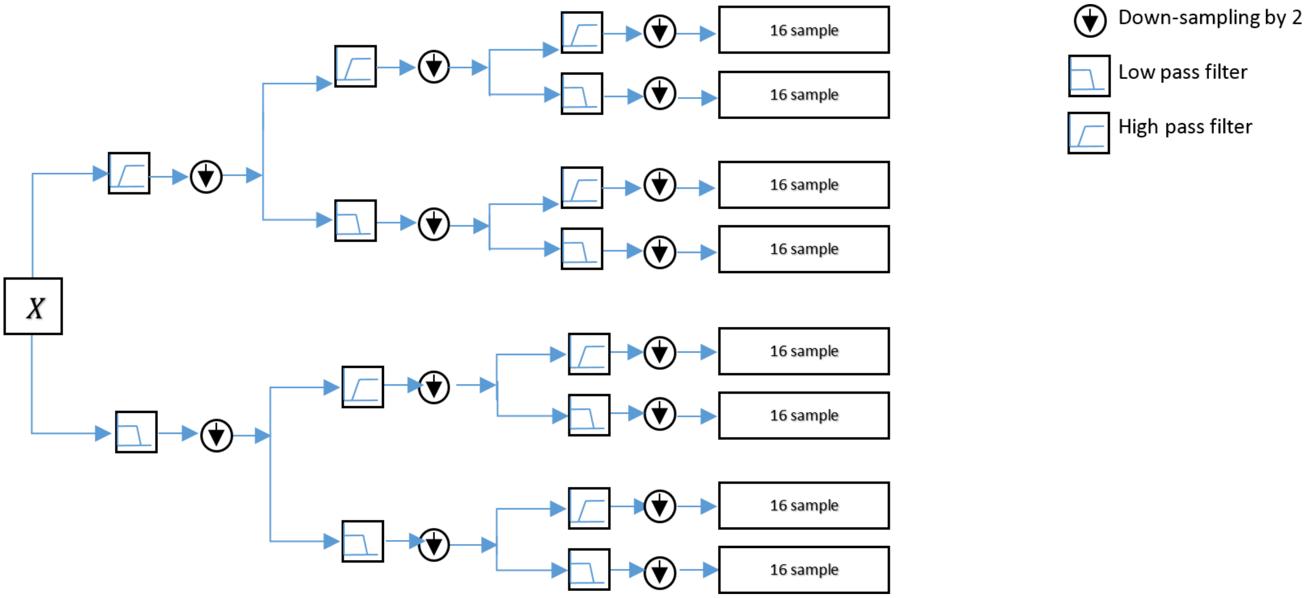


*Figure 2.15 WPT of 128 samples signal [61]*

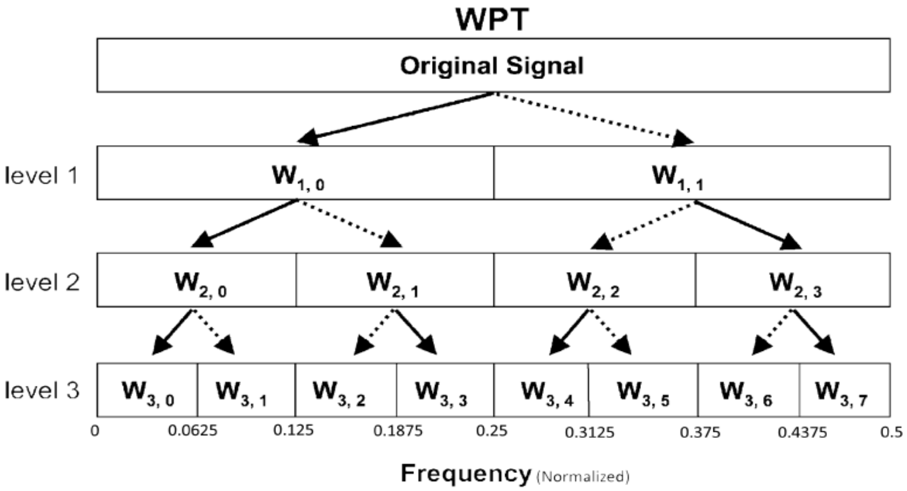


*Figure 2.16 Frequency content of WPT coefficients*

Mathematically, Equations 2.2 and 2.3 describe DWT, where the approximation and detail coefficients ($c_{j+1}$ and $d_{j+1}$) of the next wavelet level use the approximation coefficients ($c_j$ and $d_j$) of the current level. These equations breakdown the frequency content in different levels. Each level splits the frequency content into 2 frequency bands: high frequency band (output of high pass filter), and low frequency band (output of low pass filter).

$$c_{j+1}[n] = c_j \ast G_j[2n] \qquad (2.2)$$

$$d_{j+1}[n] = c_j \ast H_j[2n] \qquad (2.3)$$

In WPT, both approximation and detail coefficients are used in calculating next level coefficients, this is described mathematically in Equations 2.4 and 2.5:

$$W_{j+1,2k}[n] = W_{j,k} \ast G_j[2n] \qquad (2.4)$$

$$W_{j+1,2k+1}[n] = W_{j,k} \ast H_j[2n] \qquad (2.5)$$

where $W_{j,k}$ represents the coefficients at level $j$ for the atom $k$. The low pass filter results in even $k$, while the high pass filter results in an odd $k$. At the last level, these Equations result in similar frequency bandwidths for all atoms ($W_{j,k}$). To feed the coefficients to a Principal Component analysis (PCA) algorithm, a similar methodology to Mod-MSPCA was used where the coefficients were reconstructed using the filters of the WPT. The summation of the recovered signal components is the original signal as shown in Equation 2.6:

$$W_S W_j X = H_1^t H_1 X + G_1^t G_1 X + H_2^t H_2 X + \cdots + H_j^t H_j X + G_j^t H_j X$$

$$= X_1 + X_2 + \cdots + X_{2j-1} + X_{2j} \qquad (2.6)$$

Therefore, applying PCA analysis on each component of the signal ($X_i$), allows for fault detection and diagnosis in narrower frequency bands of measurements spectrum compared to Mod-MSPCA frequency bands.

### *2.4.2 Principal Component Analysis (PCA)*

After applying Wavelet Packet Transform (WPT) to the normalized raw data, Principal Component Analysis (PCA) [63] is then applied to each batch of resultant frequency components. PCA reduces the dimensionality of a data sample by compressing its current dimension size into a smaller dimension. Dimensionality reduction helps discard redundant information and keep only significant patterns to get a more meaningful representation of the sample. PCA uses orthogonal transformation to convert a dependent set of measurements into a set of independent variables. The deduced variables are named Principal Components (PC), which carry a compact representation of the measured system behavior. Considering a set of dependent measurements defined in a matrix form as

$$X_{(k)} = (x_1, \ldots, x_p)_{(k)} \tag{2.7}$$

where $p$ is the number of variables and $k$ is the number of measurements for each variable. Each column in this matrix represents $k$ measurements for a single sensor, such as $x_1$. A transformation is applied as follows:

$$T = XP \tag{2.8}$$

$P$ is the Principal Components loading matrix, which when multiplied by $X$, $X$ is transformed into its PC or Principal Components scores $T$. Principle Components scores $T$ are the uncorrelated signals to the cross-correlated signals $X$. Each column of the matrix represents $k$ measurements for one variable $t_i$. The first column includes the highest variance component $t_1$, and components variance decreases to the lowest variance component $t_p$. This means that the first few Principal Components (PC) carry the highest amount of information. The last few PC are low variance variables, hence carrying the least amount of information. This helps the PCA identify the measurement signals which contain the fault signature and eliminate those that do not have significant information. Geometrically, the transformation in Equation 2.8 can be illustrated by mapping the measurements onto orthogonal axes.

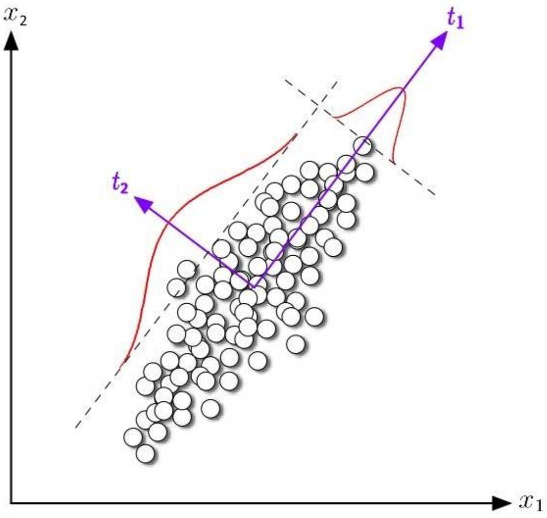


***Figure 2.17 Principal component analysis [61]***

The visualization of this concept is shown in Figure 2.17, where $x_1, x_2$ are inputs and $t_1, t_2$ are orthogonal output PCs, which means both $t_1$ and $t_2$ are uncorrelated. This allows each component to represent a different event. FDD algorithms can better detect faults if any of the main characteristics changed. In Figure 2.17, $t_1$ has higher variance which means that it contains more information about system and contributes more to the measurementsthan $t_2$. To compute the matrix $T$, a transformation matrix $P$ should be used, which defines the mathematical bases for the new orthogonal dimensions. Mathematical bases should result in uncorrelated outputs. The sample covariance of the transformation matrix $T$ can be found as follows:

$$\begin{aligned}\Sigma_T &= \frac{1}{n-1}T^tT \\ &= \frac{1}{n-1}(XP)^tXP \\ &= \frac{1}{n-1}P^t(X^tX)P \\ &= \frac{1}{n-1}P^t(X^tX)P \qquad (2.9)\end{aligned}$$

In Equation 2.9, the term $(X^tX)$ results in a symmetric matrix:

$$S = X^tX \qquad (2.10)$$

Since $S$ is symmetric, it can be decomposed into eigenvalues and eigenvectors.

$$S = B\Lambda B^{-1} \qquad (2.11)$$

where $B$ represents the eigenvectors, and $\Lambda$ is a diagonal matrix of eigenvalues. Since $B$ is orthogonal, its inverse is equal to its transpose $(B^{-1} = B^t)$. Therefore,

$$\Sigma_T = \frac{1}{n-1}P^t(B\Lambda B^t)\,P \qquad (2.12)$$

To avoid any numerical errors while finding the eigenvectors, Singular Value Decomposition (SVD) is used, which is a factorization method of matrices. SVD factors any matrix to three components: left singular matrix $U$, right singular matrix $V$, and singular values matrix $\Sigma$. The factorization for a given matrix $X$ is shown in Equation 2.13.

$$X = U\Sigma V^{t} \tag{2.13}$$

Equation 2.13 is used to calculate PCA transformation matrix $P$ and $SVD$ is applied as follows:

$$X^{t}X = V\Sigma^{t}U^{t}U\Sigma V^{t} \tag{2.14}$$

$U$ is a unitary matrix, which means that $U^{t} = U^{-1}$, which this reduces Equation 2.14 to:

$$X^{t}X = V\Sigma^{t}\Sigma V^{t} \tag{2.15}$$

The singular valuesmatrix $\Sigma$ in Equation 2.13 is the square root of the eigenvalues of ($X^{t}X$). Thus, Equation 2.15 can be rewritten as:

$$X^{t}X = V\Lambda V^{t} \tag{2.16}$$

Comparing Equation 2.16 with Equations 2.10 and 2.11, it can be found that $B = V$, and since $P = B$ to have a diagonal $\Sigma_{T}$, the transformation matrix $P$ is equal to SVD right singular matrix $V$. SVD assures that the eigenvalues, and the eigenvectors are sorted in a descending order. Thus, the output PC are assured to be sorted with the highest variance variable first ($t_1$) and decreasing to the lowest variance variable ($t_p$).

The output from PCA consists of significant principal components which best describes the original signal. In FDD applications, these principal components are then compared to the baseline (a.k.a. healthy sample) to measure the deviation of the test sample from the reference.

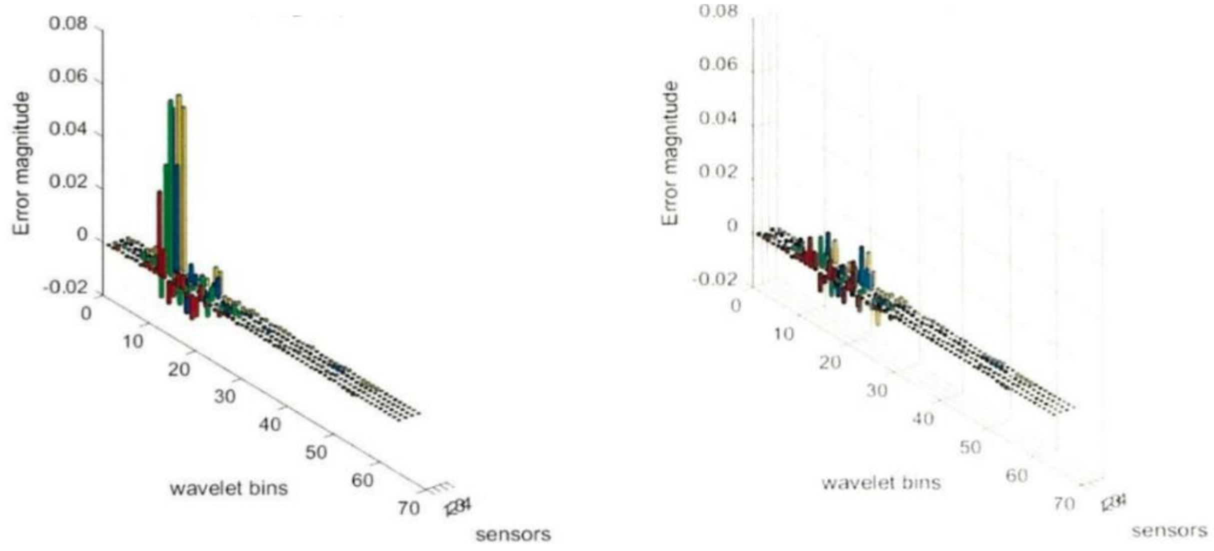


***Figure 2.18 Fault signature of a fault signal (left) against its healthy baseline (right)***

Figure 2.18 shows an example of a fault signature which describes how far the sample varies from the baseline. A fault signature consists of a bar chart which contains principal components of the original signal. The size of the fault signature depends on the level of wavelets applied to the original signal. The four rows in the z-direction represent the features coming from the sensors, each row represents a single sensor. The number of bins in each row depends on the level of wavelet used in the wavelet packet transform (WPT) process which was conducted in the EMSPCA algorithm. The number of bins is equal to 2 ^ (wavelet level). For example, for a fault signature of wavelet level 6 like the ones in Figure 2.18, the number of bins is 2 ^ 6 = 64 bins. The amplitude of each bin represents the error magnitude from healthy baselines.

### 2.4.3 Multi-layer Perceptron

Fault signatures of different classes (e.g., fault conditions) emphasize distinct patterns in each class. These patterns are used as input data to the Machine Learning Algorithm for training, which is an Artificial Neural Network (ANN), as shown in Figure 2.19. The ANN typically consists of a single input layer, multiple hidden layers, and a single output layer. The input layer consists of units (i.e., neurons). The number of units is equivalent to the number of principal components of each fault signature. The weights are calculated in the hidden layers. The number of hidden layers is set by the user and is adjustable. The output layer consists of units (i.e., neurons). Each unit provides a probability for the test sample to belong to one of the classes.

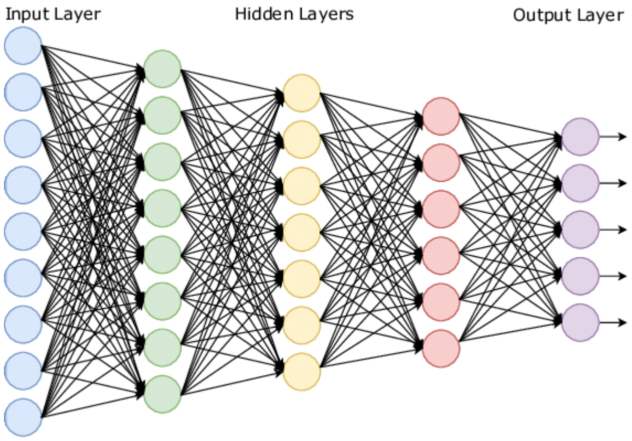


*Figure 2.19 Multi-layer Perceptron*

The fault signatures were fed to the Artificial Neural Network (ANN) by fitting every data point in an input neuron. The size of the input layer was equivalent to the size of the fault signature. For example, the fault signature in Figure 2.18 had 64 bins. Thus, the input layer of the ANN consisted of 64 neurons. Based on the weights calculated in the hidden layers, a probability was predicted for each class in the output layer. The class which achieved the highest probability was voted as the final label.

## *2.5 Fault Detection and Diagnosis in ICE Using Deep Learning*

In 2020, Zhang, et al. [64] summarized existing literature on bearing fault diagnostics using conventional machine learning methods and state-of-the-art deep learning (DL) methods. The literature included a wide variety of successful FDD models using feature extraction techniques such as Convolutional Neural Networks (CNN), Sparse Auto-Encoders (SAE), Recurrent Neural Networks (RNN) and SVM. Many methods achieved an accuracy of more than 91%. Jia et al. [65] achieved an accuracy of 99.92% in detecting bearing faults using an SAE-local connection network.

### *2.5.1 Recurrent Neural Networks (RNN)*

Recurrent neural networks are designed to recognize patterns in sequences of data such as numerical times series sensory data, stock markets and text translation. One of the most powerful sub-categories of RNNs is the Long Short-Term Memory (LSTM) [66] which was used in this project. What differentiates RNNs and LSTMs from other neural networks is that they take time and sequence into account. Recurrent networks are distinguished from feedforward networks by that feedback loop connected to their past decisions. This feedback loop effectively provides, RNNs with memory and enables them to take advantage of the information in the sequence itself to perform tasks that feedforward networks cannot [67]. An example of these tasks is finding correlations between events separated in time (i.e., long-term dependencies) since an event can be a function of one or more events that came before. That sequential information is preserved in the RNN’s hidden state which manages to span across time steps as it moves forward to affect the processing of each new example. In other words, RNNs can share weights over time.

RNNs take as their input not just the current input example they see, but also what they have perceived previously in time. The decision of a recurrent network reached at time step *t* affects the decision it will reach later step. So recurrent networks have two sources of input, the present and the recent past, which combine to determine how they respond to new data. The process of carrying memory forward mathematically is as follows:

$$h_{t+1} = \varphi\ (Wx_{t+1} + Uh_t) \tag{2.7}$$

The hidden state at time step *t+1* is $h_{t+1}$. It is a function of the input at the same time step $x_{t+1}$, modified by a weight matrix *W* added to the hidden state of the previous time step $h_t$ multiplied by its own hidden-state-to-hidden-state matrix *U*, otherwise known as a transition matrix. The weight matrices are filters that determine how much importance to accord to both the present input and the past hidden state. The error they generate will return via backpropagation and be used to adjust their weights until error cannot go any lower. The sum of the weight input and hidden state is then pitched into the activation function φ (e.g., sigmoid, tanh) which is a standard tool for condensing very large or very small values into a logistic space.

In the diagram below, each *x* is an input example, *W* is the weights that filter inputs, *a* is the activation of the hidden layer which is a combination of weighted inputs and the previous hidden states, while *b* is the output of the hidden layer after it has been transformed, using a rectified linear or sigmoidal unit. Figure 2.20 shows the structure of the LSTM model.

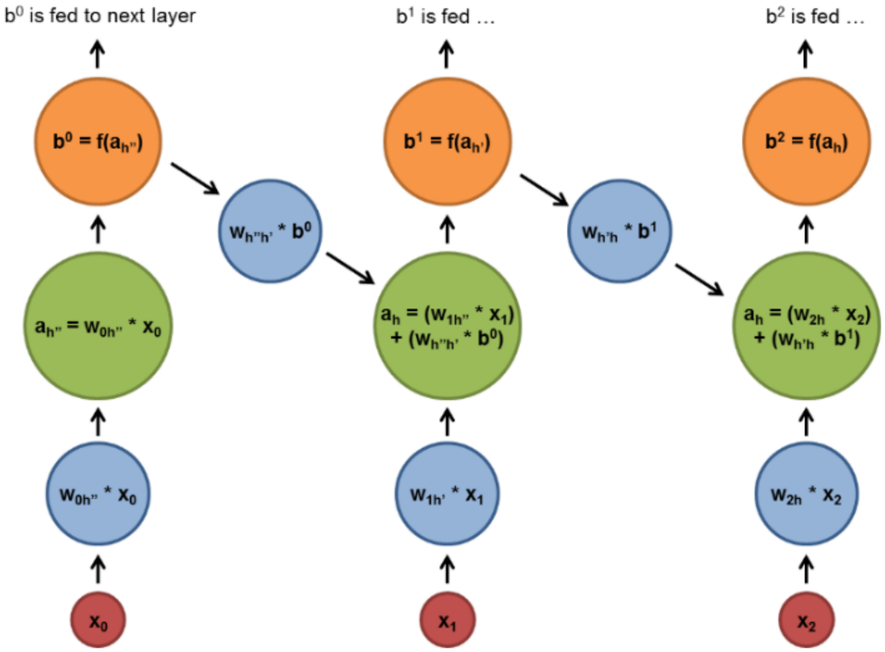


***Figure 2.20 Recurrent neural networks***

LSTM-RNN was used as a fault detection and diagnosis (FDD) tool and predictive maintenance [68, 74-76] in applications such as bearings [69, 70], rotary machinery [71], satellite reaction flywheel [72] and transmission lines [73]. However, it was never used to detect spark plug degradation using engine built-in sensors.

### *2.5.2 Convolutional Neural Networks (CNN)*

Convolutional neural networks are used primarily to classify images [77] and perform object recognition within scenes such as identifying faces, street signs, tumors, and many other aspects of visual data. CNNs are powering major advances in computer vision which has applications for self-driving cars, robotics, drones, security, and medical diagnoses. CNNs have been also applied to text analytics [78, 79] as well as sound when it is represented visually as a spectrogram, and graph data with graph convolutional networks.

Convolutional neural networks ingest and process images as tensors [80], which are matrices of numbers with additional dimensions (see Figure 2.21). Tensors are formed by arrays nested within arrays, and that nesting can go on infinitely, accounting for an arbitrary number of dimensions far greater than what we can visualize spatially. A 4-D tensor would simply replace each of these scalars with an array nested one level deeper.

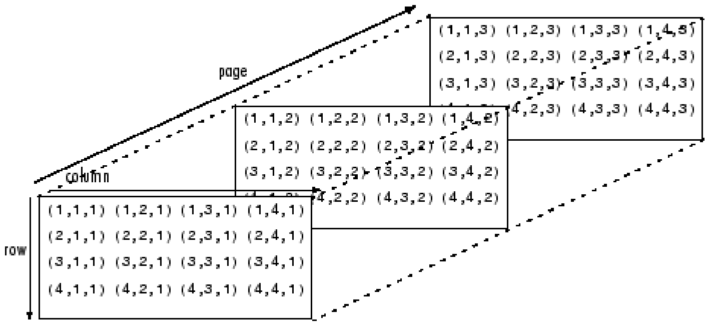


*Figure 2.21 3D tensor*

Unlike humans, convolutional networks perceive images as volumes (i.e., 3D objects), rather than 2D flat planes with width and height. That is because digital color images have a red-blue-green (RGB) encoding [80], mixing those colors to produce the color spectrum humans perceive. A convolutional network ingests such images as three separate channels (i.e., layers) of color stacked on top of each other as follows:

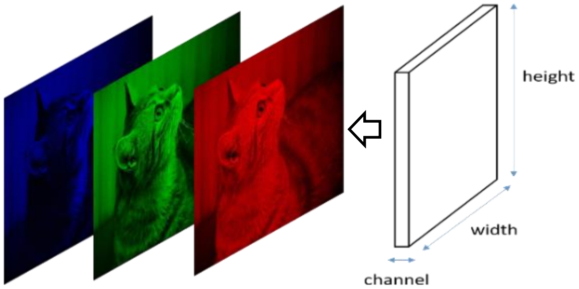


*Figure 2.22 RGB channels vs original image*

Each of the 3 layers consists of several pixels which is equal to the product of width and height. For each pixel of an image, the intensity of R, G and B is presented by a number as in Figure 2.22. Those numbers are the raw sensory features being fed into the convolutional network which finds which of those numbers are significant signals that help it classify images more accurately. Rather than focus on one pixel at a time, a convolutional net takes in square patches of pixels and passes them through a *filter* (i.e., kernel) which is responsible for finding patterns in the pixels. That filter is a square matrix that is smaller than the original image and is equal in size to the patch.

The dot product of the filter against the patch in the upper left-hand corner of the image channel is taken. If the two matrices have high values in the same positions, the dot product's output will be high, otherwise it will be low. This way, a single value (i.e., the output of the dot product) indicates whether the pixel pattern in the underlying image matches the pixel pattern expressed by the filter. Figure 2.23 shows the activation maps.

The filter is then moved across the image step by step until it reaches the upper right-hand corner. The size of the moving step is known as *stride*. The filter can move to the right one column at a time, or in larger steps. At each step, the dot product is taken, and the results are placed in a matrix known as an *activation map*. The width (i.e., number of columns) of the activation map is equal to the number of steps the filter takes to traverse the underlying image. Since larger strides lead to fewer steps, which means a smaller activation map, less computations, and less processing time. Convolutional networks are designed to reduce the dimensionality of images in a variety of ways. Filter stride is one way to reduce dimensionality. Another way is through down-sampling.

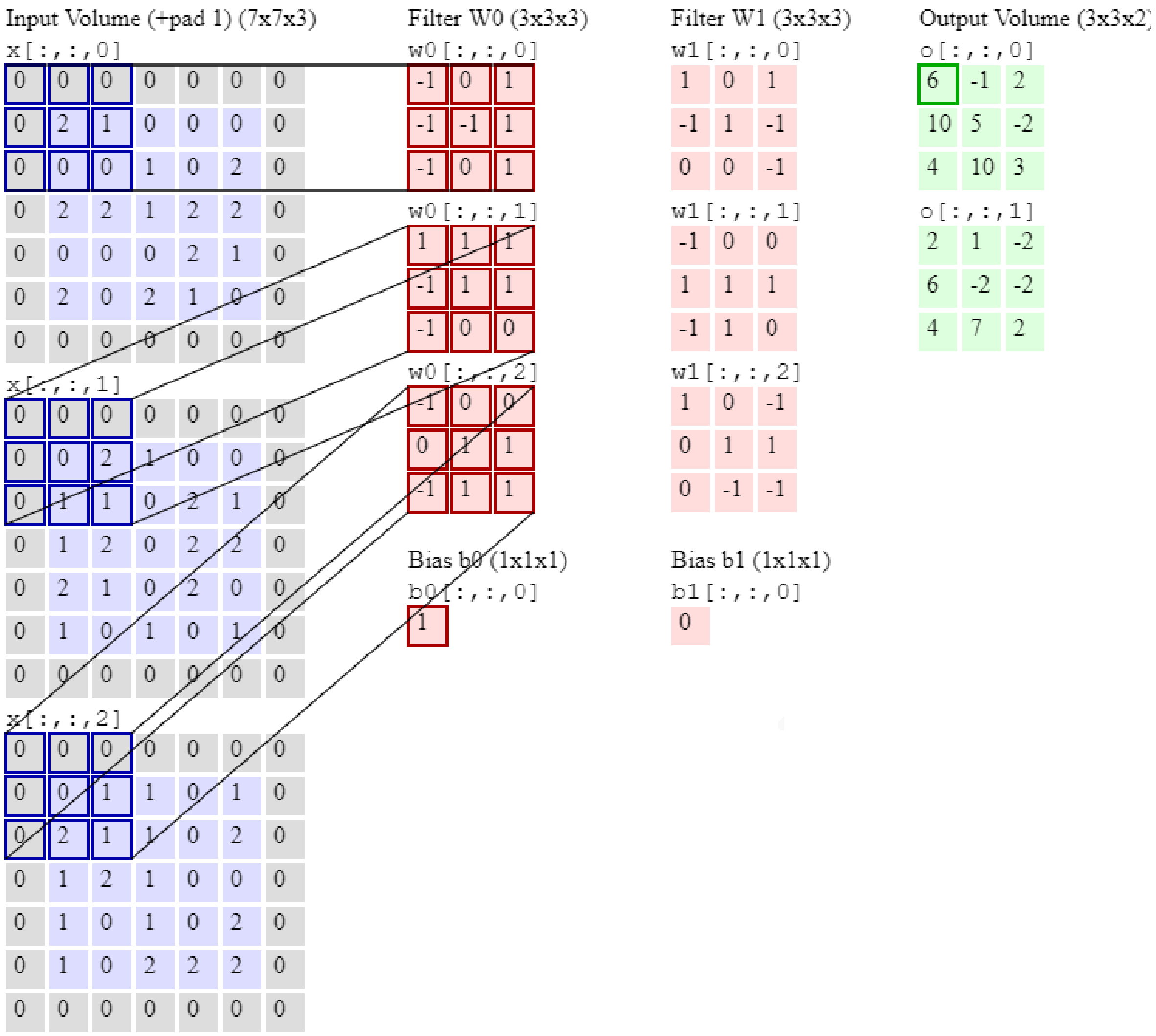


***Figure 2.23 Activation maps***

The next layer in a convolutional network has 3 names: Max pooling, down sampling (fig 2.24) or sub-sampling. The activation maps are fed into a down-sampling layer, and like convolutions, this method is applied one patch at a time. In this case, max pooling simply takes the largest values (i.e., strongest correlation) only from each patch of an image, places them in a new matrix, and discards the rest of the information contained in the activation maps.

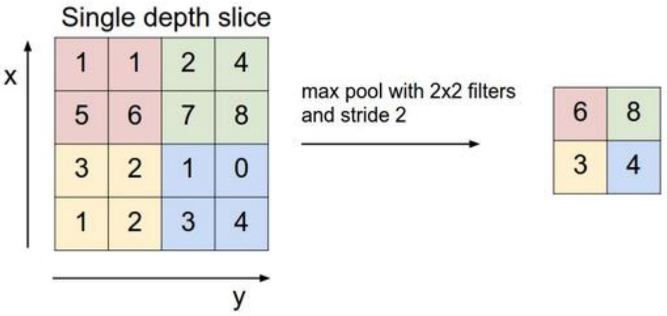


***Figure 2.24 Max pooling (i.e., down-sampling) in CNN***

Only the locations on the image that showed the strongest correlation to each feature (the maximum value) are preserved, and those maximum values combine to form a lower-dimensional space. Figure 2.25 shows an example of a typical convolutional network, which was used in this research. From left to right:

a. The original input image, where the light rectangle is the filter that passes over it.

b. Activation maps stacked one on top of the other, one for each filter. The larger rectangle is one patch to be down sampled.

c. The activation maps condensed through down-sampling.

d. A new set of activation maps created by passing filters over the first down-sampled stack.

e. The second down-sampling, which condenses the second set of activation maps.

f. A fully connected layer that classifies output with one label per node.

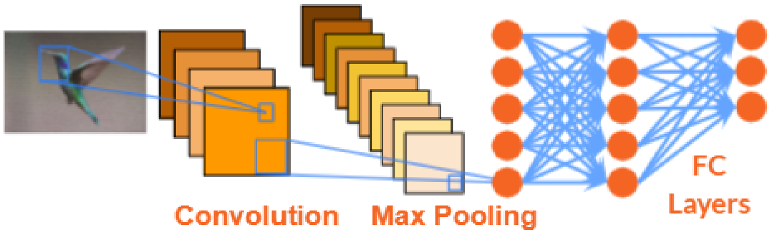


***Figure 2.25 Convolutional neural network diagram***

Convolutional Neural Networks (CNN) have been used in predictive maintenance as a FDD tool in many applications [81-89]. However, it was never used to detect spark plug degradation using engine built-in sensors. In this research, CNN will be used twice to detect spark plug degradation using engine built-in knock sensors. The first time will be using a special kind of heatmaps, which will show great prediction accuracy [71], only for test data which were collected on the same day as the training data. When tested with data collected on other days, the performance of the model drops significantly. The same problem happens when some variability occurs within the collected data due to changes in the atmosphere or experimental issues. That is why a novel contribution will be proposed later in this research which enhanced the performance of the model while testing data from different days as well as variant data. The contribution is the image pre-processing of time series data.

# Chapter 3

# Battery Data Collection

# *Chapter 3 - Battery Data Collection*

All the battery data was collected and stored by Geotab Inc. Their own GO device collects engine data through the OBD II port of the vehicle. The data is then sent to a Store & Forward (SF) server, then stored in a Google Big Query (GBQ) cloud database. In this chapter, the battery data collection and storage process will be discussed in detail. Cranking voltage was collected as the main feature. Other measurements like ambient temperature, engine temperature and coolant temperature would have added important information to the dataset, but unfortunately this data was not available due to Geotab's new data privacy policy (see section 1.1.1 for details).

## *3.1 Battery Data Collection*

GO devices use intelligent patented logging algorithms to decide when to record engine data [90]. This is to produce an accurate representation of the original data by logging the essential points and discarding redundant points (i.e., curve-based logging). Figure 3.1 shows an example of the GO device logging [90]:

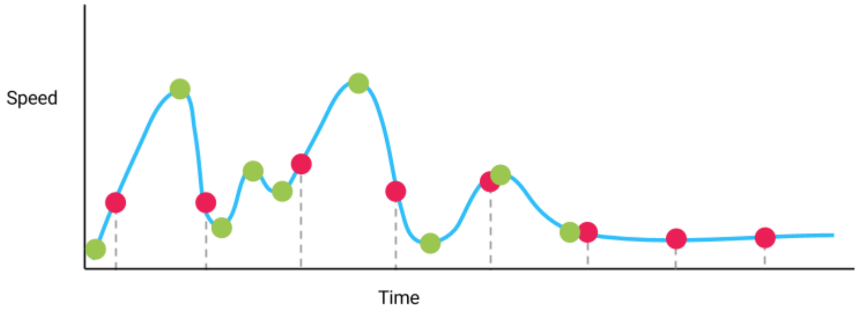


*Figure 3.1 GO device logging [90]*

The actual trip is represented by the blue curve. The red dots represent a simple time-based sampling where a speed value is recorded at regularly spaced time intervals. Some high points and low points would be missed by this approach. Also, when the speed does not change, the time-based approach continues storing the same value multiple times. The green dots represent a more descriptive record of the data with less data points. This is the basic principle that underpins the intelligent logging of the GO device [90].

The battery breakdown measurements were logged the same way as the example above. These measurements were labeled using technical reports which were provided by the repair workshops of two of their customers (a.k.a. Customer A and Customer B). The reports contained the GO device Identification Number (Hardware ID), the Vehicle Identification Number (VIN), time of breakdown and time of battery replacement. The reason of the breakdown and replacement (e.g., periodic maintenance, road call) as well as the technician's comments were also mentioned in the reports. These 2 sources of data (i.e., GO device and technical reports) were used to construct a labeled battery breakdown dataset. The cranking voltages were used as the most relevant input measurement to the machine learning algorithm and as an indicator to the battery health status. The descriptions and comments in the technical report were used for labeling each battery as healthy or unhealthy. This way, the supervised learning model had two separate labeled datasets of healthy batteries and batteries with previous breakdowns, which helps the model generalize and come up with classification rules.

## *3.2 Invalid Battery Data*

Data filtration was the biggest challenge within the data acquisition process. Extracting the data occurred through Google BigQuery which contains both the GO device measurements and the technical reports. BigQuery is a cloud database application powered by Google that requires Structured-Query-Language (SQL)-based queries to pull out datasets. The returns from these queries are tables containing information such as the GO device ID, VIN, date and time, description and cranking voltages needed to train the ML algorithm. The descriptions and comments that described the vehicle maintenance were very tricky and sometimes not clear enough. Some of the descriptions were too short and described nothing about the testing process, and others were very long such that they could not be classified by regular search filters and had to be read manually.  High precision in data filtration is highly recommended to ensure that all the input data feeding the ML algorithm belong to the same class and avoid invalid outputs. Eventually, a total number of only 523 breakdown samples from Customer A were valid as well as 1216 samples from Customer B. A few thousands of healthy samples (i.e., batteries that were never replaced since the GO device was plugged in) were available. All the breakdown samples from each customer were used for training along with an equal number of healthy samples.

As per Geotab's legal policy, it is not allowed to publish individual battery data to protect the privacy of their customers. Figure 3.2 shows a fictitious example of a typical on-duty battery breakdown. All battery signals provided in this thesis are not from real customer data and are manually drawn to help the reader imagine how real data would look like. The red line shows the moving average of the minimum cranking events while the blue circle shows the time of breakdown and replacement.

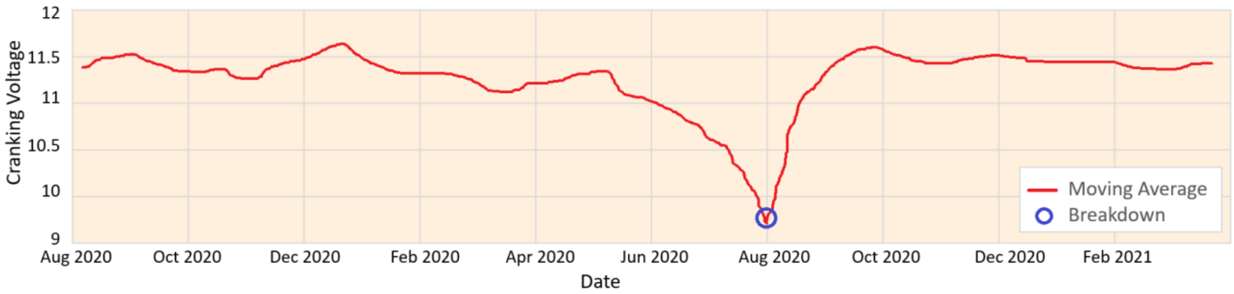


*Figure 3.2 Battery breakdown example*

Major inconsistencies were found in the dataset which required a lot of filtrations to ensure the model learned from clean and accurate training data. The presence of redundant patterns may have resulted in misleading the model during the learning and classification processes. Clean data helps guide the model to learn from informative patterns that describe the descending behavior of the battery life before it fails. The inconsistencies included the following:

- Insufficient data.
- Time gaps.
- Incorrect breakdown information.
- Multiple GO devices per VIN.
- Invalid VINs.
- Invalid cranking events.

These issues are explained in detail below:

### *3.2.1 Insufficient Data*

The number of data points of the raw cranking voltage signals varied significantly in different training samples. The size of the input layer of the ML model is typically equal to the sample with the highest number of data points. When samples with less data points are fitted to the input layer, the rest of units will be filled up with zeros or Nans. This would create an undesired flat pattern that might confuse the model. The ML model might take this pattern into consideration while looking for common patterns within the same class, which results in misclassification. One possible solution to this issue was to fill out all the missing data with the average value of cranking voltages. The maximum and minimum cranking values were also a better alternative to zeros and

Nans. This only fixed the issue for signals with an acceptable number of data points. Since enough samples were available for training, signals with less than 500 data points, as shown in Figure 3.3 were completely discarded.

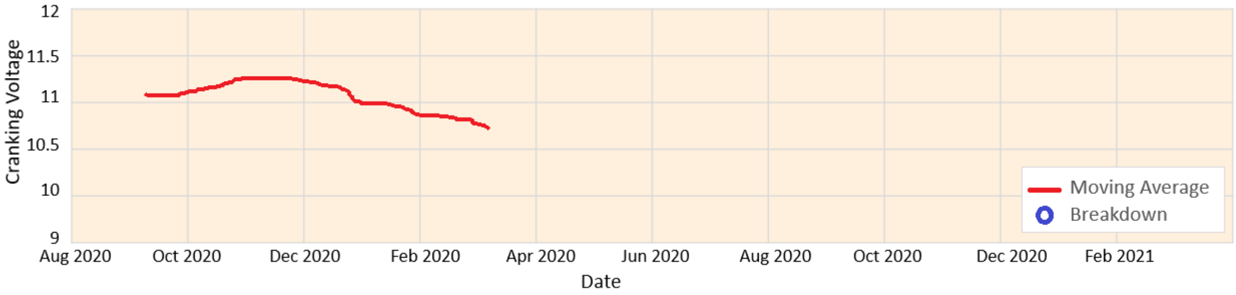


***Figure 3.3 Insufficient battery data***

### *3.2.2 Time Gaps*

Some outliers with time gaps were also found. These gaps were mostly a result of the vehicles staying in the workshop for months either waiting for new batteries or other spare parts or simply belonged to vehicles with no duty during this time. Bad installations of the GO device or miscommunications between the vehicle's Electronic Control Unit (ECU), Controller Area Network (CAN) bus and the GO device could also cause these gaps in time. Another reason why this pattern could show up was vehicles that had been running but plugged off from the GO device for a while

Although the reasons behind the time gaps were different than the missing data issue, the model dealt with both issues the same way. Although the gaps were also filled up with zeros or Nans, the solution to this issue was different. The time stamp from the classification process was dismissed since it was not needed as a feature for this specific purpose. However, timestamps would have been more important if it were a regression problem and time of failure prediction was required.

Instead, the last data point prior to the time gap was followed immediately by the first data point after the gap. This way, time gaps were avoided, as in Figure 3.4.

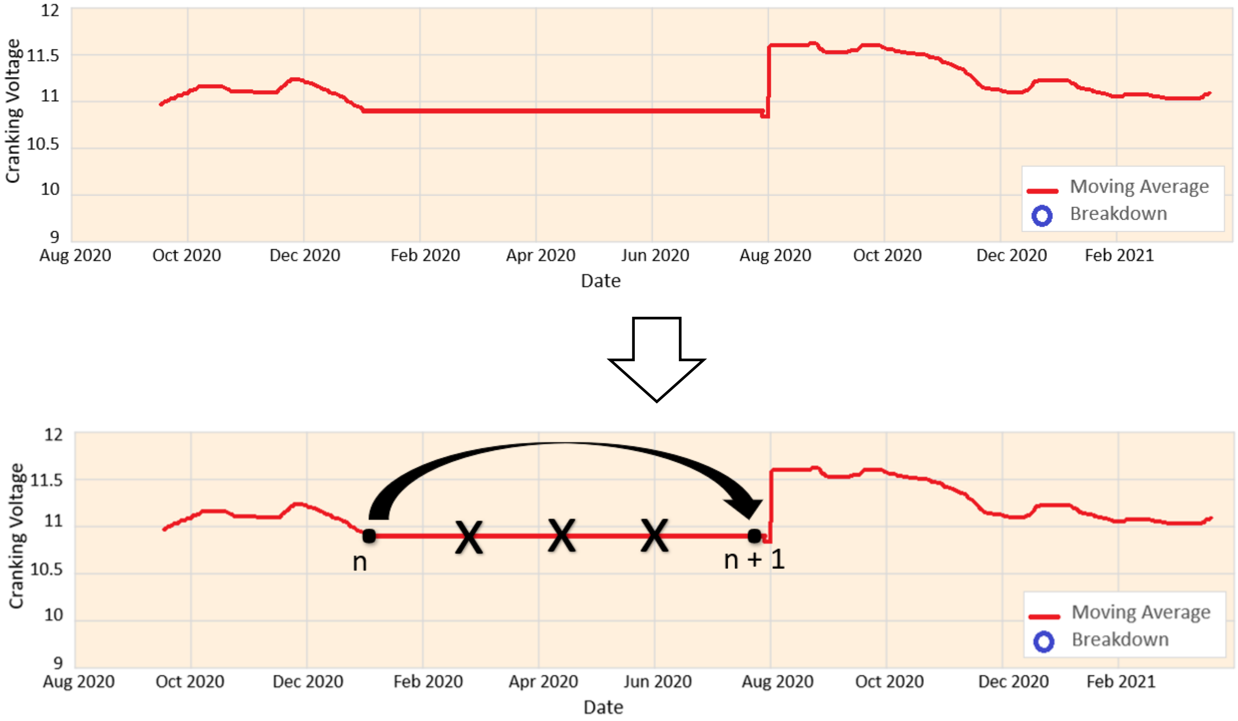


*Figure 3.4 Time gaps within battery data*

### *3.2.3 Incorrect Breakdown Information*

The repair logs (i.e., technical reports) were not very accurate since they were manually entered by technicians. Some batteries were reported as having broken down while their battery signals did not show any breakdowns as in Figure 3.5. Those batteries were only replaced as a periodic maintenance procedure. Based on the valid healthy and breakdown samples, a ML model was built specifically to detect battery signals with actual breakdowns. In other words, the ML model was used to label samples with incorrect breakdown information.

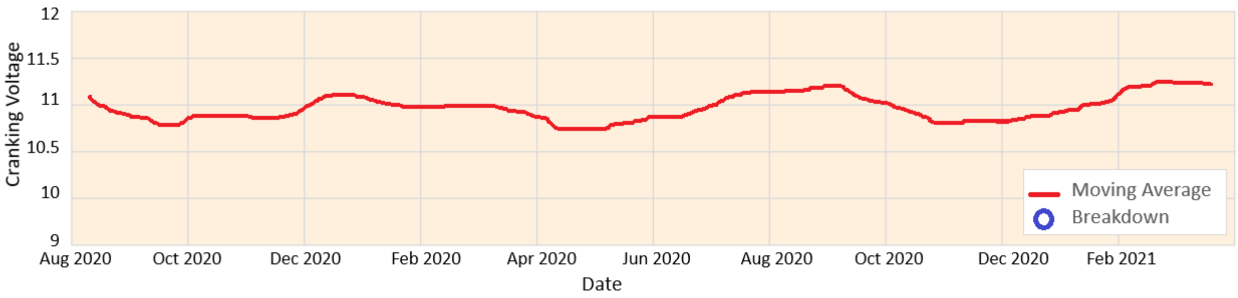


*Figure 3.5 False battery breakdown*

Some repair logs reported one breakdown only for vehicles that had multiple breakdowns. This might have resulted in including data points that do not describe actual breakdown events. Some were manually fixed while others were totally discarded.

### *3.2.4 Multiple GO Devices Per VIN*

Some vehicles were connected to multiple GO devices while cranking voltages were collected. Some re-sellers were doing this in their own vehicles to test new devices. Also, some vehicles had multiple devices connected at the same time because some re-sellers used dual adaptors to test and compare device performances. Solving this matter was not easy and was time consuming. It was necessary to create new lists, as shown in Table 3.1, for Go devices which were pugged in each VIN. At this point, VINs had to be the only identification information in all previous datasets and since then, GO devices were not used anymore for identification. Thus, any records where the GO device was plugged in for less than a week were discarded.

*Table 3.1 VIN vs GO device*

| Vin | GO Device | Plugged in | Plugged out |
|---|---|---|---|
| VIN1 | Device1 | 2016-07-14 03:15:000 UTC | 2016-10-01 22:05:000 UTC |
| VIN1 | Device2 | 2016-10-01 22:10:000 UTC | 2017-02-17 12:30:000 UTC |
| VIN1 | Device3 | 2017-03-17 12:30:000 UTC | 2018-01-03 12:47:000 UTC |
| VIN2 | Device1 | 2015-07-14 07:08:000 UTC | 2016-10-01 20:50:000 UTC |

### 3.2.5 Invalid VINs

Since VINs became the only vehicle identifier, double checking the validity of VINs was crucial. VINs were logged through the GO device using VIN decoders. Like any firmware code, it is common to find bugs that mainly happened because of incorrect device installations or bad communications between the ECU, CAN bus and the GO device. There was no way to re-generate the correct VINs, but fortunately, invalid VINs were less 3% of the dataset. Thus, all vehicles with invalid VINs were discarded from the training dataset.

### 3.2.6 Invalid Cranking Events

This was one of the biggest issues which could not be ignored. Many cranking events that were logged through the GO device were invalid. The curve below shows a typical valid cranking event. The crank starts with a significant dip when the ignition turns on, which represents the highest load on the battery. The cranking voltage (see Figure 3.6) increases gradually as the engine is starting and until it reaches a steady state. However, a noticeable number of cranking events of the vehicles in the training set did not follow this behavior.

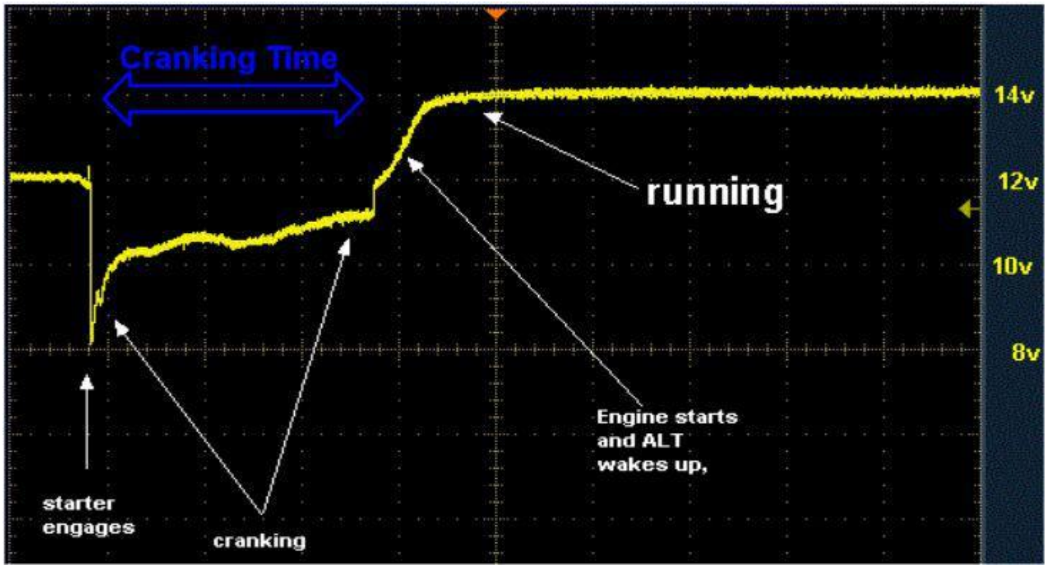


*Figure (3.6) Battery cranking*

Multiple sources might have been the cause of this issue:

(a) Bad device installation/communication.

(b) GO device was not compatible with certain makes and models.

(c) Fuel Type (e.g., Hybrid and electric vehicles reported 100% invalid cranking events).

(d) Battery Type (e.g., Some vehicles had 24v batteries, other had dual 12v batteries).

(e) A few warm starts did not make complete cranks.

(f) Software bug (i.e., the device logs cranking data, but the data shows that ignition was off).

No consistent correlations were found with any of the possible reasons mentioned above. Thus, a quick ML model was built to classify invalid cranking events. Random forest was the best candidate for this task since a typical cranking curve has a clear pattern. Instead of raw data, a special set of features, namely Curve Description (CD) features were extracted as the input data to the ML model. Like any supervised ML model, labels were required. Thus, a special visualization tool was used to label thousands of cranking events manually. The tool allowed users to select cranking events and the corresponding associated class (0: Valid, 1: Invalid). More than 2000 cranking events were manually labeled using the visualization tool within a couple of days. The Curve Description (CD) features are discussed in chapter 7.

## *3.3 Training Dataset*

After solving all the issues listed in section 3.1, the following dataset was used for training and testing. The dataset was split into 70% for training and 30% for testing as shown in Table 3.2.

***Table 3.2 Battery dataset***

| | **Training set** | **Testing set** | **Total** |
|---|---|---|---|
| **Healthy samples** | **366** | **157** | **523** |
| **Breakdown samples** | **366** | **157** | **523** |

A total of 523 labeled battery breakdown samples were collected, where 366 samples were used for training (i.e., 70%) and 157 samples were used for testing (i.e., 30%). Similarly, 523 healthy battery samples were available. The dataset of the healthy batteries consisted of 366 training samples while 157 were reserved for testing. Much more battery samples were available, but they were unlabeled.

## *3.4 Data Processing*

The final training datasets in Google Big Query (GBQ) were then exported to Google Cloud Storage (GCS), which is an online storage application on Google Cloud Platform (GCP) where the data were compressed to single (or multiple) downloadable and format-free files. The data files were then reshaped and sorted to a ML dataset format and labeled under 2 classes

(Healthy/Breakdown). The entire labeled dataset was imported to Google Cloud Data lab (GCD), which is a Python-based live notebook, and was ready to feed the training algorithm. Figure 3.7 shows the different stages of data processing. The structure and results of the Machine Learning model are discussed in later chapters.

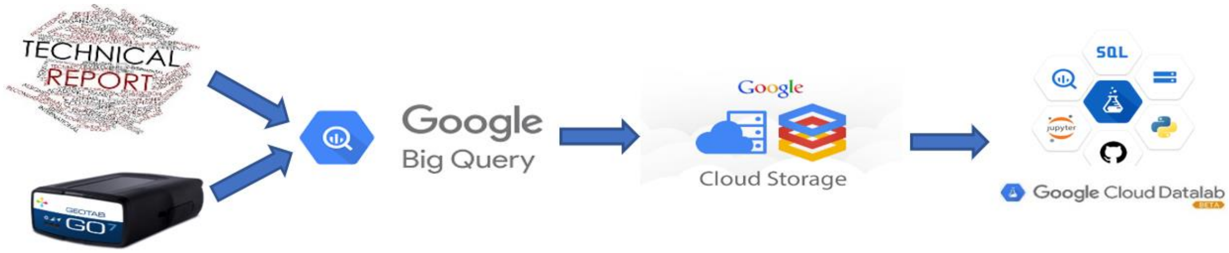


*Fig (3.7) Data Processing from source to analysis*

# Chapter 4

# Battery Degradation Detection

# Results and Discussions

# *Chapter 4 – Battery Degradation Detection Results and Discussions*

The objective of the battery degradation detection project was to detect progression of faults in 12v accessory batteries using historical data. The data collection process was described in detail in section 3.1. In this chapter, the supervised Machine Learning (ML) algorithms which were developed to detect battery degradation are discussed, along with the results. The output from these models is one of two classes: healthy battery or failing battery.

To learn from labeled data, the model accommodated historical data from both classes and their corresponding labels. The classification process is basically divided into two main stages: Training and testing. Training the Machine Learning model requires enough data to get a high detection rate. Enough data does not only mean enough samples, but also relevant and rich features in each sample.

The correlation of the input features with the target feature (e.g., cranking voltage) highly affects the accuracy of the model. The input features are transformed from high-level to low-level features in the form of vectors. Each sample is propagated in parallel with its corresponding label to give full information about each class and its characteristics. The testing process starts then with a prognostic model that diagnoses test samples and predicts the class they belong to. The output of this model is the predicted label of the test sample. Figure 4.1 shows the structure of the supervised battery degradation model which consists of two main stages: training and testing. The training stage starts with loading both healthy and breakdown training data along with their corresponding labels.

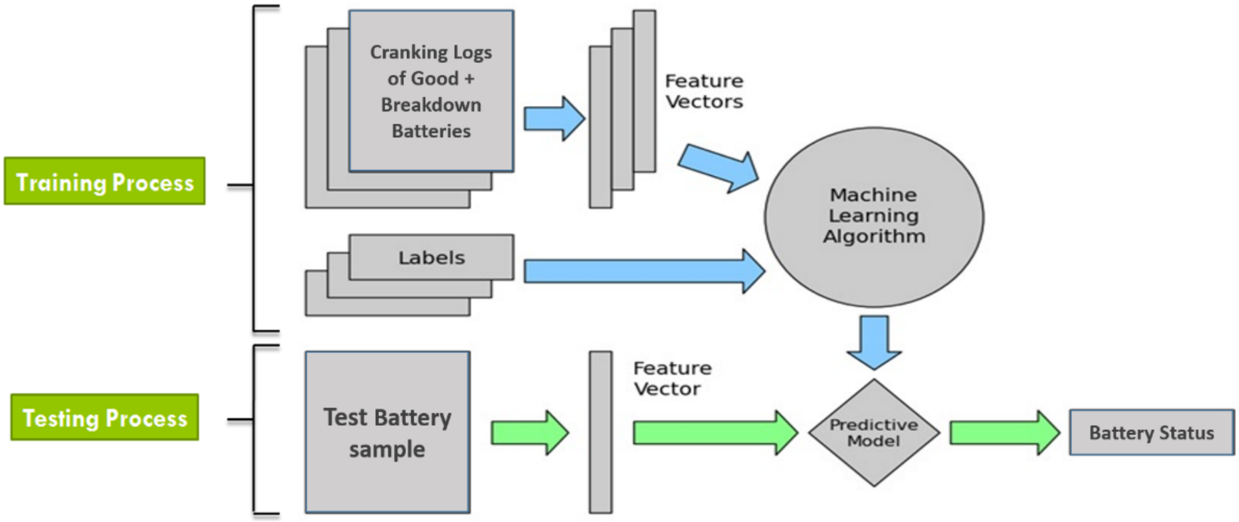


*Figure 4.1 Supervised battery degradation detection model*

Since raw battery signals are noisy, moving average was applied to smoothen the curve and emphasize patterns in the curve. The resultant moving averaged signals became the new input features. Both features and labels were then fed to the Machine Learning (ML) algorithm, which is an Artificial Neural Network (ANN). The next stage was to load test battery samples, take their moving averages and predict the label. The label was predicted by the ANN which provided probabilities for each of the 2 classes. The class which got the highest probability became the final label for each test sample.

As discussed earlier in section 3.1, thousands of battery data samples were collected and stored by Geotab through their own GO telematics device. The collected dataset consisted of cranking voltage history of Geotab customer vehicles which contained both healthy batteries, and battery breakdowns (see breakdown example in section 3.1). However, only 2 of Geotab customers provided labels for the collected data due to data confidentiality, which only covered a few hundreds of data samples (see section 3.2). Therefore, more labels were required. Two options were available to label more data:

- Manually, by looking at battery history signals and find significant voltage dips.
- Develop a ML model to detect previous breakdowns in the battery history signal.

Labeling thousands of battery samples would have been very time consuming. Therefore, an ANN-based classification model (a.k.a. Model B1, see section 1.5) was built to label more data samples. The model detects previous battery breakdowns/replacements in battery history signals. This helped create a bigger labeled dataset for the next model. Model B2 (see section 1.5) detects degrading battery signals which indicate upcoming failures. Therefore, two ML models were built to achieve battery failure detection:

1. Model B1: assigns labels to unlabeled battery data (0: no previous breakdown detected, 1: previous breakdown(s) detected).
2. Model B2: detects degrading battery signals which indicate upcoming failures.

For comparison purposes, two classifiers were developed for each of the two models to achieve the highest possible performance:

a) Random Forest (RF),
b) and an Artificial Neural Network (ANN).

Figure 4.2 shows the hierarchy of the supervised ML models. Model B1 was developed to detect previous battery breakdowns using both Artificial Neural Network (Model B1(a)) and Random Forest (Model B1(b)). Model B2 was developed to detect upcoming battery failures using both Artificial Neural Network (Model B2(a)) and Random Forest (Model B2(b)) as well.

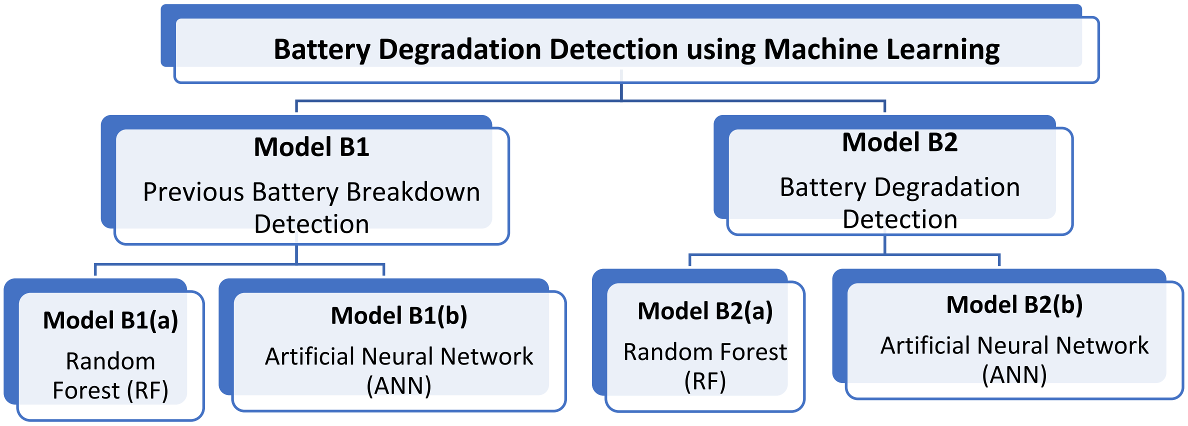

Battery Degradation Detection using Machine Learning
Model B1
Previous Battery Breakdown Detection
Model B2
Battery Degradation Detection
Model B1(a)
Random Forest (RF)
Model B1(b)
Artificial Neural Network (ANN)
Model B2(a)
Random Forest (RF)
Model B2(b)
Artificial Neural Network (ANN)

*Figure 4.2 Battery ML models*

## 4.1 Model B1(a): Battery Breakdown Detection Using ANN

Model B1(a) detects previous battery breakdowns/replacements in the battery history signal using ANN. In this model, a common distribution strategy in ML was followed, where the dataset splits up into 70% for training and 30% for testing. The labels of the testing samples were never used during the testing process to make sure the algorithm was not testing samples that it has been already used for training. The supervised learning model comes up with a label (i.e., 0: no breakdown detected, 1: breakdown(s) detected) as an output. This label is predicted based on probability computations. The label having the highest probability is the one the test sample is assigned to. To calculate the detection rate, the predicted labels are compared with the actual labels of the test samples. The detection rate is the number of right predictions the algorithm can make. For example, if the model was able to make 90 right predictions out of 100 test samples, the detection rate is 90%.

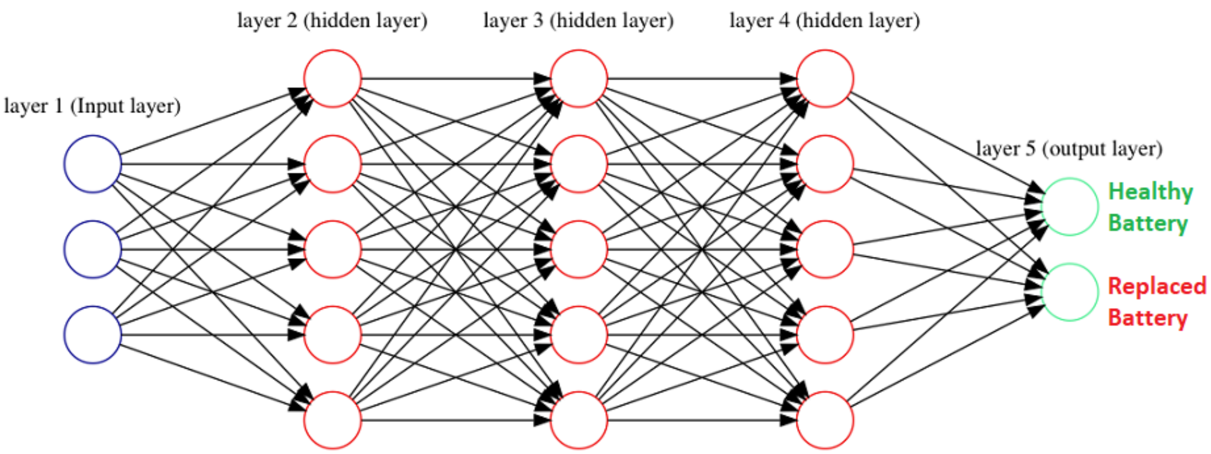


*Figure 4.3 ML detection of battery replacement*

Given privacy restriction associated with Geotab's data policy as explained in Chapter 3, Figure 4.4 is a fictitious but realistic example of a typical on-duty battery breakdown. The red line shows the moving average of the minimum cranking events while the blue circle shows the time of breakdown and replacement. All the cranking records of each sample were fully used for training. This included the cranking logs before and after the breakdown event, as well as the breakdown event itself. The classification in this case was mainly based on the curve dip which represents a battery breakdown or replacement event.

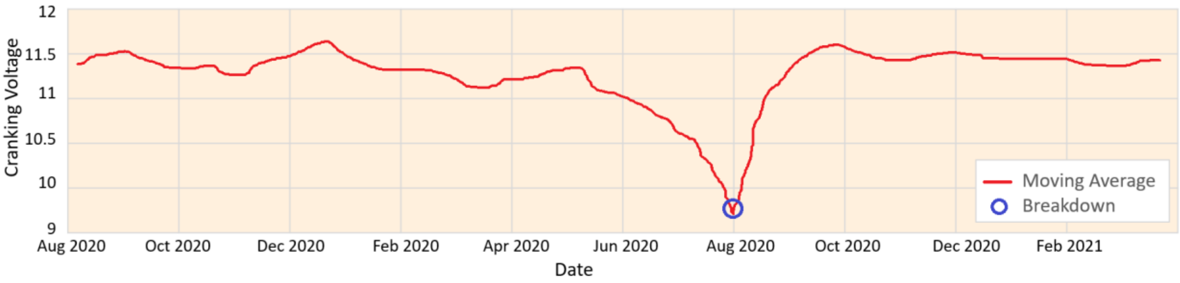
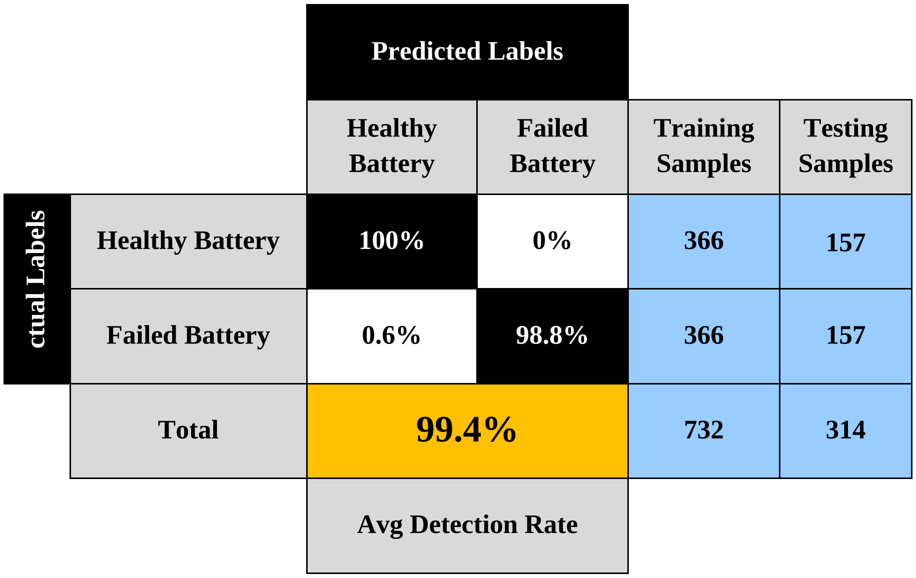


*Figure 4.4 Battery breakdown example*

Figure 4.5 shows the confusion matrix for Model B1(a). A confusion matrix is a common way to describe the performance of a classification model. A total of 523 breakdown samples were available from Customer1. The number of labeled training samples was 366 (i.e., 70%) while 157 (i.e., 30%) samples were saved for testing. A similar number of healthy samples was available for training and testing. The training set of the healthy batteries consisted of 366 samples while 157 were reserved for testing. An average battery replacement detection rate of 99.4% was successfully achieved using an Artificial Neural Network (ANN).



| | | Predicted Labels | | | |
|---|---|---|---|---|---|
| | | Healthy Battery | Failed Battery | Training Samples | Testing Samples |
| ctual Labels | Healthy Battery | 100% | 0% | 366 | 157 |
| | Failed Battery | 0.6% | 98.8% | 366 | 157 |
| | Total | 99.4% | | 732 | 314 |
| | | Avg Detection Rate | | | |

*Figure 4.5 Confusion matrix of battery breakdown detection using ANN*

## *4.2 Model B1(b): Battery Breakdown Detection Using RF*

For comparison purposes, Model B1(b) was developed for the same objective as Model B1(a). It detects previous battery breakdowns or replacements in the battery history signal but using Random Forest (RF). RF works best when it is trained on statistical features, rather than raw input data. That is why RF has not been used often in time-series applications. In this research, a novel pre-processing method is proposed which enables time-series classification using Random Forest. The method suggests a set of features that describe the shape of battery curve, referred to as Curve Description (CD) features.

The input features were single individual points which described the battery history curve. The following thirty features were carefully selected to replace raw time-series data as input to RF:

- *First Voltage*
- *Last Voltage*
- *Min Voltage*
- *Max Voltage*
- *Mean Voltage*
- *First Voltage - Min Voltage*
- *Avg Voltage - Min Voltage*
- *Max Voltage - Min Voltage*
- *Difference between First and Last Voltage in number of events*
- *Difference between First and Last Voltage in Months*
- *Difference between First and Min Voltage in number of events*
- *Difference between First and Min Voltage in Months*
- *Difference between Max and Min Voltage in number of events*
- *Difference between Max and Min Voltage in Months*
- *Start Type (Cold/Warm)*
- *Standard Deviation of the battery history signal (see Figure 4.6)*
- *2 * Standard Deviation of the battery history signal*
- *3 * Standard Deviation of the battery history signal*

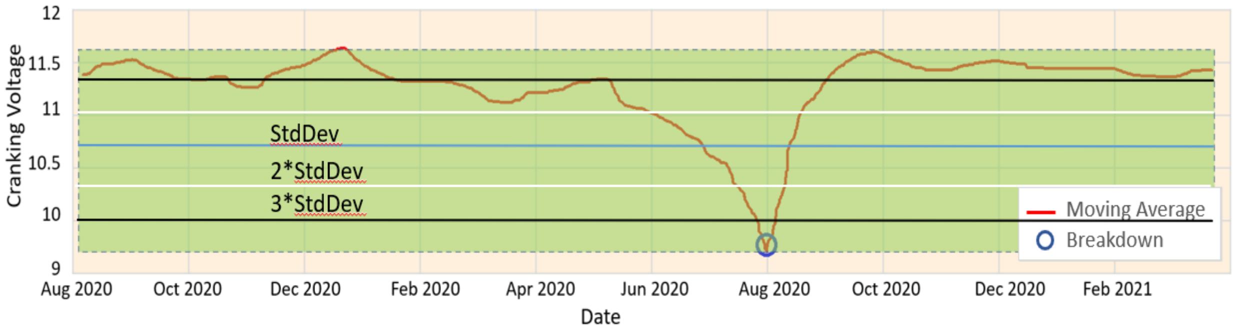


***Figure 4.6 Standard deviation of battery history signal***

- *Voltage at 25% of the battery history signal (see Figure 4.7)*
- *Voltage at 50% of the battery history signal*
- *Voltage at 75% of the battery history signal*

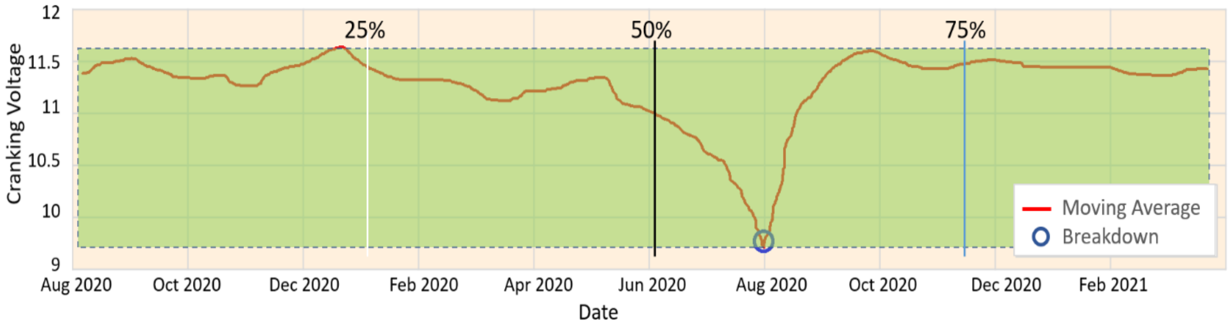


***Figure 4.7 Voltages at quarter portions of battery signal***

Feeding the Random Forest model with curve-shape description features resulted in a detection rate of 98.6%. This means that in every 100 battery history signals, the RF model could accurately classify 98 previous breakdowns correctly. Generating the right features was key to success of this model. Figure 4.7 shows the confusion matrix of of the Battery Breakdown Detection model using the novel Curve Description (CD) pre-processing method, along with Random Forest, referred to as CD-RF.

| | | Predicted Labels | | | |
|---|---|---|---|---|---|
| | | Healthy Battery | Replaced Battery | Training Samples | Testing Samples |
| Actual Labels | Healthy Battery | 99.8% | 0.2% | 1800 | 600 |
| | Replaced Battery | 2.6% | 97.4% | 1800 | 600 |
| | Total | 98.6% | | 3600 | 1200 |
| | | Avg Detection Rate | | | |

*Figure 4.8 Confusion matrix of battery breakdown detection using CD-RF*

## 4.3 Model B2(a): Battery Degradation Detection Using ANN

The main goal of this model is to predict upcoming battery failures by detecting degrading battery curves (i.e., progression of faults). Thus, the model should be trained on an early portion of the battery curve prior to the breakdown event. For example, in Figure 4.9, the breakdown event (e.g., blue circle) should not be included in the portion used for training. Here, only an earlier portion (e.g., the green area) was used for training.

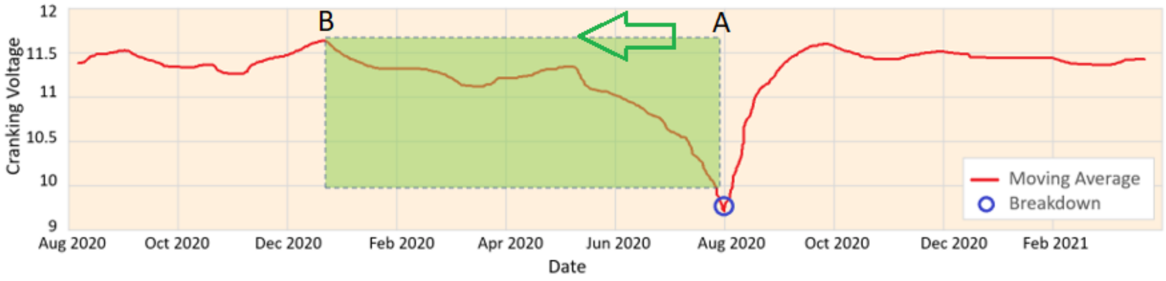
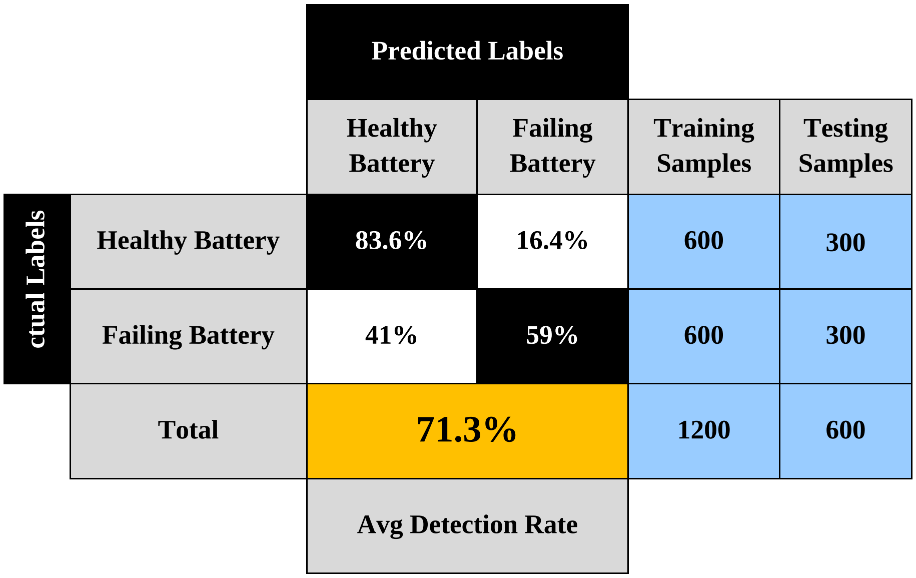


*Figure 4.9 Training portion: 49 cranking events prior to breakdown, to 630 events backwards*

Generalizing a rule on how to extract this portion from all battery signals was challenging since different vehicles have been used very differently based on the daily routine of the driver. For example, local food delivery vehicles will make many cranking events throughout the day but not a lot of mileage, while commercial long-hauls make more mileage but conduct much fewer cranking events per day. Employee buses have fixed daily routes and will run almost the same mileage every day but will only make a couple of cranking events for the day. This means that different vehicles make similar number of cranking events in very different periods of time. Therefore, time (e.g., number of days/hours prior to the breakdown event) was not used as a threshold to decide on what portion of the battery signal will be used for training. Instead, the number of cranking events was selected as the threshold. After many fine-tuning experiments, the threshold was set to 49 cranking events prior to breakdown (i.e., starts at point A in Figure 4.9). The training portion covers 630 cranking events backwards (i.e., ends at point B in Figure 4.9). The 49 cranking events before the breakdown represented an average of one week worth of data for an average of 7 Cranking events per day, every day. The 630 prior cranking events represented an average of 3 months of cranks, with the same rate.

The development strategy for Model B2 followed the same distribution strategy of 70% for training and 30% for testing. The labels of the testing samples were still hidden during the testing. The supervised learning model predicts a label (0: Healthy Battery, 1: Failing Battery). Each label gets a probability, where the sum of probabilities must be equal to 100%. The label having the highest probability was the one the test sample was assigned to. To calculate the detection rate, the predicted labels were compared to the actual labels of the test samples. Figure 4.10 shows the test confusion matrix for of Model B2(a) using ANN. The model was only able to detect 71.3% of the provided test samples.



| | | Predicted Labels | | | |
|---|---|---|---|---|---|
| | | Healthy Battery | Failing Battery | Training Samples | Testing Samples |
| ctual Labels | Healthy Battery | 83.6% | 16.4% | 600 | 300 |
| | Failing Battery | 41% | 59% | 600 | 300 |
| | Total | 71.3% | | 1200 | 600 |
| | | Avg Detection Rate | | | |

*Figure 4.10 Confusion matrix of battery degradation detection using ANN*

## 4.4 Model B2(b): Detection of Battery Degradation Using Random Forest

The first step in Model B2(b) was to implement the novel pre-processing Curve Description (CD) method. A set of 30 features which describe the curve shape were extracted. These features were then fed into a Random Forest (RF) classifier to detect battery degradation. Same as in Model B2(a), the signal portion used for training had to start and end before the actual breakdown. The training portion started at 49 cranking events prior to the breakdown and went backwards to 630 cranking events prior to the breakdown as shown in Figure 4.10. The following features were extracted from the training portion to be used as input data for the Model B2(b):

- *First Voltage*
- *Last Voltage*

- *Min Voltage*
- *Max Voltage*
- *Mean Voltage*
- *First Voltage - Min Voltage*
- *Avg Voltage - Min Voltage*
- *Max Voltage - Min Voltage*
- *Difference between First and Min Voltage in number of events*
- *Difference between Max and Min Voltage in number of events*
- *Start Type (Cold/Warm)*
- *Standard Deviation of the training portion (e.g., green area), see Figure 4.11.*
- *2 * Standard Deviation of the training portion.*
- *3 * Standard Deviation of the training portion.*

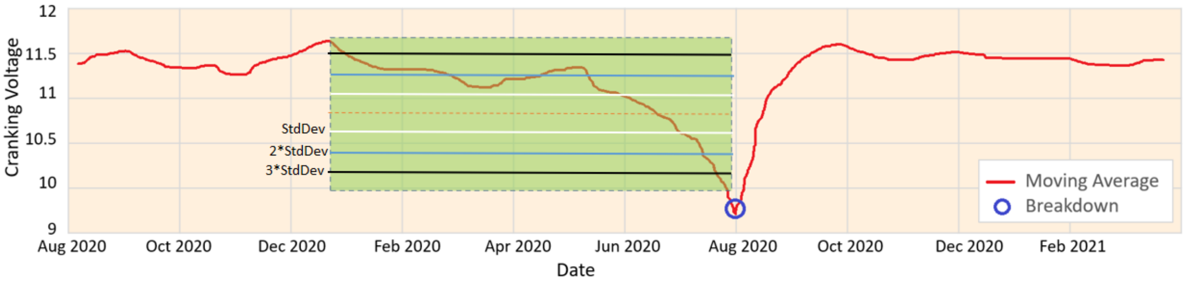


***Figure 4.11 Standard deviations of battery history signals***

- *Voltage at 25% of the training portion (e.g., green area), see Figure 4.12.*
- *Voltage at 50% of the training portion.*
- *Voltage at 75% of the training portion.*

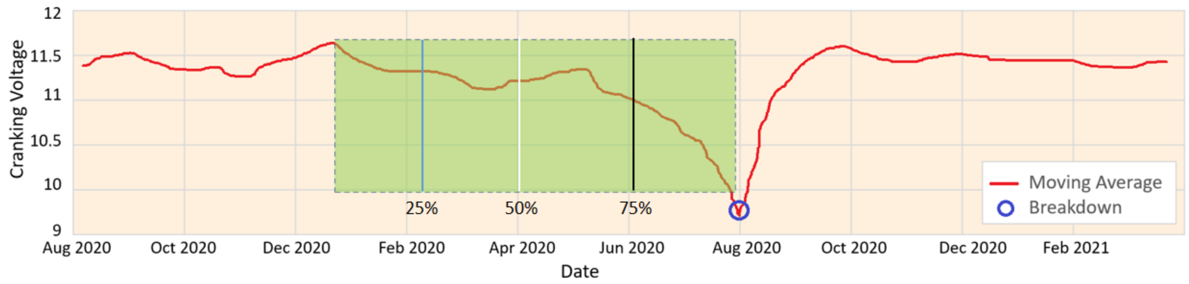
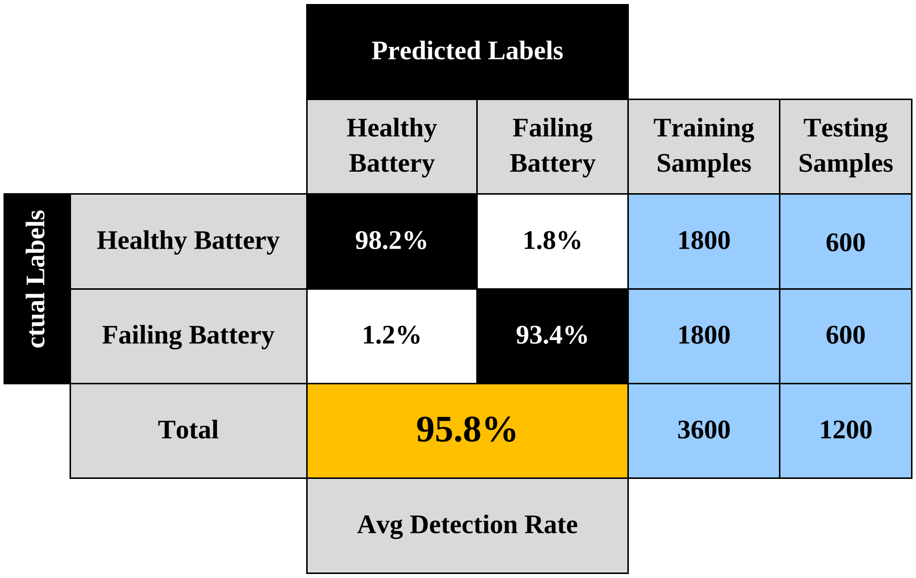


*Figure 4.12 Voltage checkpoints of battery signals*

The novel Curve Description (CD) features along with RF method outperformed the Artificial Neural Network (ANN) in Model B2(b). Generating the right features for different models was key to success of this model. The model was able to detect battery degradation in 95.8% of the test samples. Figure 4.13 shows the confusion matrix of Model B2(b).



| | | Predicted Labels | | | |
|---|---|---|---|---|---|
| | | Healthy Battery | Failing Battery | Training Samples | Testing Samples |
| ctual Labels | Healthy Battery | 98.2% | 1.8% | 1800 | 600 |
| | Failing Battery | 1.2% | 93.4% | 1800 | 600 |
| | Total | 95.8% | | 3600 | 1200 |
| | | Avg Detection Rate | | | |

*Figure 4.13 Confusion matrix of battery degradation detection using CD-RF*

Table 4.1 shows a summary of the results of the four battery fault detection sub-models. Both the Artificial Neural Network (ANN) and Curve Description Random Forest (CD-RF) models have shown great performances in Model B1(a) and 1(b), respectively. However, Model B1(a) achieved a higher detection rate in detecting previous battery breakdowns.

In the battery degradation detection problem, Curve Description Random Forest (CD-RF) model has outperformed the ANN model by achieving 95.8% detection rate.

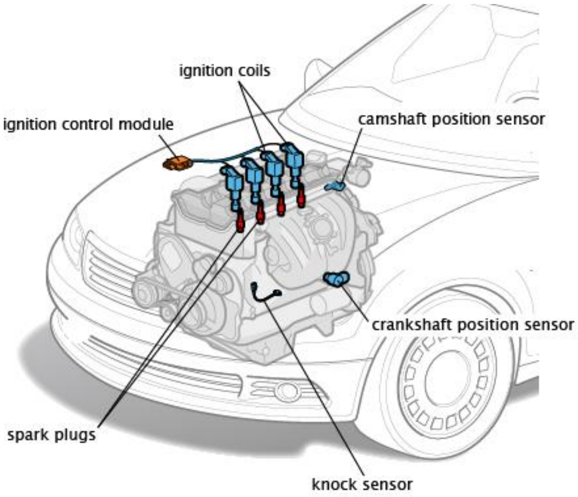
| | **MODEL** | |
|---|---|---|
| **Battery Breakdown Detection** | **Model B1 (a)**<br>[ANN] | **Model B1 (b)**<br>[CD-RF] |
| | 99.4% | 98.4% |
| **Battery Degradation Detection** | **Model B2 (a)**<br>[ANN] | **Model B2 (b)**<br>[CD-RF] |
| | 71.3% | **95.8%** |

***Table 4.1 Results of battery fault detection models***

# Chapter 5

# Engine Database and Experimental Setup

# Chapter 5 – Experimental Setup and Engine Database



All the experiments and engine data acquisition procedures were performed at McMaster's Center for Mechatronics and Hybrid Technology (CMHT) by its own test engineers. A rich and clean engine-test database was collected and was then structured and migrated to the cloud. This database can be very useful for AI researchers in the future. Not only that this kind of data is not available on the web but collecting a similar amount of data would be very costly and time consuming. The engine data was carefully measured and organized and was then migrated to a cloud database. The cloud database can be shared remotely with other researchers or institutions just by giving access to their email account. The database includes both healthy and faulty conditions of physically simulated spark plug and EGR valve faults. In this chapter, the experimental setup and the engine data collection process is discussed. The experimental setup presents all the equipment and tools as well as the operating conditions used to collect engine data. The data collection process describes the data acquisition devices and techniques as well as the sensors used for measurements.

## 5.1 Ford Engine

A 2018 Gen 3.0, 5.0L, V8, Ford Coyote engine was utilized during the data collection process. The cylinder head was modified to receive pressure transducer ports to measure the pressure inside the combustion chamber of selected cylinders. The general specifications of the engine are summarized in Table 5.1. A piezoelectric pressure transducer (Kistler type 6125C), with a range between 0 and 300 bar, was mounted a machined port on cylinder # 1 in order to measure its in-cylinder pressure.

***Table 5.1 Engine specifications***

| | |
|---|---|
| ***Engine Type*** | *2018 Ford Gen 3 Coyote Engine* |
| ***Valve-train*** | *DOHC, Direct Acting* |
| ***Block/head*** | *Aluminum/aluminum* |
| ***Bore x stroke*** | *3.661 x 3.649 in.* |
| ***Displacement*** | *5.038 Liter* |
| ***Compression ratio*** | *12:1* |
| ***Electronic fuel injection*** | *Sequential multiport* |
| ***Power (1)*** | *343 kW @ 6750 rpm* |
| ***Torque (1)*** | *569 Nm (420 lb.-ft) @ 4500 rpm* |
| ***Fuel delivery*** | *Direct Injection (DI) and Port Injection (PFI)* |
| ***Firing order*** | *1-3-7-2-6-5-4-8* |
| ***Engine weight (W/O accessories)*** | *453.0 lb. (205.5 kg)* |

An AVL type 2614CK optical crank angle encoder was mounted on the free end of the crankshaft on the damper pulley. This encoder had a resolution of 720 x 0.5°. The crank angle encoder was used to provide crank angle data in relation to cylinder # 1 Top Dead Center (TDC) that is essential for the calculation of crank angle-based results related to the combustion cycle (e.g., combustion duration, indicated mean effective pressures, mass fraction of fuel burned and maximum pressure magnitude and location). Figure 5.1 shows the Ford Coyote engine used for data collection.

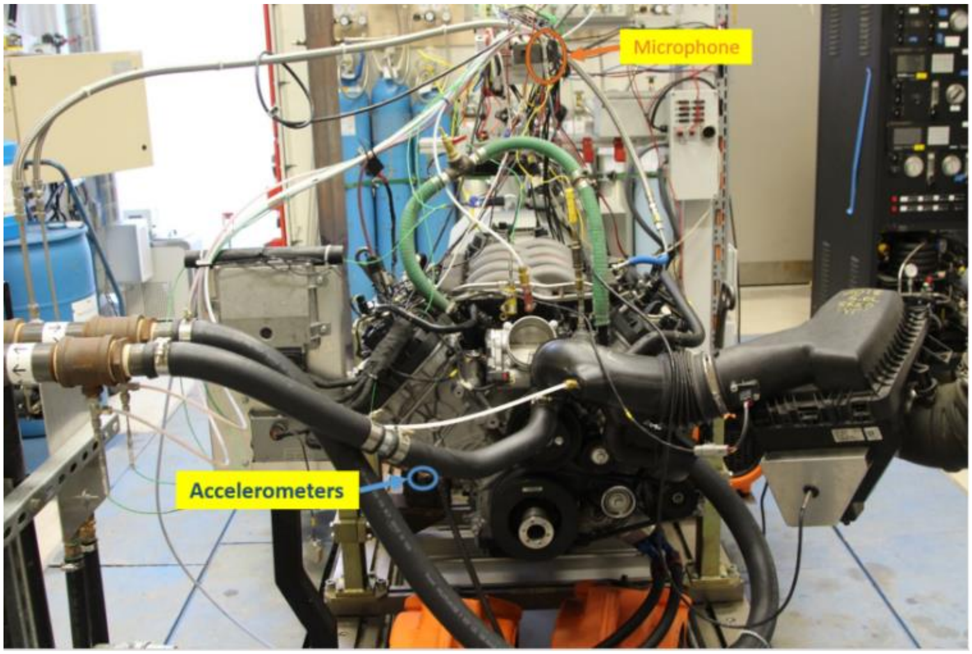


*Figure 5.1 Ford Coyote engine*

Looking from the front side of the engine, Cylinder #1 is in the left bottom corner of the engine. Moving upwards, Cylinder #2, cylinder #3, and cylinder #4 are listed in the same column all the way to the left top corner of the engine where Cylinder #4 is located. Cylinder #5 to cylinder #8 are listed the same way in the opposite side of the engine. Cylinder #5 marks the right bottom corner of the engine while cylinder #8 marks the right top corner. The 4 knock sensors are distributed over the 4 corners of the engine. Knock sensor #1 is located inwards next to cylinder #1, knock sensor #2 is next to cylinder #5, while knock sensors #3 and #4 are next to cylinders #4 and #8, respectively. The microphone is facing downwards in a perpendicular position on top of the engine. It is horizontally located right in the middle of the engine while it is vertically located in between cylinders #1 and #2. The 2 accelerometers are both located on the left bottom side of the engine. One is facing downwards in a perpendicular position to the engine, and the other is facing to the left. Figure 5.2 shows the locations of the knock sensors, accelerometers, and microphone.

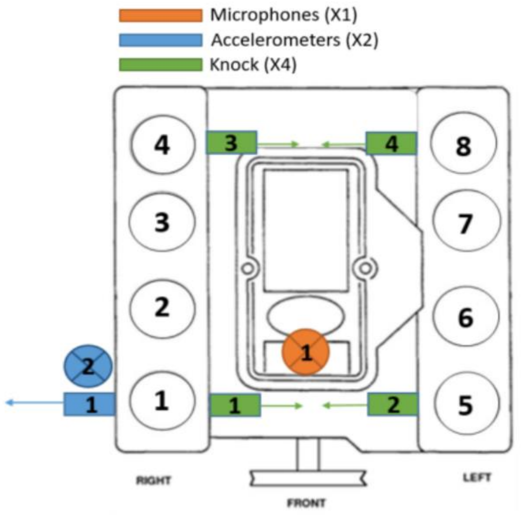


*Figure 5.2 Ford engine map*

An EFI Communication Tool (ECT) tool from EFI Technology Co. was used to access the engine Control Unit (ECU) and its calibration setup. This tool was used to control and record key ECU parameters such as spark timing, engine throttle position and air-to-fuel ratio (AFR). The engine and test cell were instrumented to measure key engine parameters. Several type K and T thermocouples were used to measure engine temperature at selected points. A list of the measured parameters is presented in Table 5.2. A T-250 HORIBA Titan AC induction dynamometer was used to run and load the engine. The maximum torque and speed of the dynamometer were 400 Nm and 8000 rpm, respectively. The Titan system was equipped with a SPARC engine test stand controller and STARS test automation system. These were used to control, monitor, and record the engine parameters listed in the previous table.

***Table 5.2 Engine and test cell instrumentation***

| | |
|---|---|
| • *Engine speed* | • *Engine torque* |
| • *Engine intake airflow* | • *Spark advance/ignition timing* |
| • *Fuel flow rate* | • *Barometric pressure* |
| • *Humidity* | • *Inlet air temperature* |
| • *Inlet air pressure* | • *Intake manifold air temperature* |
| • *Intake manifold air pressure* | • *Fuel rail temperature* |
| • *Ambient air temperature* | • *Engine coolant inlet temperature* |
| • *Engine coolant outlet temperature* | • *Test cell temperature* |
| • *Oil gallery pressure* | • *Oil temperature* |
| • *Exhaust gas temperature (Left bank)* | • *Exhaust gas temperature (Right bank)* |
| • *Fuel rail pressure* | • *Engine exhaust backpressure* |
| • *Engine intake restriction* | • *Throttle position* |
| • *Pedal position* | • *Battery Voltage* |

A Kistler KiBox to Go (Type 2893A) analysis system was used along with KiBox Cockpit Software to monitor and record engine measurements. These measurements included in-cylinder pressure, the output of four (4) engine's OEM knock sensors, two accelerometers and an optical crank angle encoder, from which the crank angle instantaneous velocity was calculated. The Kistler system was also used to perform combustion analysis based on the in-cylinder pressure data.

## 5.2 Spark Plug Data Collection

Machine Learning (ML) models are designed to find similarities in each class. Detecting spark plug degradation using ML or DL required data which represent the different spark plug fault conditions. This includes data of healthy as well as faulty spark plugs. As discussed earlier in the introduction, spark plug degradation may happen because of incorrect installation or simply because the normal spark plug life comes to an end. Deficiency in other engine parameters such as fuel injectors, air-to-fuel ratio (AFR), engine temperature and pressure can result in spark plug failure in the long term. One of the most common symptoms of spark plug degradation is the change in the size of the spark plug gap. Thus, spark plugs with different gap sizes had to be physically created to introduce the fault to the engine. Three spark plug samples with a standard gap, a smaller gap, and a larger gap, respectively, were used to artificially simulate aging. Figure 3.12 shows the spark plugs which were used in the CMHT lab. The spark plugs represent 3 different gap sizes which are:

- Gap Size 1 = 0.020”
- Gap Size 2 = 0.050” (Healthy)
- Gap Size 3 = 0.080”

The 3 different gap sizes represent 3 classes for the neural networks that were built to classify good and bad spark plugs. The 3 classes are presented in Figure 5.3 and Table 5.3 along with their corresponding gap sizes as follows:

| Class | Gap Size |
|---|---|
| **Fault 1** | 0.020” |
| **Healthy** | 0.050” |
| **Fault 2** | 0.080” |

***Table 5.3 Spark plug gap sizes***

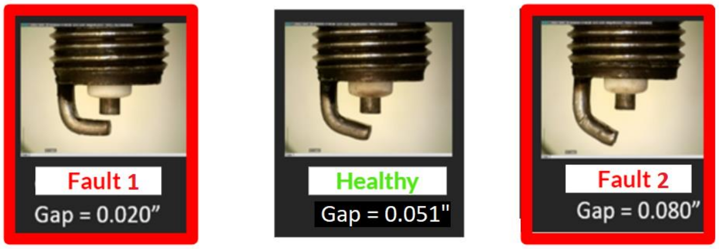


***Figure 5.3 physically simulated spark plugs***

Data collection occurred at 2 different operating conditions while spark plugs were placed in cylinder # 1, namely operating condition 1 and operating condition 2. In condition 1, a constant engine speed of 700 rpm and 15% engine load were applied, while in condition 2, a constant speed of 1000 rpm and 25% load were applied. The operating conditions are stated below in Table 5.4.

***Table 5.4 Engine operating conditions while collecting spark plug data***

| Condition | Engine Speed | Engine Load |
|---|---|---|
| **Operating Condition 1** | 700 rpm | 15% load |
| **Operating Condition 2** | 1000 rpm | 25% load |

The dynamometer was connected to a visualization software where the data being collected could be monitored. A total of 5 sensors were used to collect engine data while running with each of the 3 spark plug gaps and at each of the 2 operating conditions stated above. The following list includes the 10 sensors that were involved during the data collection process:

1. *Knock Sensor 1*
2. *Knock Sensor 2*
3. *Knock Sensor 3*

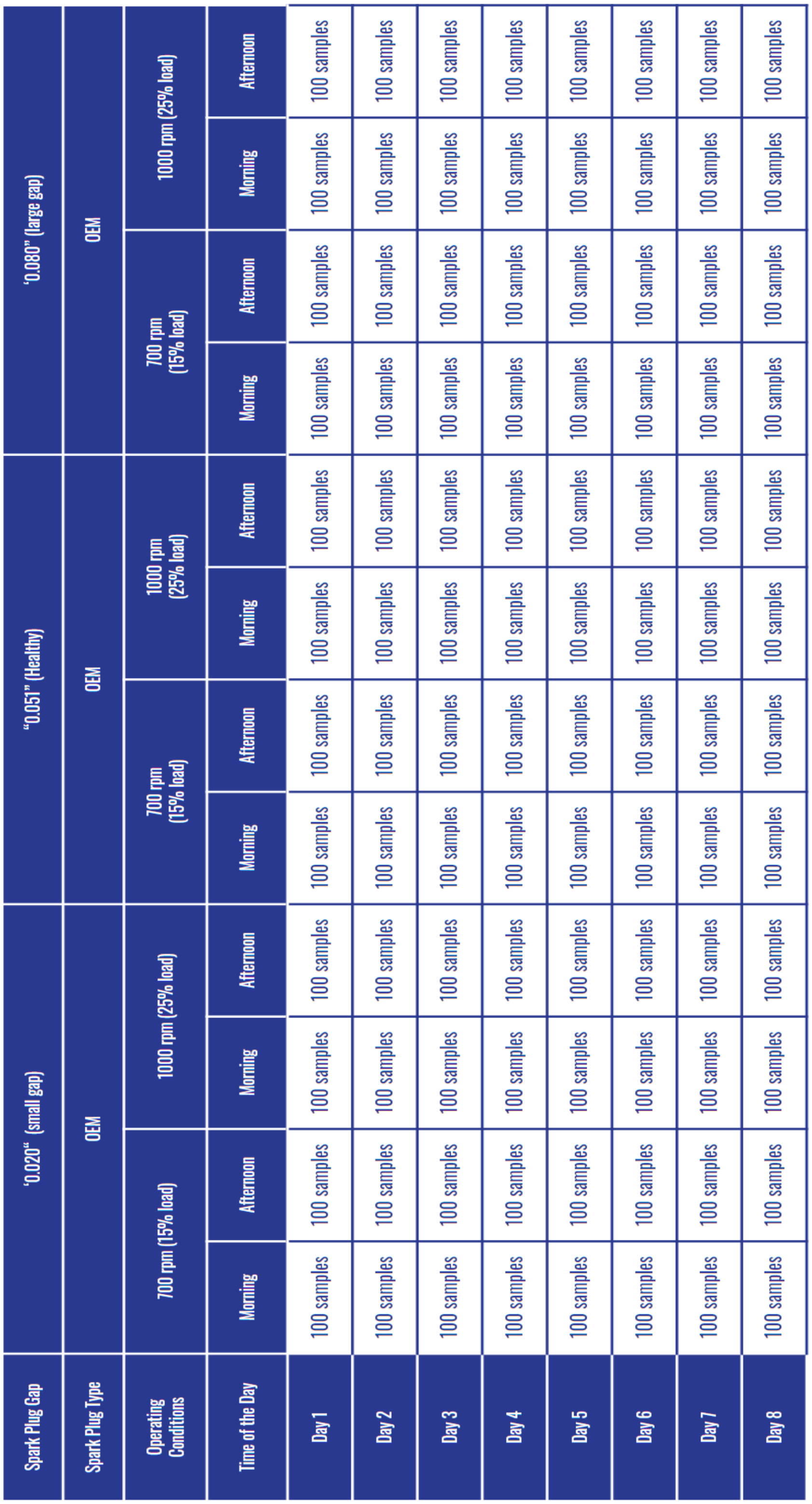
4. *Knock Sensor 4*
5. *Optical Encoder*

A total of 20 engine cycles per data sample were collected for each fault. Each engine cycle contains 144,720 data points per sensor. Having the 5 sensors listed above, the number of data points tallies up to more than 700,000 data points per sample. Thousands of samples were collected for training and testing which makes billions of data points. All these samples were collected in the morning time of the data collection days. An additional set of data with the exact same conditions was collected in the afternoon time of the same days. Not all sensors were used for training purposes but were collected to build a rich and reliable labeled engine database. The spark plug data were collected over eight separate days. Information about the dates and times when the data were collected are listed in Table 5.5. The table shows the distribution of the collected data over the 8 days and the different operating conditions.

***Table 5.5 Data collection dates and times***

| ***Index*** | ***Date*** | ***Morning time*** | ***Afternoon time*** |
|---|---|---|---|
| ***Day 1*** | March 10, 2020 | 9 am to 12 pm | 1 pm to 4 pm |
| ***Day 2*** | March 11, 2020 | 9 am to 12 pm | 1 pm to 4 pm |
| ***Day 3*** | June 22, 2020 | 9 am to 12 pm | 1 pm to 4 pm |
| ***Day 4*** | June 23, 2020 | 9 am to 12 pm | 1 pm to 4 pm |
| ***Day 5*** | June 24, 2020 | 9 am to 12 pm | 1 pm to 4 pm |
| ***Day 6*** | June 29, 2020 | 9 am to 12 pm | 1 pm to 4 pm |
| ***Day 7*** | June 30, 2020 | 9 am to 12 pm | 1 pm to 4 pm |
| ***Day 8*** | July 1st, 2020 | 9 am to 12 pm | 1 pm to 4 pm |

Table 5.6 shows the matrix of the engine database which contains spark plug fault conditions. Eight days were collected on the dates and times listed in Table 5.5. Each day consisted of 12 sets of data which covered the 3 fault conditions listed in Table 5.3 and the operating conditions listed in Table 5.4. Each set of data contained 100 training samples, where each training sample had 20 engine cycles. This added up to 9,600 training samples per sensor over the 8 days. The same number of samples was collected through 5 built-in sensors: 4 knock sensors and 1 optical encoder.



| Spark Plug Gap | '0.020" (small gap) | | | | "0.051" (Healthy) | | | | '0.080" (large gap) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Spark Plug Type | OEM | | | | OEM | | | | OEM | | | |
| Operating Conditions | 700 rpm (15% load) | | 1000 rpm (25% load) | | 700 rpm (15% load) | | 1000 rpm (25% load) | | 700 rpm (15% load) | | 1000 rpm (25% load) | |
| Time of the Day | Morning | Afternoon | Morning | Afternoon | Morning | Afternoon | Morning | Afternoon | Morning | Afternoon | Morning | Afternoon |
| Day 1 | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples |
| Day 2 | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples |
| Day 3 | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples |
| Day 4 | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples |
| Day 5 | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples |
| Day 6 | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples |
| Day 7 | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples |
| Day 8 | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples |

*Table 5.6 Spark plug data*

## 5.3 Spark Plug Data Variability

As a common practice in data science is to run sanity checks over the collected data to ensure its validity and have a sense of its variability. Validity indicates whether data samples of the similar tests follow the same pattern, while variability shows different scenarios that the data can be exposed to while following the same pattern. Variability can happen because of many reasons such as:

- change in temperature.
- change in humidity.
- change of equipment (e.g., sensor, battery, or cord replacement).
- noise; and
- experimental issues (e.g., loose sensor or cord).

Data samples of similar tests which follow different characterizations (i.e., anomalies) can be labeled as variant data. Variant data can have the following characterizations (i.e., symptoms):

- significantly lower sensor voltage amplitudes.
- lower frequencies.
- lower standard deviation; and
- lower variance.

In this research, Sanity checks in both crank and frequency domains were conducted to each sample of all the 8 days of data. Almost 60% of the collected data followed the same pattern for the same fault conditions, while the rest were relatively different. The days were classified in Table 5.7. Day 1, 2, 4 and 5 were labeled as set "A" which had very similar data, while day 3, 6, 7 and

8 were labeled as set "B" because they had variant characterizations. This section will explain all the analytical methods which were conducted to label the data into these 2 sets.

| ***Set A*** | ***Set B*** |
|---|---|
| *Day 1* | |
| *Day 2* | |
| | *Day 3* |
| *Day 4* | |
| *Day 5* | |
| | *Day 6* |
| | *Day 7* |
| | *Day 8* |

***Table 5.7 Regular (Set A) vs variant (Set B) data***

The first basic check was to plot the raw data of the knock sensors and visually search for abnormalities. A typical knock sensor signal that results from a healthy spark plug is plotted in Figure 5.4 which represents 201 engine cycles collected on day 1. The signal shows a normal behavior of a knock sensor # 1 with an average amplitude of 3V, which means that the sample is valid. All the data samples of day 1 were checked and no anomalies were observed. The same visualization check was conducted over all the 8 days. Figures 5.5 – 5.11 show the raw signals of knock sensor # 1 of day 2 to 8, respectively, against day 1:

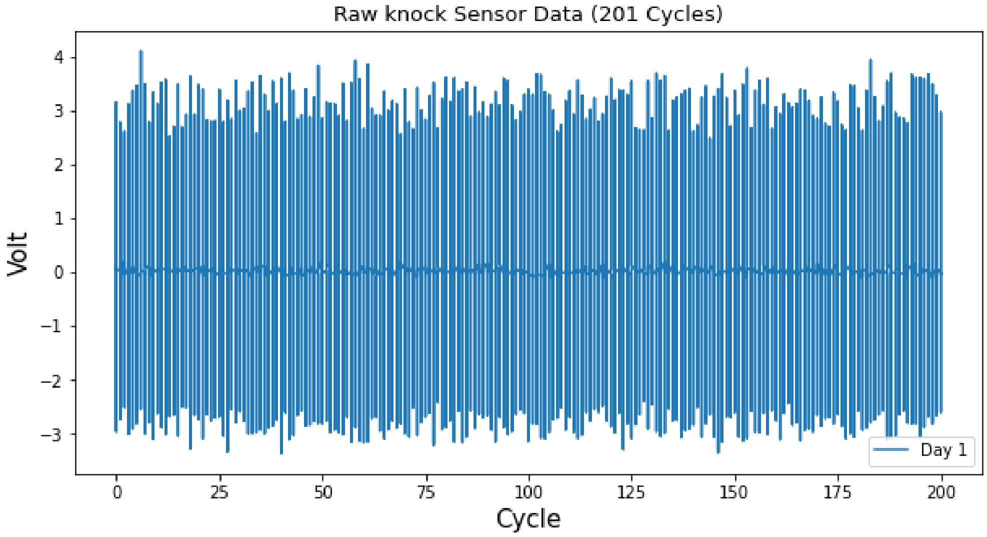


***Figure 5.4 Day 1 of raw spark plug data***

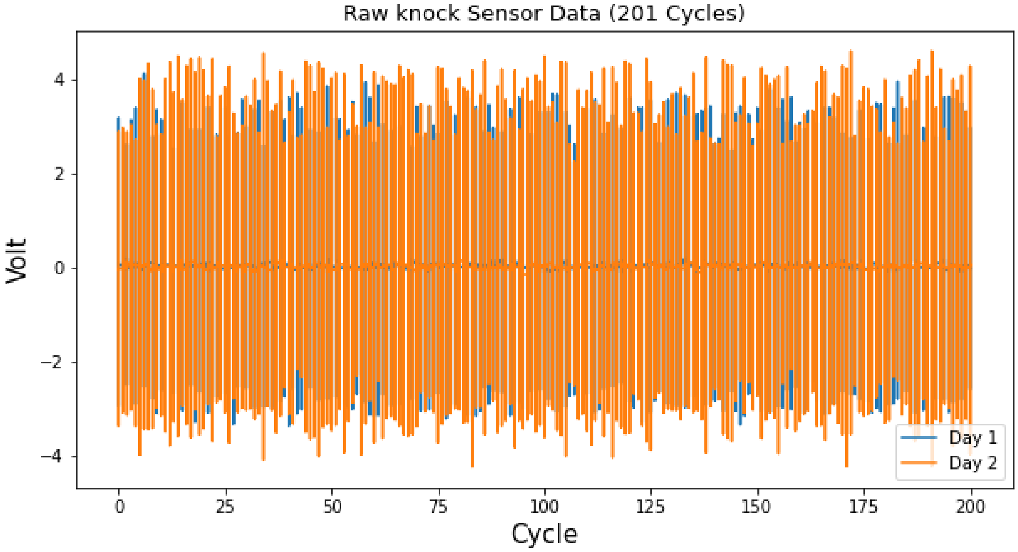


***Figure 5.5 Day 1 of raw spark plug data***

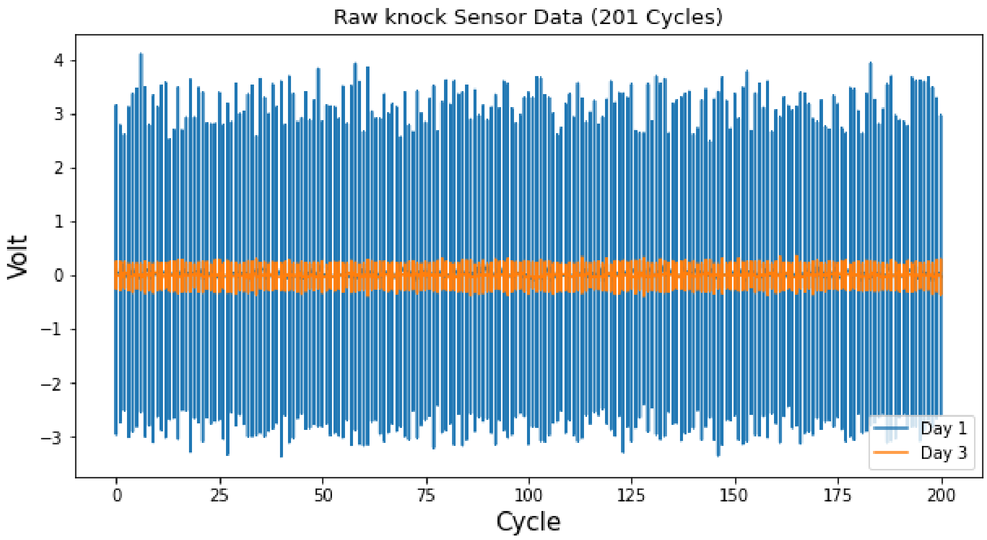


*Figure 5.6 Day 3 of raw spark plug data*

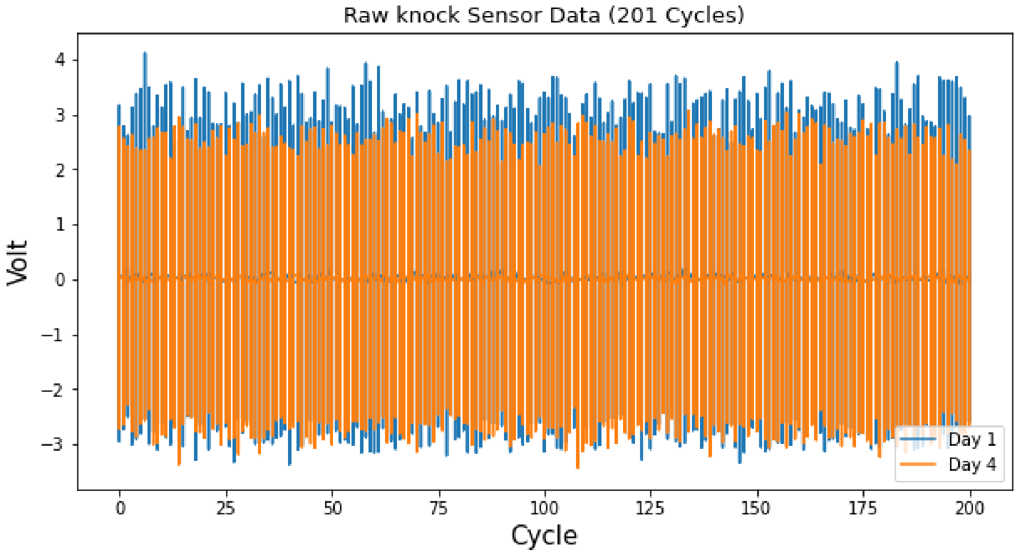


*Figure 5.7 Day 4 of raw spark plug data*

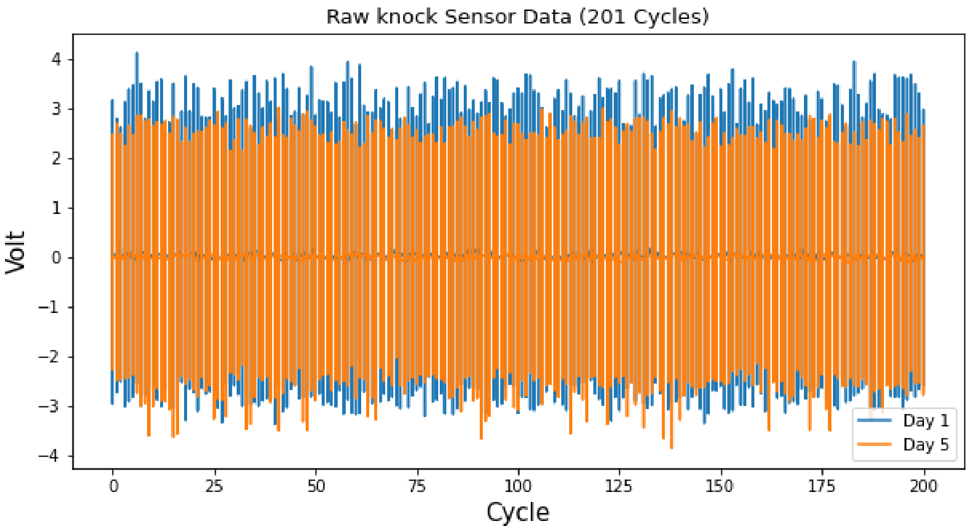


*Figure 5.8 Day 5 of raw spark plug data*

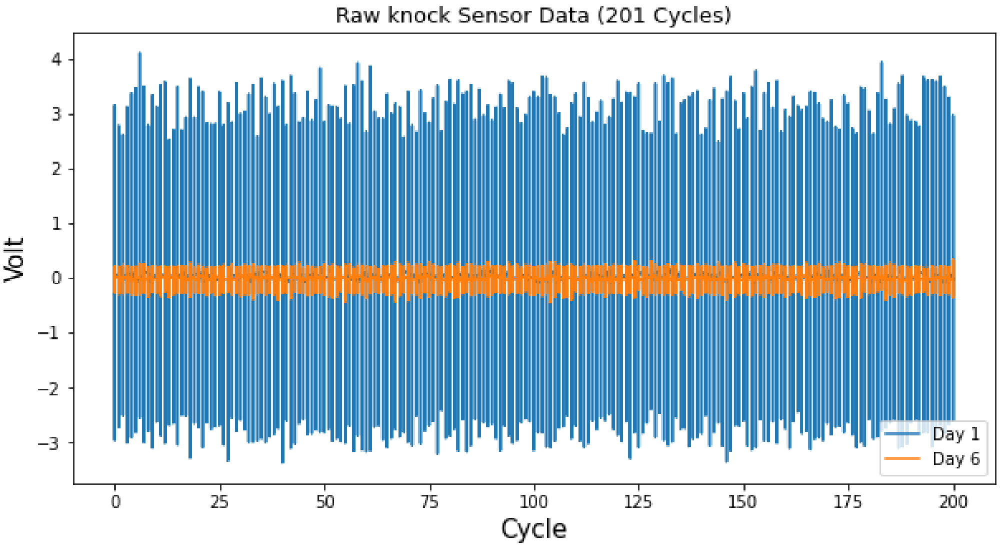


*Figure 5.9 Day 6 of raw spark plug data*

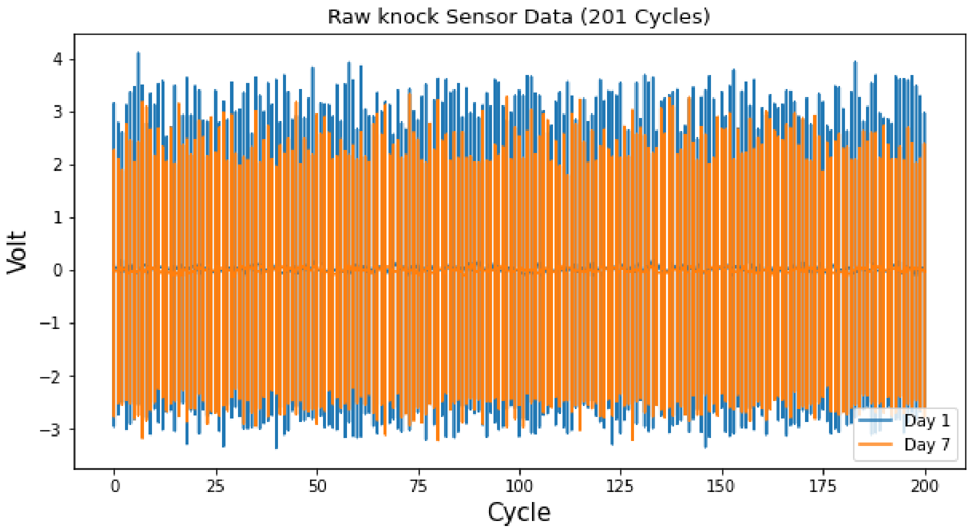


***Figure 5.10 Day 7 of raw spark plug data***

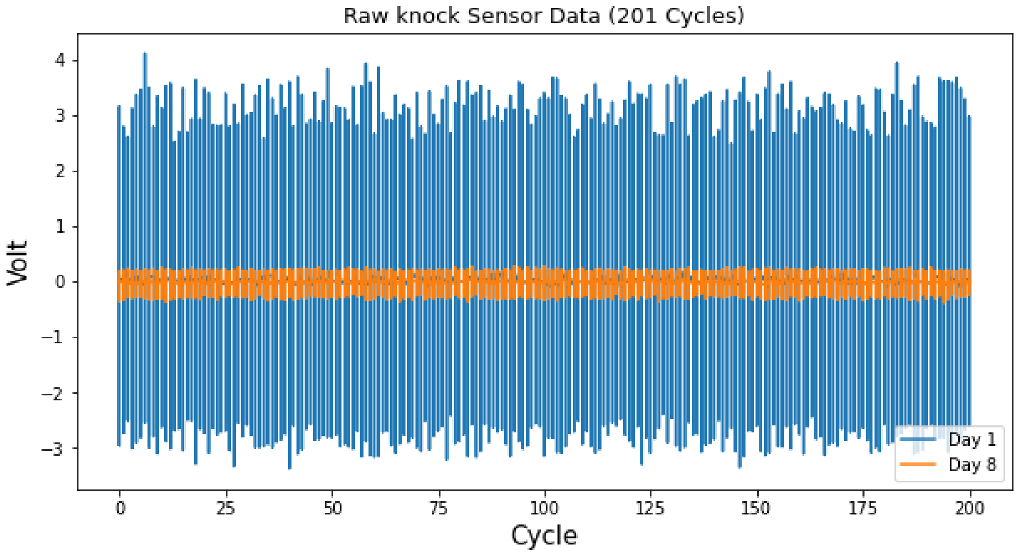


***Figure 5.11 Day 8 of raw spark plug data***

A clear drop in voltage in the crank angle domain was noticed in days 3, 6 and 8. This voltage drop introduces an undesired pattern to the signal, which might result in FDD misclassifications attributed to the spark plug rather than sensor fault. Therefore, all data samples of days 3, 6 and 8 were flagged as invalid and were dismissed from the training and testing process. The previous analysis was based on knock sensor #1 which showed abnormal behavior and was then replaced for further data collection.

### *4.3.1 Spark Plug Data Statistical Analysis*

Noisy signals such as knock sensor data may not be easy to visualize using raw data. Therefore, further statistical analysis was required to confirm the previous observations in the raw data. Many traditional statistical analysis methods can be used for such investigation. Mean value, variance and standard deviation are common methods and were used in this investigation. If more anomalies were found in this analysis on top of what was observed already in the raw data visualization, then further analysis would have been required. However, the statistical analysis results matched with those of the visualization. Figure 5.12 shows the results of the statistical analysis over spark plug data. All the 8 days almost shared the same mean value. This was because even the data samples with lower magnitudes were symmetric around zero voltage, which makes the average almost consistent. Therefore, the mean value was not a good tool to see any uncommon behavior in the data. Variance was a better indicator for anomalies. It represents the average of the squared differences from the mean value. While variance was normal in days 1, 2, 4, 5 and 7, it showed noticeable change in day 3, 6 and 8, which confirms the previous visualization method. Standard deviation, which represents the square root of the variance, was also used as a statistical analysis. Standard deviation shows the deviation of each data point from the mean value. Figure 5.12 shows a significant drop in standard deviation in days 3, 6 and 8 as well. The results of the variance and standard deviation confirmed the same conclusion of the visualization tool.

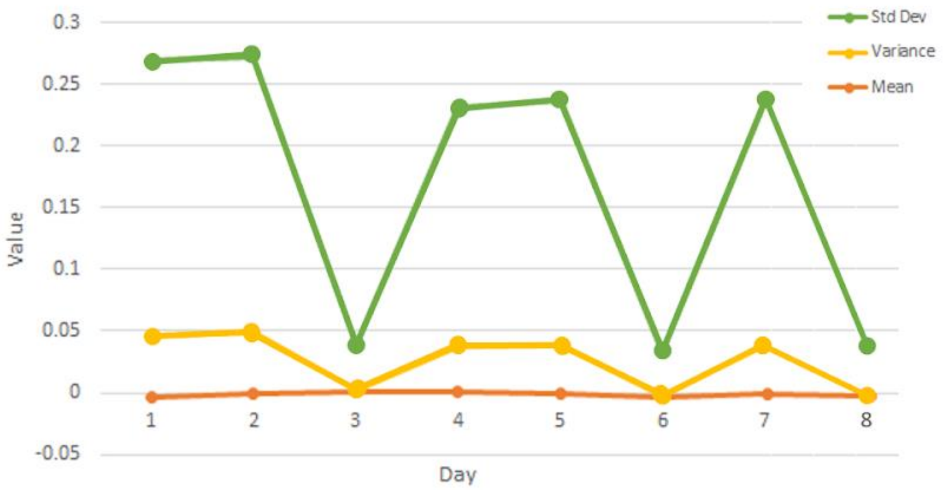


*Figure 5.12 Spark Plug Data Statistical Analysis*

### *5.3.2 Spark Plug Data Distribution Fitter Analysis*

The previous analysis methods were both in the crank angle domain. It was worth looking at the data from a different point of view. The frequency domain is a common way to look at the data from a different perspective and see information which was not clear in other domains. The distribution fitter toolbox in MATLAB was used in this analysis to see how the data looks like in the frequency domain. This toolbox provides a visual and interactive approach to fitting univariate distributions to data. Figures 5.13 – 5.19 show an example from day 2 to day 8, respectively, against day 1, respectively.

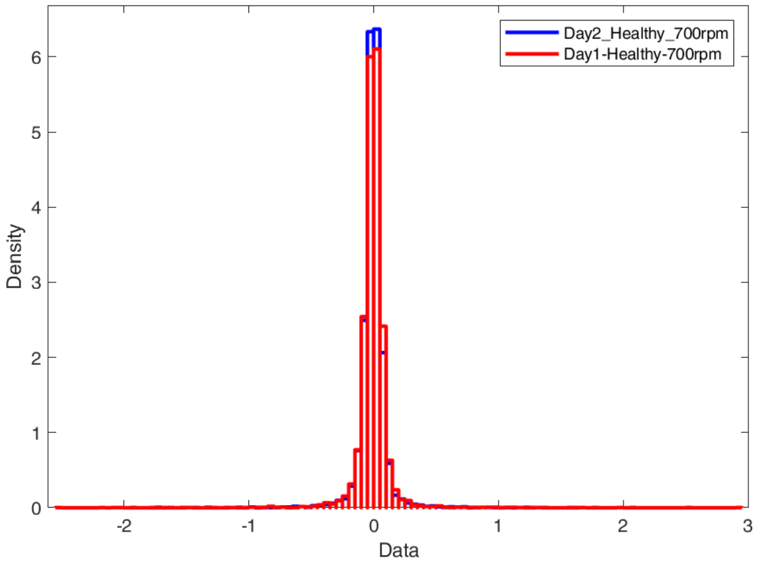


***Figure 5.13 Distribution fitter results of day 1 and 2 of spark plug data***

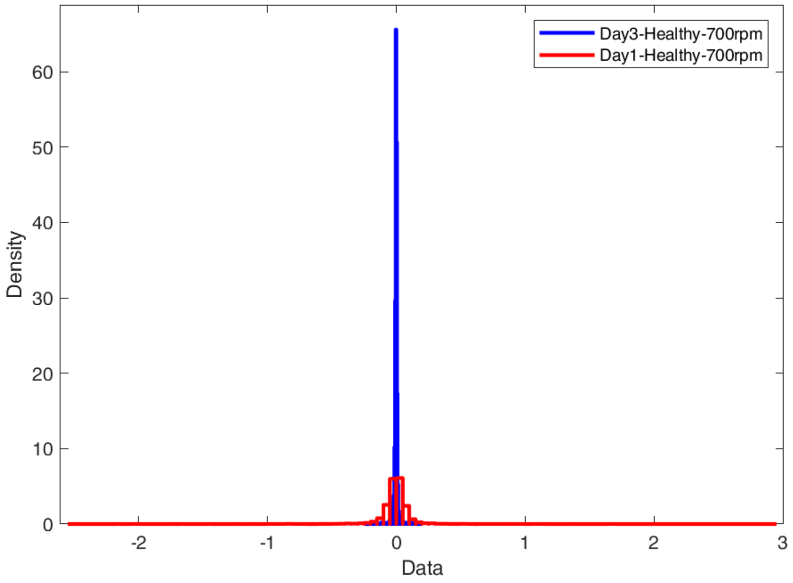


***Figure 5.14 Distribution fitter results of day 3 of spark plug data***

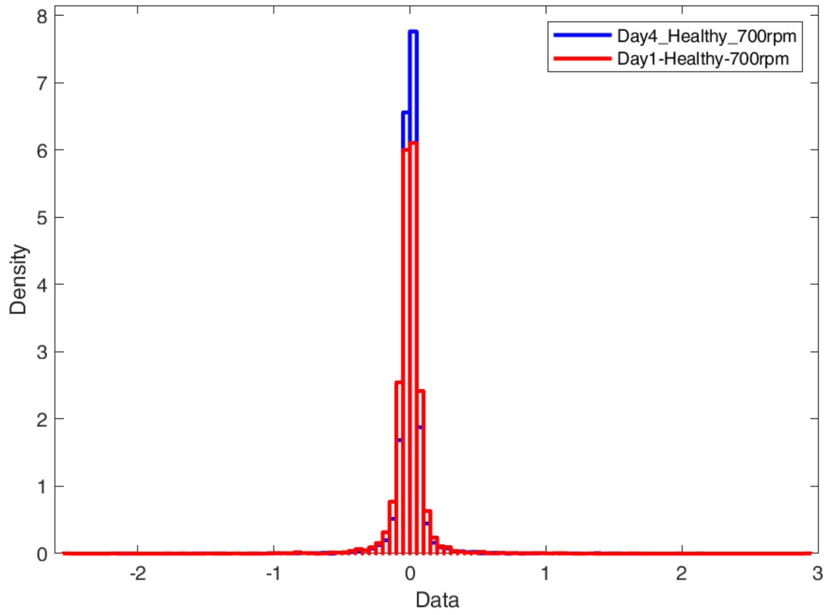


*Figure 5.15 Distribution fitter results of day 4 of spark plug data*

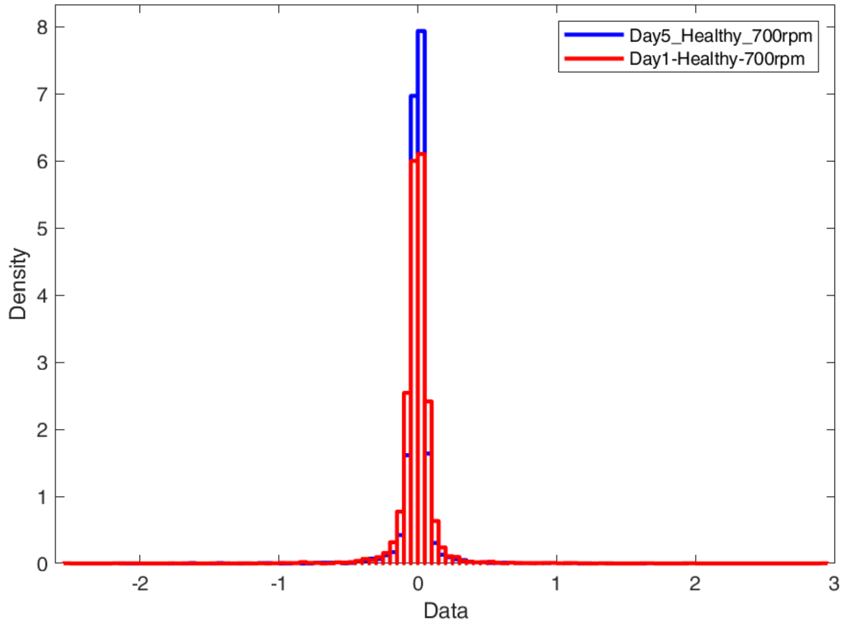


*Figure 5.16 Distribution fitter results of day 5 of spark plug data*

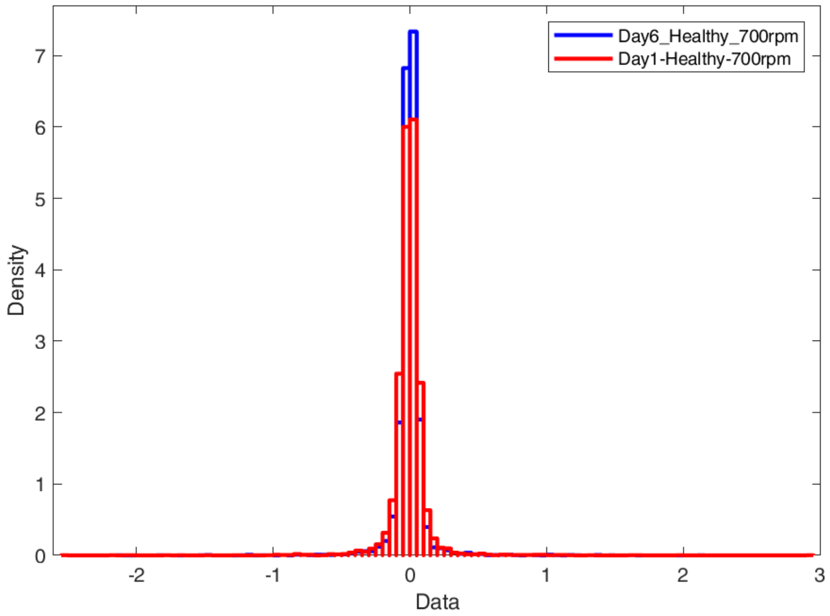


*Figure 5.17 Distribution fitter results of day 6 of spark plug data*

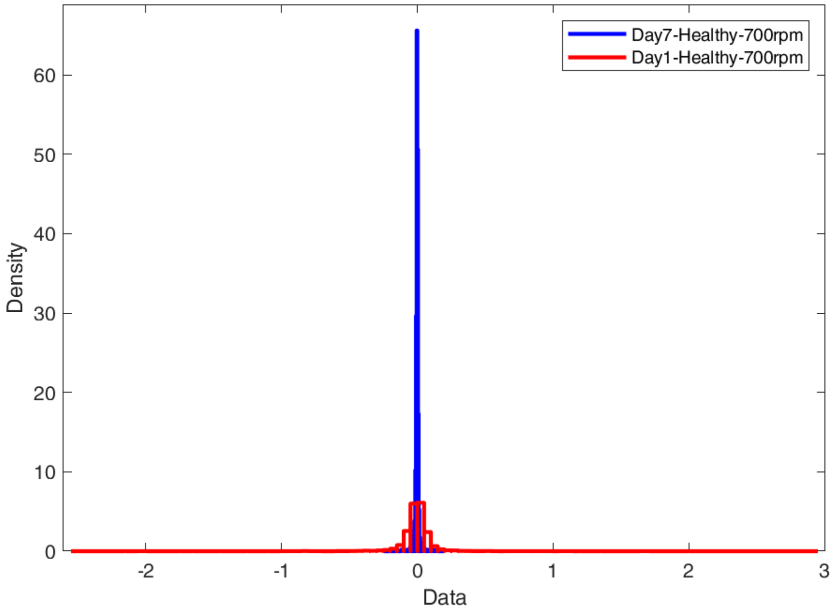


*Figure 5.18 Distribution fitter results of day 7 of spark plug data*

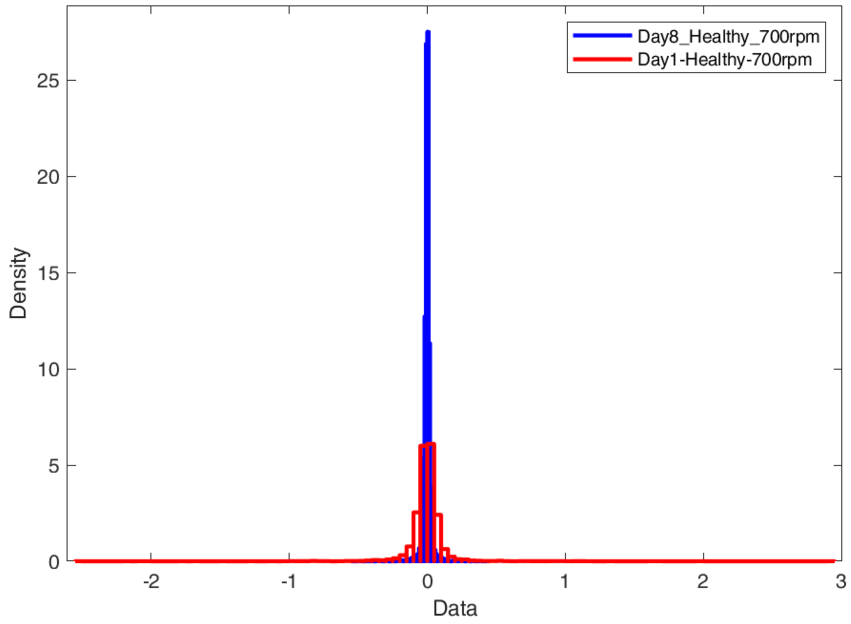


*Figure 5.19 Distribution fitter results of day 8 of spark plug data*

The MATLAB-based distribution fitter toolbox confirmed the anomalies in day 3 and 8, but showed a normal behavior in day 6, which was not the case in the crank angle domain. Day 7 also showed a different pattern, which was not clear in the other domain. This was the reason why it was necessary to look at the data in different domains. Therefore, day 7 was also flagged as invalid and was not used for training and testing. Only days 1, 2, 4 and 5 were used in the results which will be discussed later in this thesis.

## *5.4 Exhaust Gas Recirculation (EGR) Valve Data Collection*

The exhaust gas recirculation (EGR) valve is an engine management component. It recirculates exhaust gas to the engine intake system to increase engine efficiency. This also reduces the fuel consumption rate and lowers NOx emissions. Detecting EGR valve faults using Machine Learning (ML) required historical data of the different levels of failure. This included examples of defective EGR valves as well as perfectly healthy valves. EGR valves may fail because of several reasons, such as the valve mechanism being stuck open or closed, or the air filter being clogged or dirty. Deficiency of other engine parts such as, intake leaks, incorrect vacuum hose connection and turbocharger problems may result in EGR valve faults or complete failure. Like any other engine part, the EGR valve's life may simply come to an end after a high amount of mileage. One of the symptoms of EGR valve faults is engine knocking.

The EGR faults were simulated by introducing external CO2 to the engine which results in diluting the combustible mixture with CO2 gas. Two levels of CO2 dilution were introduced to the engine: Level 1 which represents 5% and Level 2 which represents 10%. Three different levels of CO2 dilution:

- Level 0 = 0% (Healthy)
- Level 1 = 5% CO2 dilution
- Level 2 = 10% CO2 dilution

The 3 different EGR levels represent 3 classes for the neural networks which were built to classify EGR fault conditions. The 3 classes are presented in Table 5.8 along with their corresponding dilution levels as follows:

| Class | EGR Level | CO2 Dilution |
|---|---|---|
| **Healthy** | Level 0 | 0% |
| **Fault 1** | Level 1 | 5% |
| **Fault 2** | Level 2 | 10% |

***Table 5.8 EGR levels***

Data collection occurred at 2 different operating conditions while EGR faults were being introduced to the engine, namely operating condition 1 and operating condition 2. In condition 1, a constant engine speed of 700 rpm and 15% engine load were applied, while in condition 2, a constant speed of 1000 rpm and 25% load were applied. The operating conditions are stated below in Table 5.9.

| Condition | Engine Speed | Engine Load |
|---|---|---|
| **Operating Condition 1** | 700 rpm | 15% load |
| **Operating Condition 2** | 1000 rpm | 25% load |

***Table 5.9 Engine operating conditions while collecting EGR data***

The dynamometer was connected to a visualization software where the collected data could be monitored. The same sensors which were used to collect engine data with different spark plugs were also used for the EGR fault conditions and each of the operating conditions stated above. A matrix showing the collected data in detail is provided in Table 5.10.

| EGR Level | Level 0 (Healthy) | | | | Level 1 (Fault 1) | | | | Level 2 (Fault 2) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Operating Conditions | 700 rpm (15% load) | | 1000 rpm (25% load) | | 700 rpm (15% load) | | 1000 rpm (25% load) | | 700 rpm (15% load) | | 1000 rpm (25% load) | |
| Time of the Day | Morning | Afternoon | Morning | Afternoon | Morning | Afternoon | Morning | Afternoon | Morning | Afternoon | Morning | Afternoon |
| Day 1 | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples |
| Day 2 | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples |
| Day 3 | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples |
| Day 4 | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples |
| Day 5 | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples |
| Day 6 | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples |
| Day 7 | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples |
| Day 8 | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples | 100 samples |

*Table 5.10 EGR data*

Like spark plug data, another round of analysis had to be conducted over the EGR data to ensure its validity. Both crank angle and frequency domain tests were conducted to check the validity of every day of the EGR data. Variant samples were found in some days due to experimental and technical issues.

## *5.5 EGR Data Variability*

First, the raw knock sensor data were plotted and visually inspected to search for abnormal behaviors. Figure 5.20 represents 201 engine cycles of knock sensor # 1 signal that resulted from a healthy EGR valve in day 1. The signal shows a normal behavior with an average amplitude of 3.0 V, which means that the sample was valid. All data samples of day 1 were checked using visualization tools and no anomalies were observed in day 1.

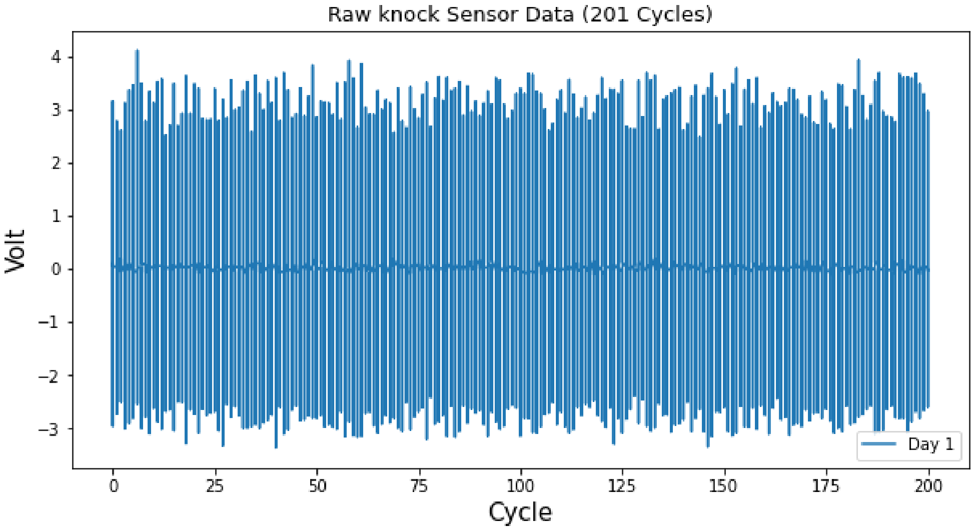


***Figure 5.20 Day 1 of EGR data***

The same visualization check was conducted over all the 8 days. Figures 5.21 and 5.22 show the raw signals of knock sensor # 1 of day 2 and day 3, respectively, both against day 1. Day 2 clearly has similar specifications, unlike day 3 which is variant.

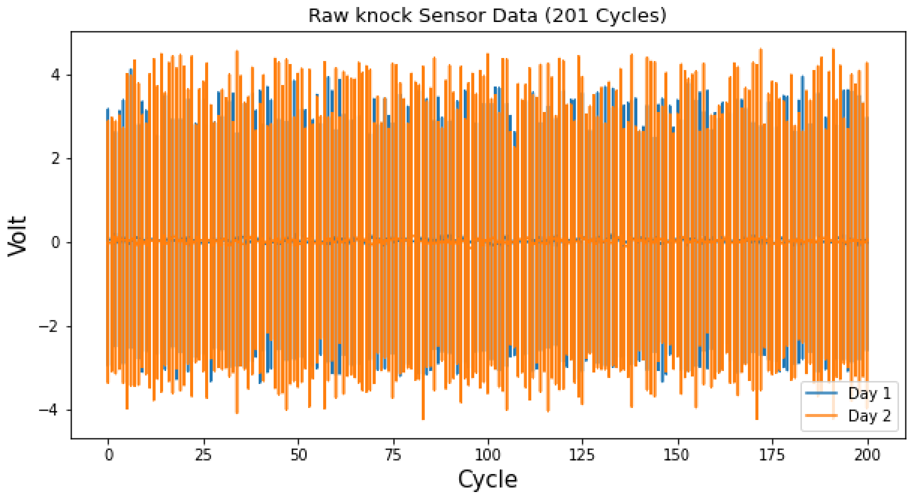


*Figure 5.21 Day 2 of EGR data*

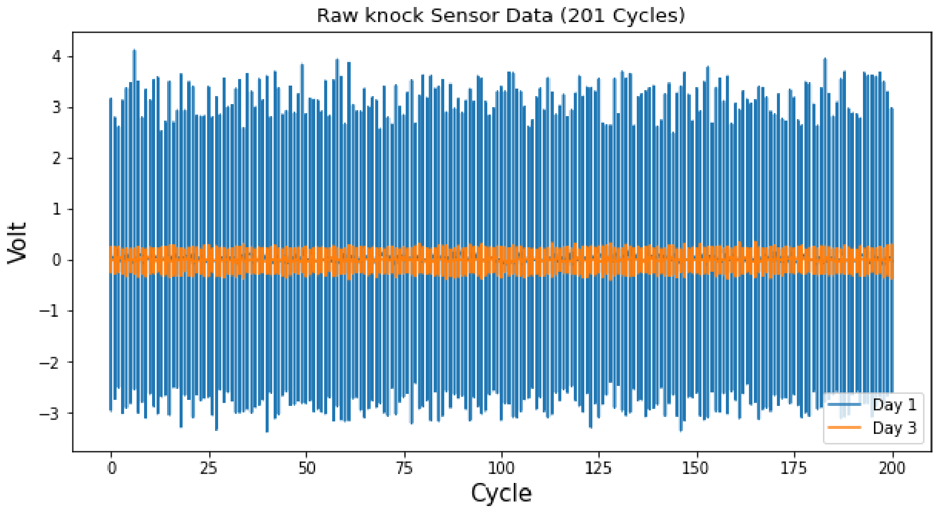


*Figure 5.22 Day 3 of EGR data*

Voltage drops in day 3, 7 and 8 were noticed in the crank angle domain. This anomaly may lead to misclassification since samples of the same class do not share the same patterns. Therefore, all data samples of day 3, 7 and 8 were flagged as variant EGR data and were dismissed from the training and testing process. All the samples plotted above represented signals of knock sensor #1 which was replaced for further data collection.

### 5.5.1 EGR Data Statistical Analysis

The same analysis was conducted to EGR data as well. Figure 5.23 shows the results of the statistical analysis over EGR data. All the 8 days had similar mean values. Although voltage amplitudes were significantly different, they were all symmetric around zero voltage, which made the average almost zero. Therefore, the mean value did not help with seeing any abnormal behavior in the data.

Variance was a better indicator for anomalies. It represents the average of the squared differences from the mean value. While variance was normal in days 1, 2, 4, 5 and 6, it showed noticeable change in days 3, 7 and 8, which confirms the previous visualization method. Standard deviation, which represents the square root of the variance, was also used as a statistical analysis. Standard deviation shows the deviation of each data point from the mean value. A significant drop in standard deviation in days 3, 7 and 8 was observed. Both variance and standard deviation led to the same conclusion of the visualization tool.

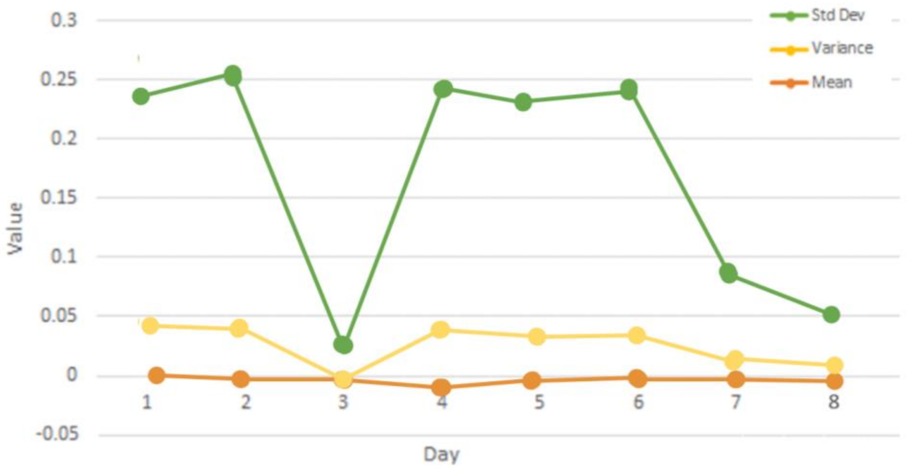


*Figure 5.23 EGR data statistical analysis*

### 5.5.2 EGR Data Fault Signatures

The EGR data was analyzed in the frequency domain as well using EMSPCA fault signatures to extract further information which was not clear in the crank angle domain. The fault signatures were used in this analysis to see the data trend in the frequency domain. This feature provides a visual representation which showed the deviation of every sample from the healthy baseline, such as a healthy EGR data sample from Day 1.

Distribution analysis was also conducted to observe the variance of the data in variant days. The fault signatures of these days were generated and visualized. Figure 5.24 - 5.26 show the average fault signature of healthy, level 1 (i.e., fault 1) and level 2 (i.e., fault 2) EGR data samples, respectively, of all the 4 knock sensors. Level 1 represents 5% of CO2 dilution while level 2 represents 10%. The average was taken of days 1, 2, 4 and 5, which were perfectly valid based on previous analysis. Fault signatures of healthy data look flat since they were compared to similar healthy baselines, while those of level 1 and level 2 CO2 dilution show intense amplitudes which explains the effect of the CO2 dilution on engine knocking.

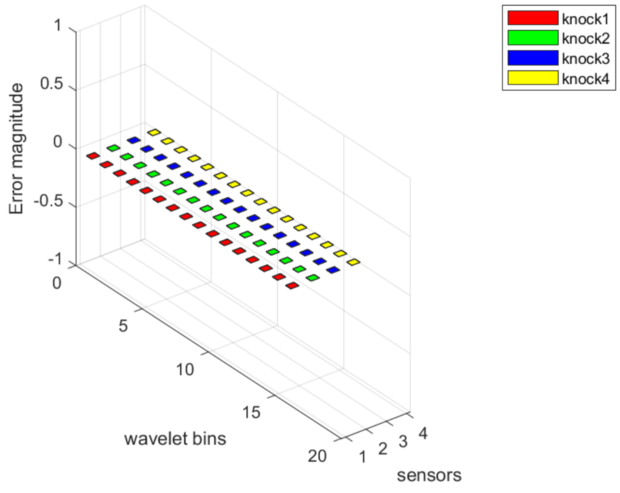


*Figure 5.24 Average fault signature of healthy EGR data*

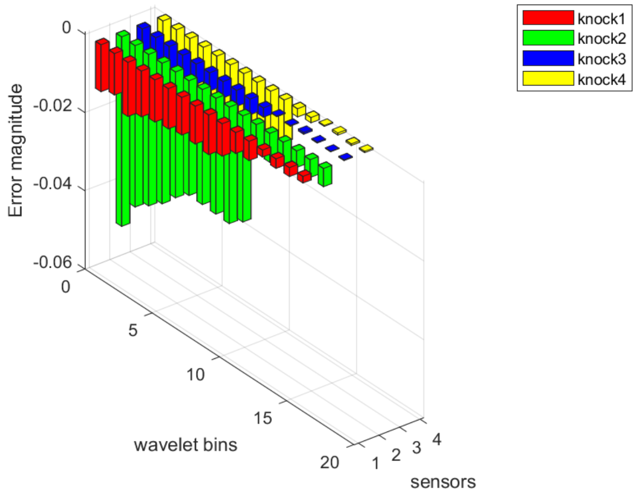


*Figure 5.25 Average fault signature of fault 1 EGR data*

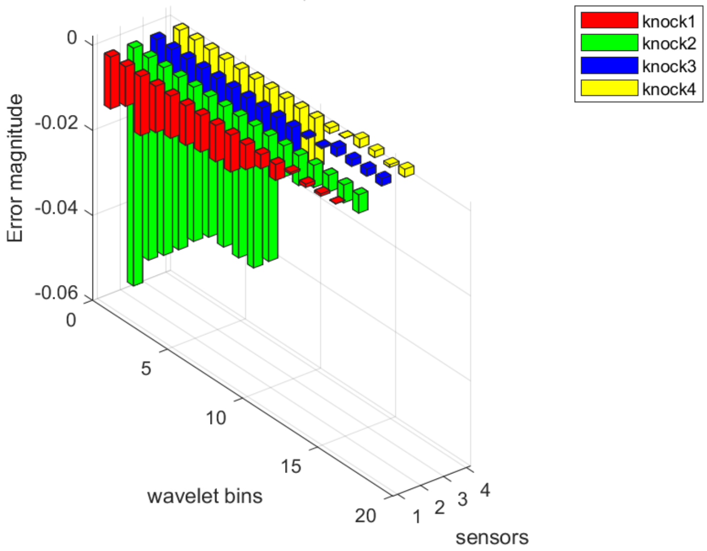


*Figure 5.26 Average fault signature of fault 2 EGR data*

To examine the days which were previously flagged as variant, their fault signatures were generated. Figure 5.27 shows a healthy, level 1 and level 2 samples from day 7, which is an example of fault signatures of variant days which clearly have different patterns than the averaged valid fault signatures. This confirmed that there might have been something wrong while collecting the data on these days.

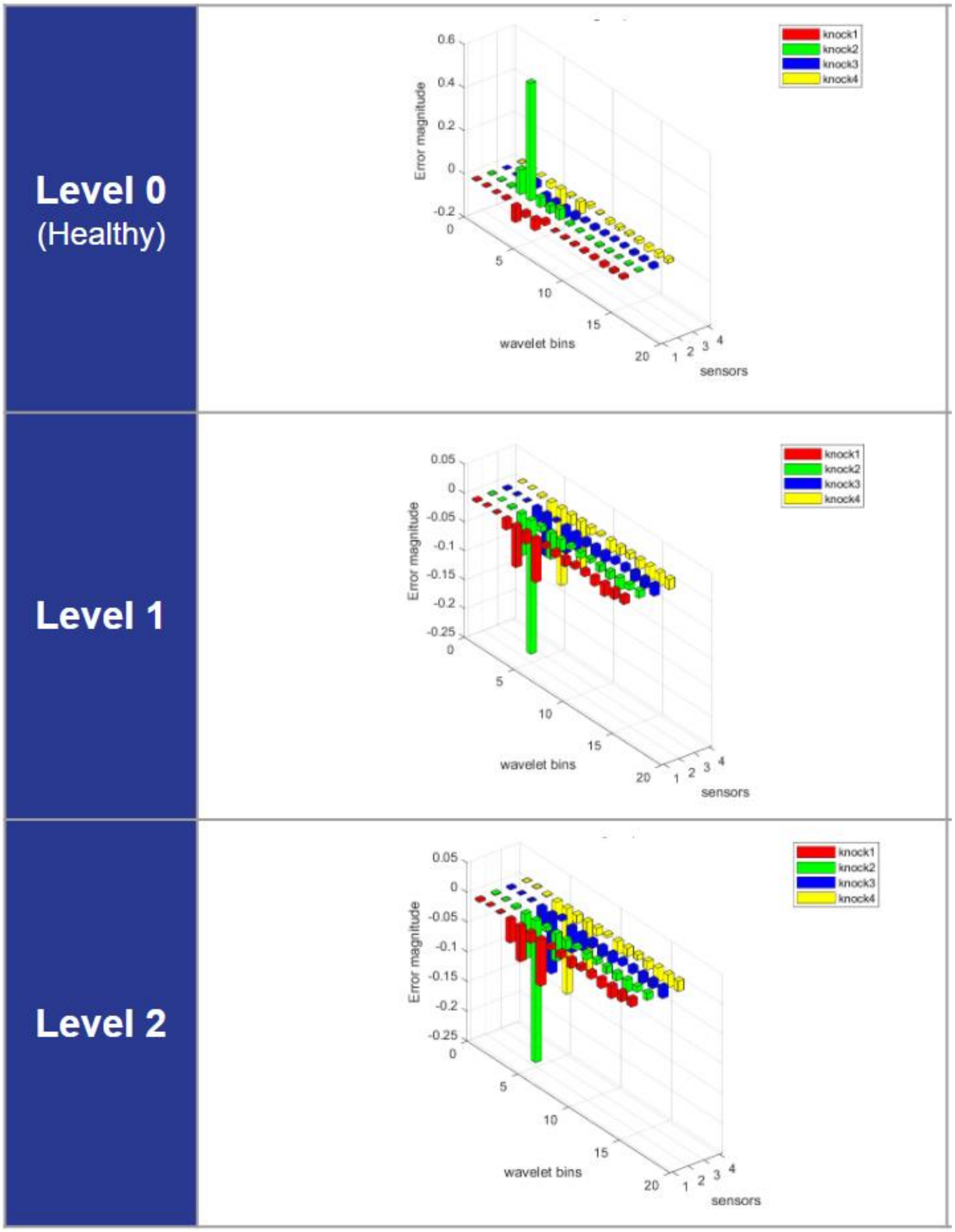
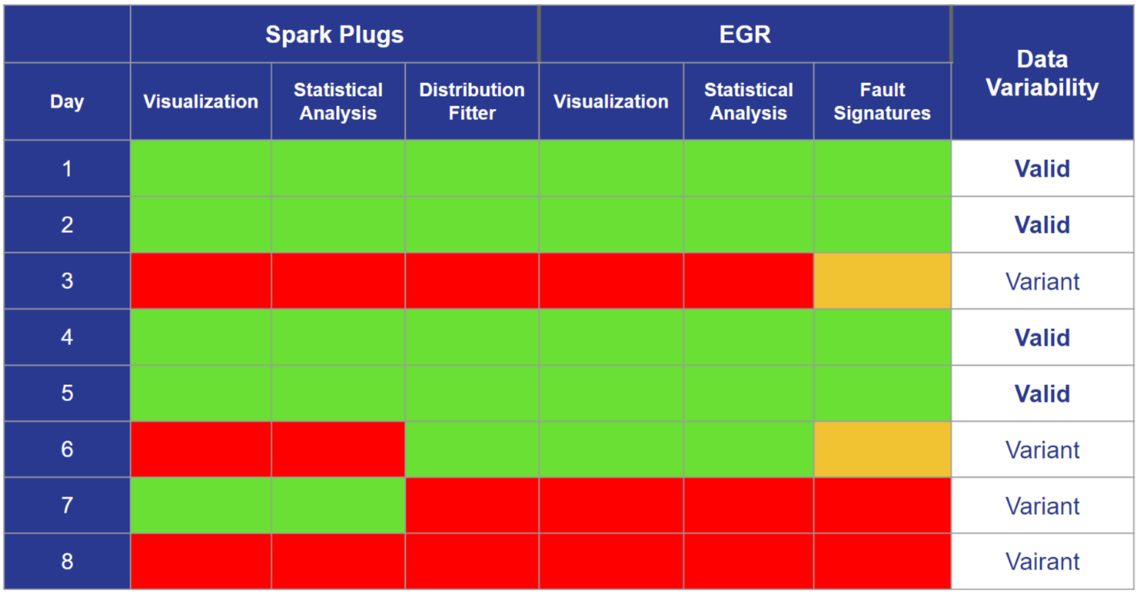


***Figure 5.27 Fault signature of EGR data from day 7***

The fault signatures confirmed the anomalies in day 7 and 8, but showed an acceptable behavior in day 6, which was not the case in the crank angle domain.

## 5.6 *Summary of Spark Plug and EGR Data Analysis*

Finding new issues in each type of analysis was the reason why it was necessary to look at the data in different domains. Some issues were clear in the crank angle domain while others were found in the frequency domain. To summarize the conducted analysis, Table 5.11 shows each analysis along with the issues found in each day. The red color means significant variance, yellow means suspicious and green means identical. To ensure a perfectly clean dataset for training and testing, any day with either significant or suspicious issues (i.e., red, and yellow) was flagged as variant. Only green days were considered valid.



| | Spark Plugs | | | EGR | | | Data Variability |
|---|---|---|---|---|---|---|---|
| Day | Visualization | Statistical Analysis | Distribution Fitter | Visualization | Statistical Analysis | Fault Signatures | |
| 1 | 🟩 | 🟩 | 🟩 | 🟩 | 🟩 | 🟩 | **Valid** |
| 2 | 🟩 | 🟩 | 🟩 | 🟩 | 🟩 | 🟩 | **Valid** |
| 3 | 🟥 | 🟥 | 🟥 | 🟥 | 🟥 | 🟨 | Variant |
| 4 | 🟩 | 🟩 | 🟩 | 🟩 | 🟩 | 🟩 | **Valid** |
| 5 | 🟩 | 🟩 | 🟩 | 🟩 | 🟩 | 🟩 | **Valid** |
| 6 | 🟥 | 🟥 | 🟩 | 🟩 | 🟩 | 🟨 | Variant |
| 7 | 🟩 | 🟩 | 🟥 | 🟥 | 🟥 | 🟥 | Variant |
| 8 | 🟥 | 🟥 | 🟥 | 🟥 | 🟥 | 🟥 | Vairant |

*Table 5.11 Fault signature of EGR data from day 8*

The table shows that days 3, 6, 7 and 8 were all flagged as variant and were not used for training and testing. Only days 1, 2, 4 and 5 were used in the training and classification process. The next step was to test the capability of the EMSPCA algorithm on diagnosing variant data. Based on the results which will be discussed in chapter 8, the algorithm was able to detect faults in regular data samples from day 1,2,4 and 5, as well as variant data samples from day 3, 6, 7 and 8. However, the algorithm was not able to diagnose faults using raw fault signatures and Artificial Neural Network (ANN). Thus, an invariant AI method was needed to solve this problem.

As explained earlier, the EMSPCA algorithm generates fault signatures which describe the variance of the test sample against healthy baselines. However, this technique works well within a certain range of data variability. When the raw sensory data are corrupted and variant beyond what is caused by a known fault condition, the ANN could not distinguish between the different levels of severity of fault conditions. To remedy this, the ANN was replaced with a CNN as an invariant deep learning classifier. The difference between ANN and CNN is the ability to maintain high-accuracy classification regardless of data variance. The CNN gains this ability from its max pooling layers (see section 2.5.2.) which does not exist in the ANN. Max pooling takes the highest value from each patch of an image and ignores the rest of the information in the activation maps. This indicates that an important incident has happened in this patch can not tell where exactly it happened. This allows changes to the signal due to experimental issues within an acceptable range.

# Chapter 6

# Engine Fault Classification Using Artificial Intelligence (AI)

# *Chapter 6 – Spark Plug FDD Using Artificial Intelligence (AI)*

In this chapter, Machine Learning (ML) and Deep Learning (DL) models are used as tools to learn from the engine database and detect the degradation of two engine parts: spark plugs and Exhaust Gas Recirculation (EGR) valves.

Detecting spark plug degradation using existing engine sensors was the most difficult problem in this research. Spark plug performance degradation is often a consequence of a change in the air gap size. Given that the engine has 8 cylinders, the degradation of a single spark plug results in a small deviation in the sensory measurements. The biggest challenge was to select a proper learning model for each kind of input. For example, Convolutional Neural Networks would not be a proper classification model for raw time-series spark plug data since they were designed to classify images. Time-series classification models such as Recurrent Neural Networks (RNN) would be a better match. Similarly, a proper pre-processing approach is important to emphasize patterns in the input data before fitting them to the learning model.

Two existing sensors were selected to collect engine data while defective spark plugs are placed inside the engine: knock sensors and speed sensor. The knock sensors were selected as the primary sensors since they can measure engine vibrations and since one of the most common symptoms of defective spark plugs is engine knocking. The speed sensor was selected since defective spark plugs may cause a change in the pressure inside the combustion chamber and thus, a change in the piston power and flywheel speed.

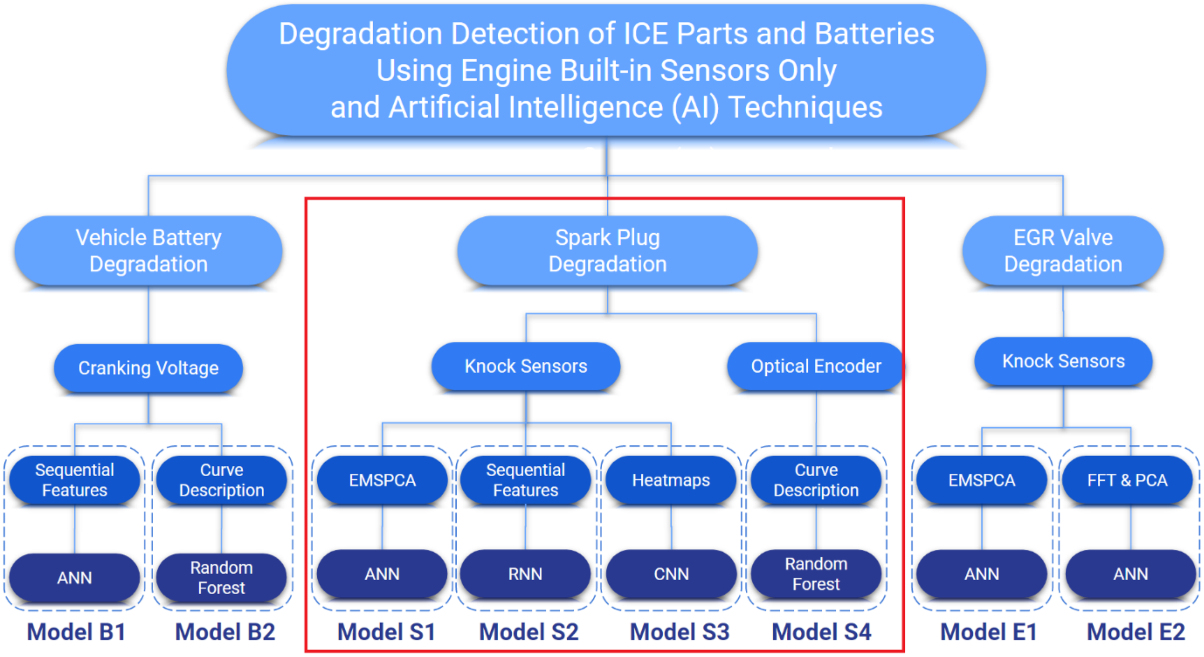


*Figure 6.1 Hierarchy of spark plug degradation detection models*

Four AI-based models were built to detect spark plug degradation, namely Models S1, S2, S3, and S4 (see Figure 6.1). The first three models were trained with the measurements of four knock sensors only while the last model was based on speed sensor measurements. The reason why three models were made for the knock sensors will be discussed in the results chapter. Model S1 consists of McMaster CMHT's advanced feature extraction algorithm, the Extended Multi-Scale Principal Component Analysis (EMSPCA) along with a simple Artificial Neural Network (ANN). Model S2 starts with a simple feature engineering layer, followed by a Deep Recurrent Neural Network (RNN), called Long Short-Term Memory (LSTM) network. Like the previous model, Model S3 has a simple feature engineering step and feeds then into a Deep Convolutional Neural Network (CNN). Model S4, which uses optical encoder sensory data, consists of a regular Random Forest (RF) classifier which was trained on a set of the new Curve Descriptive (CD) features proposed in this research.

## *6.1 Spark Plug Degradation Detection Using EMSPCA*

As explained in the introduction chapter, the Extended Multi-Scale Principal Component Analysis (EMSPCA) algorithm was previously developed at McMaster's CMHT research lab. The algorithm has shown excellent results in the fault identification of electric motors such as starters and alternators [61] and has been used in this research to support FDD of Internal Combustion Engines as well.

One of the contributions of this research was to improve the run time of the CMHT's EMSPCA algorithm to make it more time effective while classifying engine data without affecting the performance negatively. For comparison purposes, 300 data samples per class were used to assess the time improvements. Having 3 classes of spark plugs (i.e., 0.020", 0.050", and 0.080") resulted in having 900 data samples in total to train the model. In 2019, the run time of the original version of the EMSPCA algorithm (a.k.a. EMSPCA_v0) was about 1,416 hours (i.e., 59 days) which was uncompetitive when compared to other advanced classification algorithms. The first step to improve the run time was to look for complex logic operations in the training process that could be simplified. The EMSPCA algorithm uses healthy training samples to create healthy baselines and applies Wavelet Packet Transform (WPT) to all of them. Then it applies WPT to faulty training samples and compares them to all healthy baselines as shown in Figure 6.2. This meant that WPT was applied 300 times for 900 training samples, which tallied up to 270,000 times. Given the large number of data points in each sample, this explained the high run time. After a review of the fault signatures, it became evident that as expected, healthy baselines were very similar. The results which come from training the algorithm with multiple healthy baselines were therefore almost the same as just using ne baseline. Therefore, the average of all data samples of the same conditions was used as a single baseline. These updates were made to EMSPCA_v1.0 as shown in Figure 6.3.

The next step was to investigate the hardware configurations which were used to run the EMSPCA algorithm. EMSPCA_v0 was originally operated on a desktop computer with a 2.3GHz i4 processor. Given that the algorithm is MATLAB-based, two possible solutions were proposed to enhance the run time: Graphical Processing Unit (GPU) and Parallel Computing. At the time of this research, MATLAB was supporting GPU functioning for only 500 MATLAB functions. Many of the Wavelet Packet Transform (WPT) functions used in the algorithm were not among the supported functions.

MATLAB R2020b offered Parallel Computing for many of its functions. Parallel Computing allows a single operation, such as a for loop, to be distributed over multiple cores which run in parallel then gathers the results before moving to the next function. However, some operations were not eligible for parallel computing due to the nature of the functions inside the for loop. All eligible functions were updated to support parallel computing, which resulted in a new version, namely EMSPCA_v1.1. This enhanced the original run time by 63%, brining it down to 528 hours. A new high-performance desktop computer with an Intel Xeon Gold 5222, 16.5M cache and 3.8GHz, processor was purchased for the EMSPCA algorithm. The run time on the new computer was 288 hours which decreased the original run time by 80%. A new graphics card was installed later in the new high-performance computer, which unexpectedly decreased the run time to 96 hours.

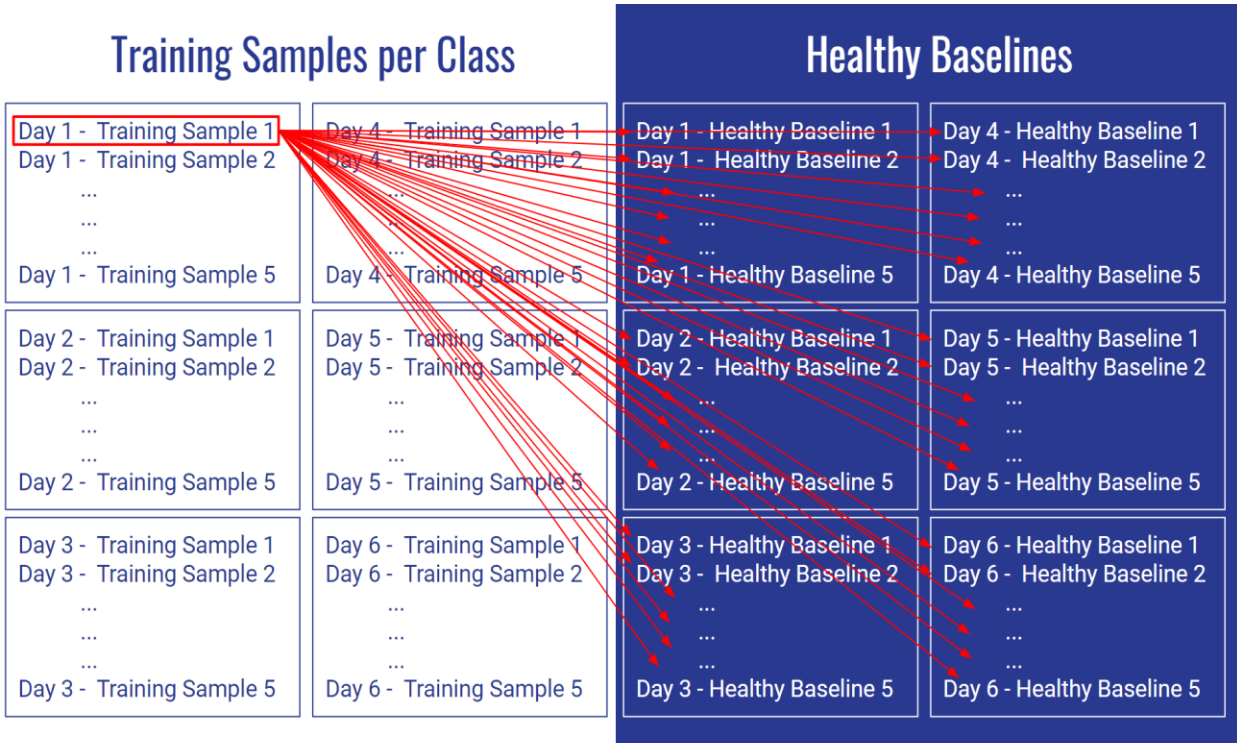

Training Samples per Class
Healthy Baselines
Day 1 - Training Sample 1
Day 1 - Training Sample 2
...
...
...
Day 1 - Training Sample 5
Day 4 - Training Sample 1
Day 4 - Training Sample 2
...
...
...
Day 4 - Training Sample 5
Day 1 - Healthy Baseline 1
Day 1 - Healthy Baseline 2
...
...
...
Day 1 - Healthy Baseline 5
Day 4 - Healthy Baseline 1
Day 4 - Healthy Baseline 2
...
...
...
Day 4 - Healthy Baseline 5
Day 2 - Training Sample 1
Day 2 - Training Sample 2
...
...
...
Day 2 - Training Sample 5
Day 5 - Training Sample 1
Day 5 - Training Sample 2
...
...
...
Day 5 - Training Sample 5
Day 2 - Healthy Baseline 1
Day 2 - Healthy Baseline 2
...
...
...
Day 2 - Healthy Baseline 5
Day 5 - Healthy Baseline 1
Day 5 - Healthy Baseline 2
...
...
...
Day 5 - Healthy Baseline 5
Day 3 - Training Sample 1
Day 3 - Training Sample 2
...
...
...
Day 3 - Training Sample 5
Day 6 - Training Sample 1
Day 6 - Training Sample 2
...
...
...
Day 6 - Training Sample 5
Day 3 - Healthy Baseline 1
Day 3 - Healthy Baseline 2
...
...
...
Day 3 - Healthy Baseline 5
Day 6 - Healthy Baseline 1
Day 6 - Healthy Baseline 2
...
...
...
Day 6 - Healthy Baseline 5

*Figure 6.2 EMSPCA_v0.0*

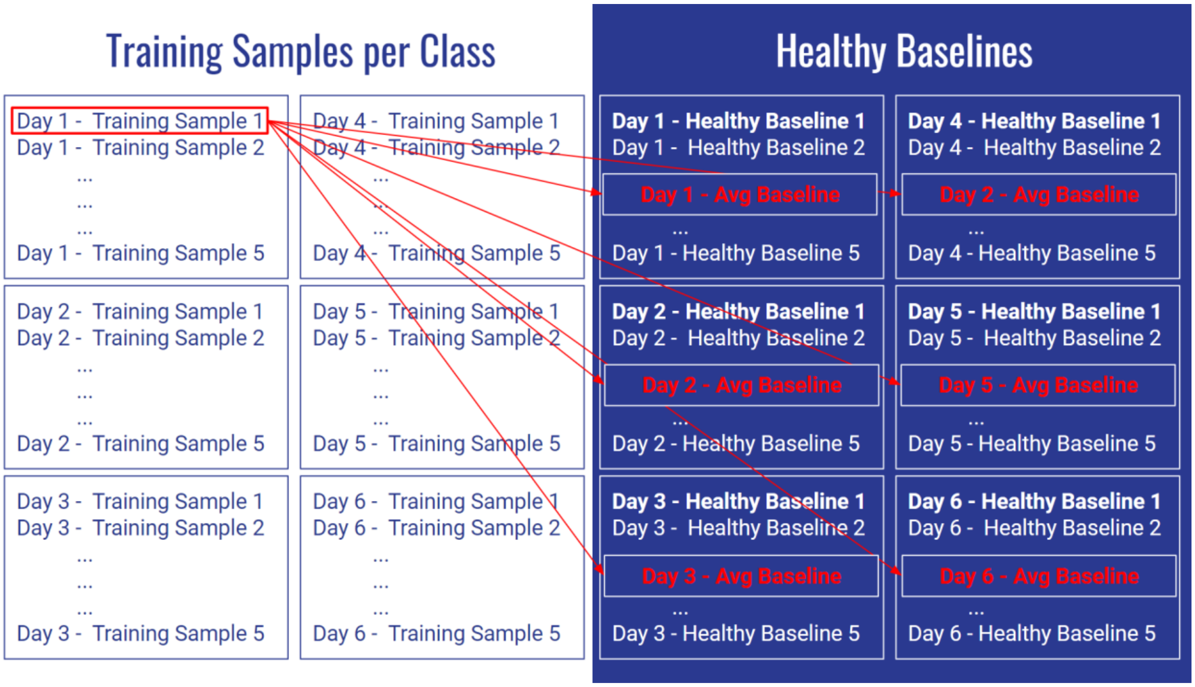
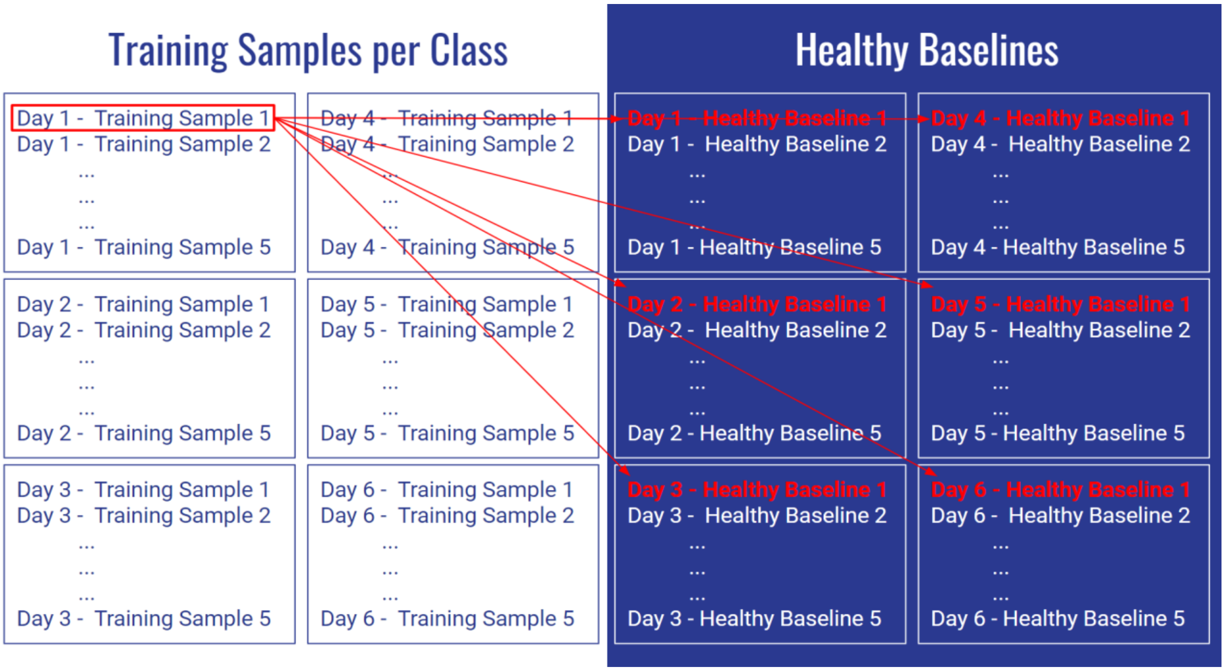

Training Samples per Class
Healthy Baselines
Day 1 - Training Sample 1
Day 1 - Training Sample 2
...
...
...
Day 1 - Training Sample 5
Day 4 - Training Sample 1
Day 4 - Training Sample 2
...
...
...
Day 4 - Training Sample 5
Day 1 - Healthy Baseline 1
Day 1 - Healthy Baseline 2
Day 1 - Avg Baseline
...
Day 1 - Healthy Baseline 5
Day 4 - Healthy Baseline 1
Day 4 - Healthy Baseline 2
Day 2 - Avg Baseline
...
Day 4 - Healthy Baseline 5
Day 2 - Training Sample 1
Day 2 - Training Sample 2
...
...
...
Day 2 - Training Sample 5
Day 5 - Training Sample 1
Day 5 - Training Sample 2
...
...
...
Day 5 - Training Sample 5
Day 2 - Healthy Baseline 1
Day 2 - Healthy Baseline 2
Day 2 - Avg Baseline
...
Day 2 - Healthy Baseline 5
Day 5 - Healthy Baseline 1
Day 5 - Healthy Baseline 2
Day 5 - Avg Baseline
...
Day 5 - Healthy Baseline 5
Day 3 - Training Sample 1
Day 3 - Training Sample 2
...
...
...
Day 3 - Training Sample 5
Day 6 - Training Sample 1
Day 6 - Training Sample 2
...
...
...
Day 6 - Training Sample 5
Day 3 - Healthy Baseline 1
Day 3 - Healthy Baseline 2
Day 3 - Avg Baseline
...
Day 3 - Healthy Baseline 5
Day 6 - Healthy Baseline 1
Day 6 - Healthy Baseline 2
Day 6 - Avg Baseline
...
Day 6 - Healthy Baseline 5

*Figure 6.3 EMSPCA_v1.0*

The averaging step of the healthy baselines resulted in a new version (a.k.a. EMSPCA_v2.0) which decreased the run time to 13 hours, which improved it by 99%. Figure 6.4 shows the logic which was followed in EMSPCA_v2.0. An additional version of the algorithm was designed for quick verification purposes, namely EMSPCA_v2.1. Only a single healthy baseline was taken from all baselines and compared to the training samples. The data samples were always introduced to the algorithm in batches. Each batch contains 200 engine cycles which the algorithm splits into 10 data samples, 20 cycles each. In version EMSPCA_v2.1, only one sample from each batch was used for training. This version was only being used to quickly confirm the validity of certain results and sometimes for easy-to-expect classification problems. The run time of this version was only half an hour.

Figure 6.5 shows a diagram of the different stages of improving the run time of the EMSPCA algorithm from 2019 to 2021. The diagram shows that the run time was decreased from 1,416 hours in EMSPCA_v0.0 to 0.5 hour in EMSPCA_v2.1, which presents a total decrease of 99.9% from the original version.



Training Samples per Class

Day 1 - Training Sample 1
Day 1 - Training Sample 2
...
...
...
Day 1 - Training Sample 5

Day 4 - Training Sample 1
Day 4 - Training Sample 2
...
...
...
Day 4 - Training Sample 5

Day 2 - Training Sample 1
Day 2 - Training Sample 2
...
...
...
Day 2 - Training Sample 5

Day 5 - Training Sample 1
Day 5 - Training Sample 2
...
...
...
Day 5 - Training Sample 5

Day 3 - Training Sample 1
Day 3 - Training Sample 2
...
...
...
Day 3 - Training Sample 5

Day 6 - Training Sample 1
Day 6 - Training Sample 2
...
...
...
Day 6 - Training Sample 5

Healthy Baselines

Day 1 - Healthy Baseline 1
Day 1 - Healthy Baseline 2
...
...
...
Day 1 - Healthy Baseline 5

Day 4 - Healthy Baseline 1
Day 4 - Healthy Baseline 2
...
...
...
Day 4 - Healthy Baseline 5

Day 2 - Healthy Baseline 1
Day 2 - Healthy Baseline 2
...
...
...
Day 2 - Healthy Baseline 5

Day 5 - Healthy Baseline 1
Day 5 - Healthy Baseline 2
...
...
...
Day 5 - Healthy Baseline 5

Day 3 - Healthy Baseline 1
Day 3 - Healthy Baseline 2
...
...
...
Day 3 - Healthy Baseline 5

Day 6 - Healthy Baseline 1
Day 6 - Healthy Baseline 2
...
...
...
Day 6 - Healthy Baseline 5

*Figure 6.4 EMSPCA_v2.0*

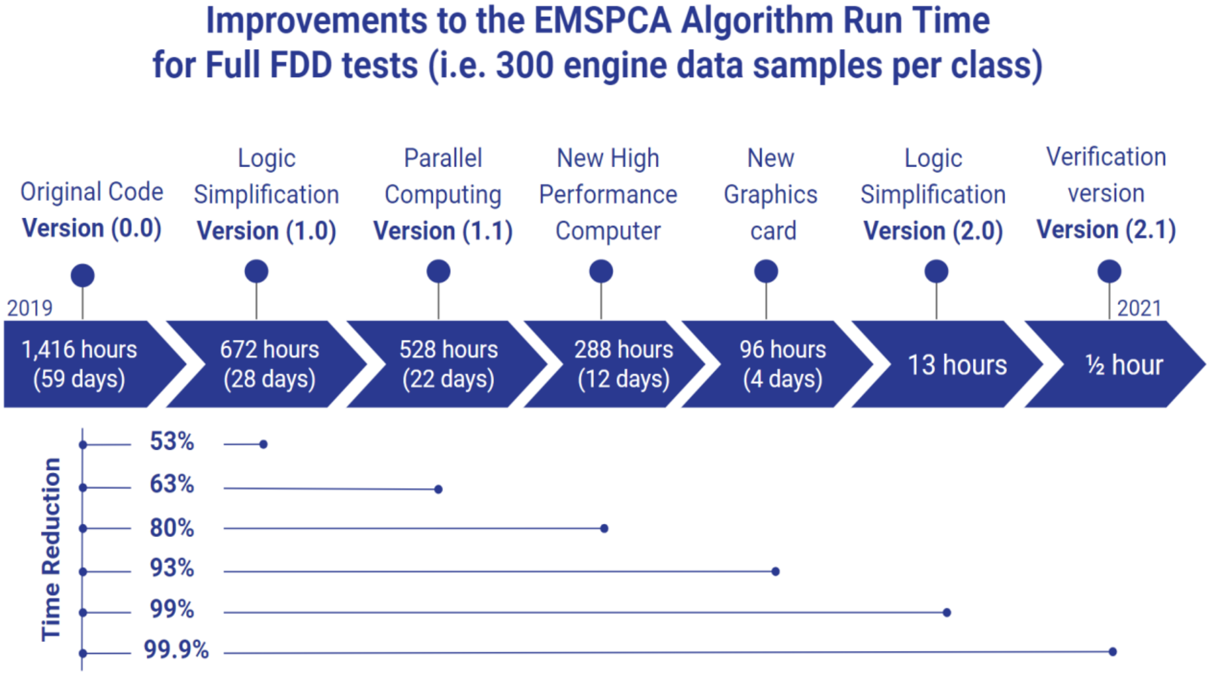


*Figure 6.5 EMSPCA run time improvements*

## *6.2 Spark Plug Degradation Detection Using Deep Learning (DL)*

This model consists of simple feature extraction algorithms along with advanced Deep Learning algorithms. The feature extraction algorithms convert raw data into sequential features and heat maps that feed into Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN), respectively. The mission of deep neural networks is to find similarities within features of the same class and mapping them accordingly. The same 3 fault conditions, see Figure 6.6, which were used to train the PRT classifier were also used to train the deep learning models.

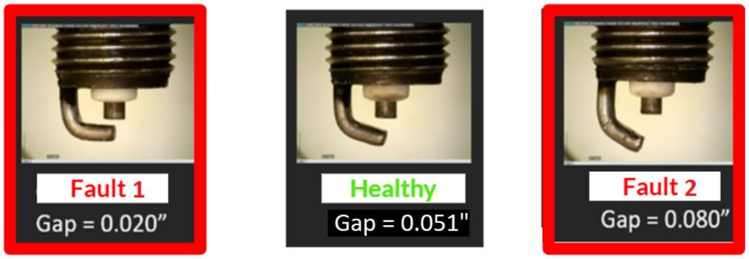


***Figure 6.6 Spark plug gaps***

The engine was operated to collect data through existing sensors: 4 knock sensors and 1 optical encoder. Each batch of data consists of 200 engine cycles containing 1.4M data points each. Figure 6.7 shows an example of 1 batch of data collected through a knock sensor.

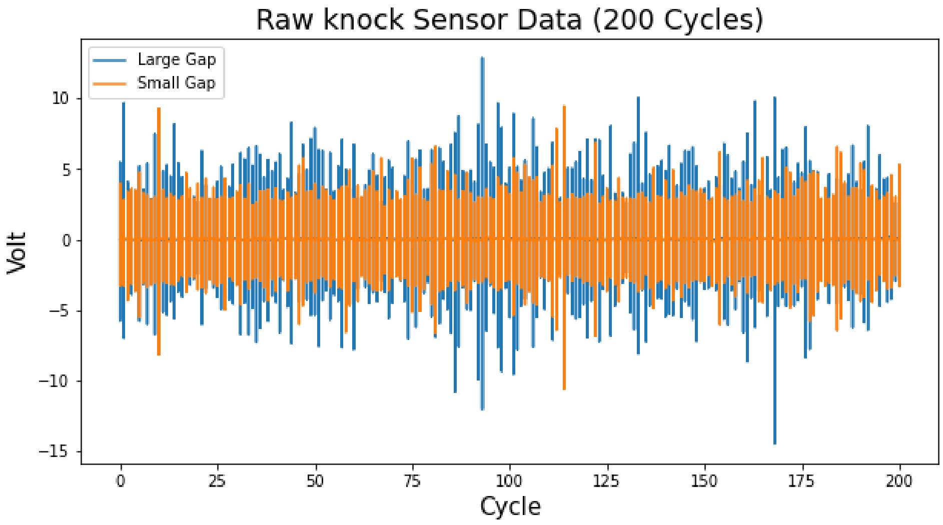


*Figure 6.7 Raw knock sensor data (200 engine cycle)*

The signal is extremely noisy which would be very challenging for the neural network to interpret. Thus, instead of using every set of data (i.e., 200 engine cycles) as a training sample, each set of data was split into 200 separate engine cycles, each is a training sample. Figure 6.8 shows an example of 1 engine cycle of a knock sensor containing 7,200 data points, which represents one training sample.

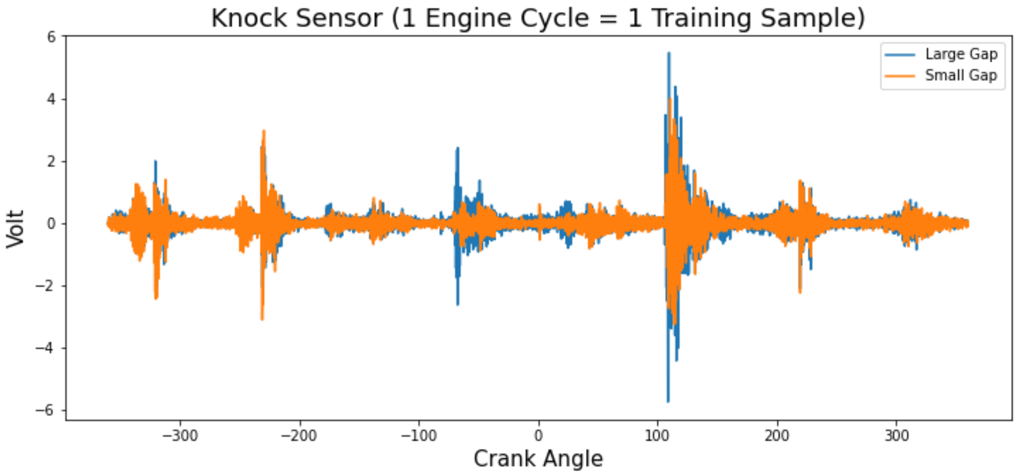


*Figure 6.8 Raw knock sensor data*

The sensory data were then transformed to two forms of features as a preparation for 2 different Deep Learning algorithms:

i. Sequential features.

ii. Heat maps.

Sequential features maintained the time-series information in the data, which made them a good match for a deep Long Short-Term Memory Recurrent Neural Network (LSTM-RNN). Heat maps were pre-processed through Mel-frequency Cepstral Coefficients (MFCC). MFCC heatmaps are 2-dimensional images which can be a good match for a deep Convolutional Neural Network (CNN). Thus, 2 models were developed as shown in Figure 6.9, namely Models S2 and S3 (see models in section 1.5):

1. Sequential features + LSTM-RNN.
2. MFCC heatmaps + CNN.

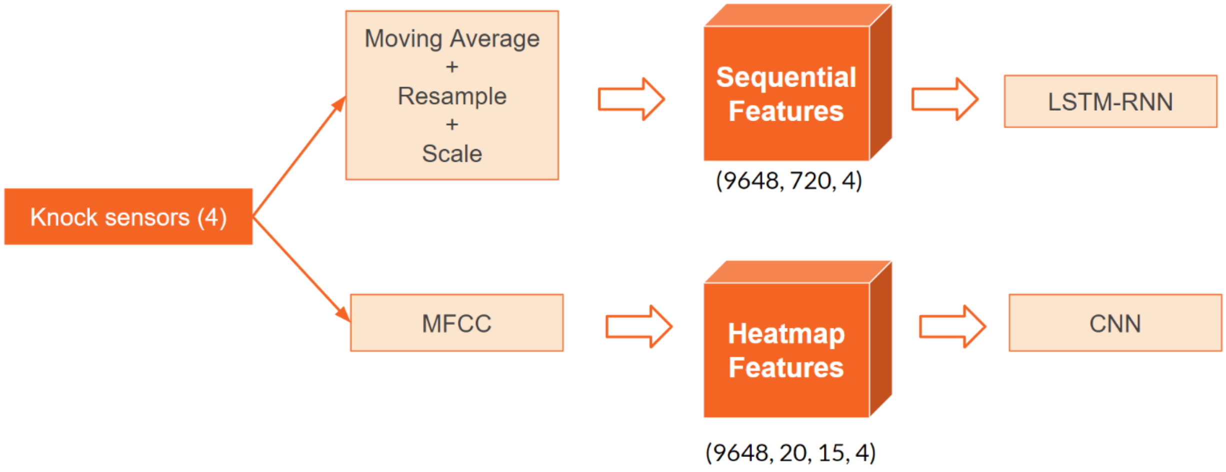


*Figure 6.9 DNN features Diagram*

### *6.2.1 Spark Plug Degradation Detection Using LSTM-RNNs*

The sequential features were created by applying a moving average to the raw signal. Every training sample contained one engine cycle of 7200 data points. Each sample was averaged with a window size 10 into 720 points to denoise the signal which made it simpler for the neural network to distinguish. This means that every 10 points were average into 1 point throughout the signal. The following equation shows the moving average procedure, where *n* is the number of data points.

$$\frac{a_1+a_2+a_3+\cdots+a_n}{n} \qquad \text{eq. (6.1)}$$

This helped the network focus on the behavior of the signal rather than looking at every single point individually. The window size of the moving average is adjustable and can be changed upon demand. Figure 6.10 shows a moving averaged knock sensor signal of a single engine cycle.

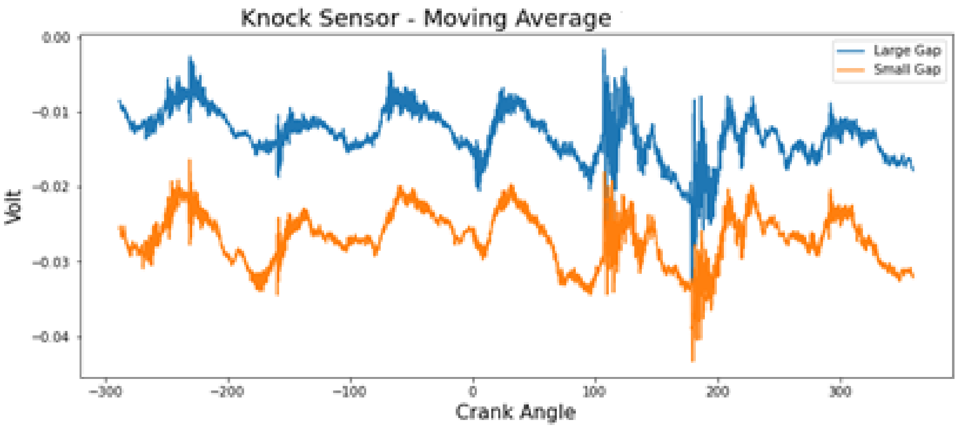


*Figure 6.10 Moving average*

The same denoising methods were dynamically used for other sensors when needed to generate sequential features. They were used as an additional hyper parameter to improve the accuracy of the deep learning models. All the featured signals were then normalized between 0 and 1 and fed into a Long Short-Term Memory Recurrent Neural Network (LSTM-RNN).

### 6.2.2 Spark Plug Degradation Detection Using CNNs

The second version of the sensory data, namely heatmaps, was created by applying *Mel-Frequency Cepstral Coefficients (MFCC)*. MFCC is often used in machine learning experiments involving audio or vibration signals. They are the coefficients that form the envelope of the time power spectrum (a.k.a. Mel-frequency Cepstrum) which is representative of the source of vibration or audio. Generally, the first 15 coefficients (i.e., lower dimensions) of MFCC are taken as features as they represent the envelope of spectra, while higher dimensions which express the spectral details are discarded. The following steps are followed to construct Mel-frequency Cepstral Coefficients:

(1) The raw signal is framed into short frames.

(2) For each frame, the periodogram estimate of the power spectrum is calculated through Fast Fourier Transform (FFT).

(3) The Mel filter bank is applied to the power spectra, and the energy is summed in each filter.

(4) The logarithm of all filter bank energies is taken.

(5) The Discrete Cosine Transform (DCT) of the log filter bank energies is applied.

(6) Only DCT coefficients 1-15 are kept, while the rest are discarded.

The raw signal, referred to as s(n) is framed into short frames creating several samples (n). Once it is framed, it is called $s_i(n)$ where *i* ranges over the number of frames. When the Fast Fourier Transform (FFT) is calculated, the resultant is $s_i(k)$ where *i* denotes the frame number corresponding to the crank-angle-domain frame. $P_i(k)$ is then the power spectrum of frame $i$. To take the Discrete Fourier Transform (implemented though a FFT implementation) of the frame, the following equation is performed, where *h(n)* is an *N* sample long analysis window, and *K* is the DFT length:

$$S_i(k) = \sum_{n=1}^{N} s_i(n)h(n)e^{-j2\pi kn/N} \qquad 1 \leq k \leq K$$

eq. (6.2)

The periodogram-based power spectral estimate for the speech frame $s_i(n)$ is calculated by:

$$P_i(k) = \frac{1}{N}|S_i(k)|^2$$

eq. (6.3)

where the square of the absolute value of the complex Fourier transform is taken. The Mel-spaced filter bank is then computed, which is a set of triangular filters that are applied to the periodogram power spectral estimate. Each vector is mostly zeros but is non-zero for a certain section of the spectrum. To calculate filter bank energies, each filter bank is multiplied with the power spectrum, then the coefficients are summed. Figure 6.11 shows an example of a filter bank.

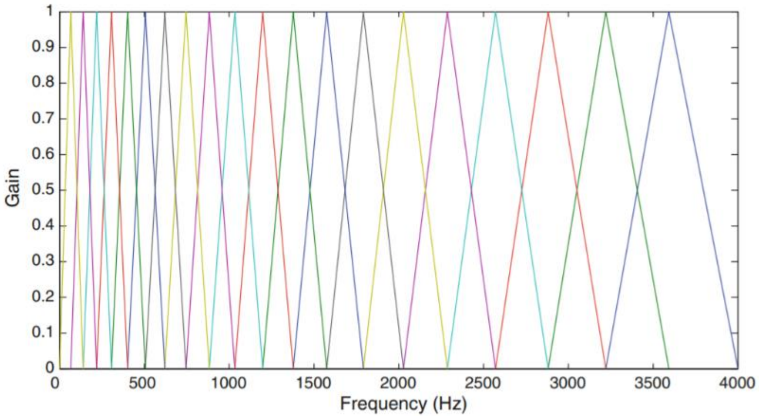


***Figure 6.11 Filter bank example***

The logarithm of each of the energies is then taken, followed by the Discrete Cosine Transform (DCT) of the log filter bank energies where cepstral coefficients are constructed. Only the first 15 of the resulting features, namely, Mel Frequency Cepstral Coefficients, are kept. Figure 6.12 shows the block diagram of the MFCC construction process.

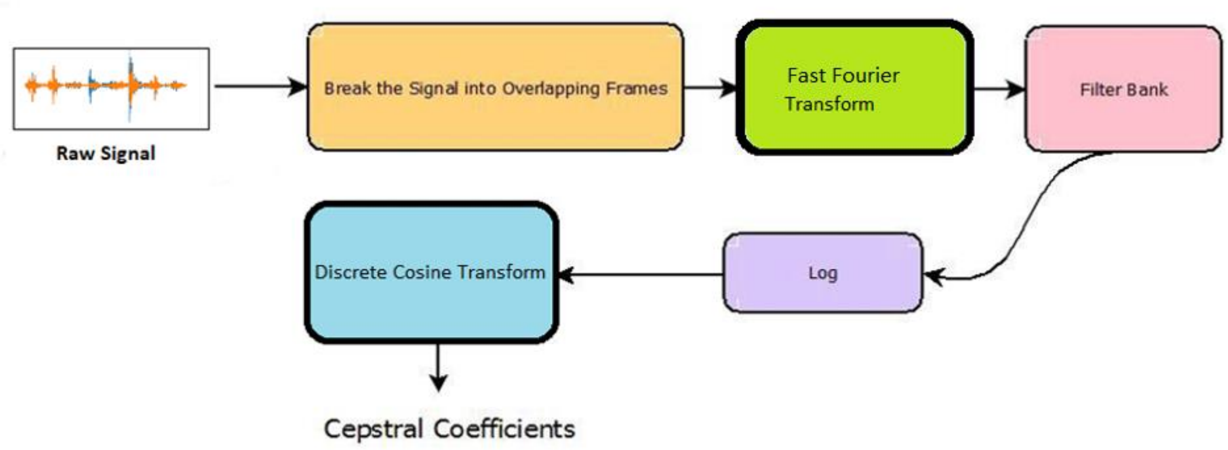


*Figure 6.12 MFCC*

The resulting MFCC heat maps represent the original data through 2-dimensional image-based features which a deep Convolutional Neural Network (CNN) can classify. This strategy allowed the CNN to perform time-series classification of spark plug data by converting knock sensor signals into images. Figure 6.13 is an example of a training sample which was processed into MFCC heatmaps. The heatmaps show different patterns in small and large spark plug gaps. All MFCC heatmaps were then fed into a CNN for classification.

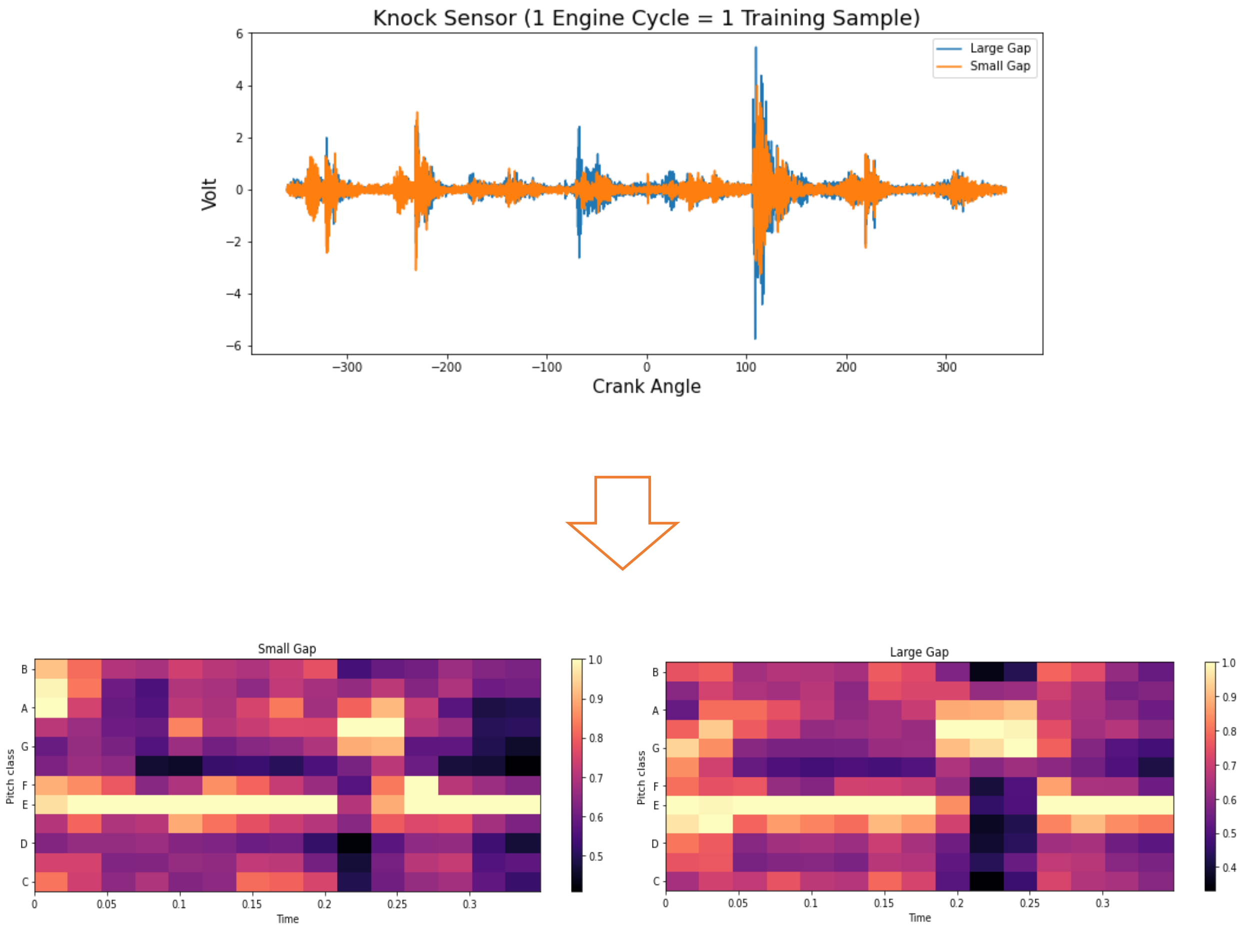


*Figure (6.13) MFCC heat maps example*

# Chapter 7

# Advanced Pre-processing Techniques for Deep Learning (DL) Algorithms

# Chapter 7 – Advanced Pre-processing Techniques for Deep Learning (DL) Algorithms

As discussed in chapter 4, the collected data were split into 2 categories: regular data and variant data (see section 5.1). An invariant Deep Learning (DL) model was required to detect and diagnose variant spark plug data. The Convolutional Neural Network (CNN) was selected as an invariant DL model. However, CNN works best with image classification. In this chapter, a novel pre-processing method is described to convert fault signatures into images to enable CNN classification.

## 7.1 Image Pre-processing of Fault Signatures

The proposed method is one of the novel contributions in this research. The method converts EMSPCA fault signatures into a mosaic-like map which graphically represents the same features of the fault signatures to the Convolutional Neural Network.

As described earlier, a fault signature consists of several bins. These bins represent frequency components which were generated by the wavelet packet transform (WPT) and principal component analysis (PCA) conducted in the EMSPCA process. The components represent the variance of each segment in the test sample from its corresponding segment of an original healthy baseline. Figure 7.1 shows an example of a fault signature of a knock sensor signal of a faulty spark plug with a gap size of 0.020”.

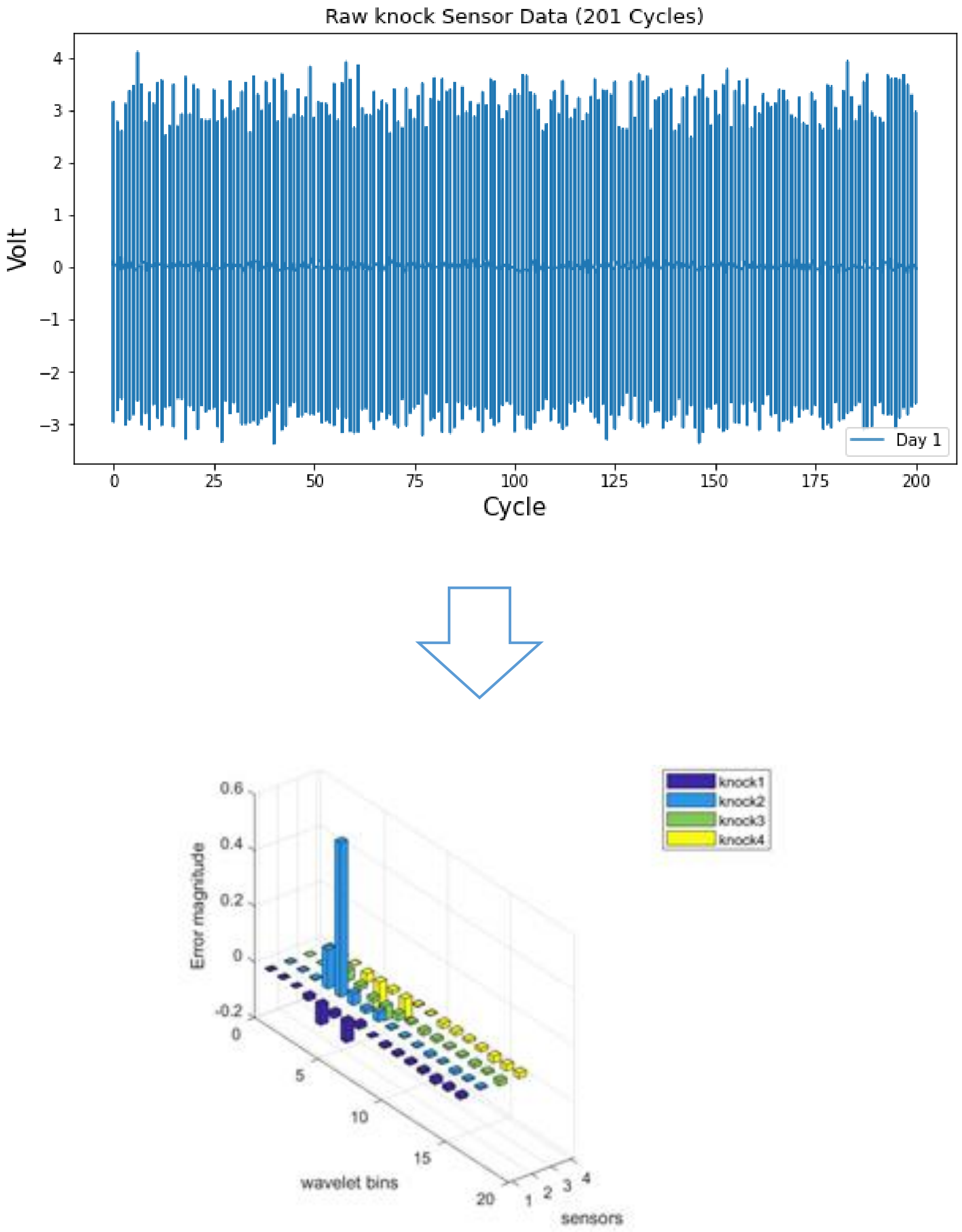
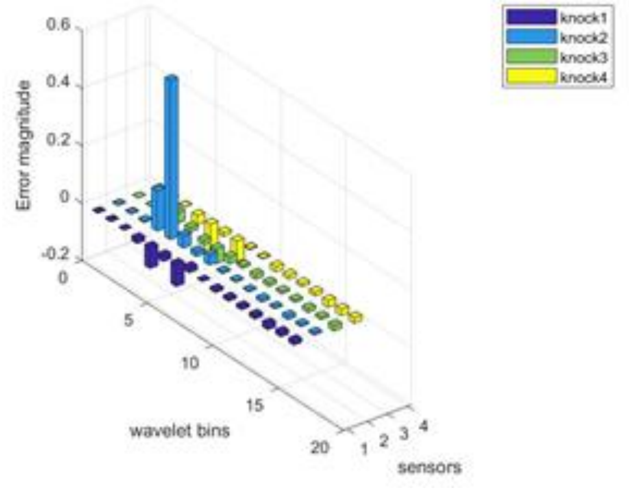
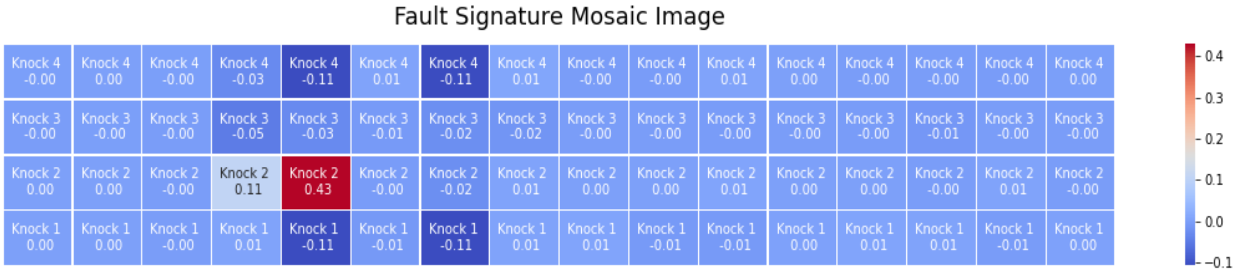


**Figure 7.1 Spark plug fault signature**

The fault signature shows a bar chart which contains four rows of 16 bins. The four rows represent the four engine built-in knock sensors. The dark blue row belongs to knock sensor 1, the light blue belongs to knock sensor 2, the green belongs to knock sensor 3 and the yellow belongs to knock sensor 4. The number of bins in each row depends on the level of wavelet used in the wavelet packet transform process which was conducted in the EMSPCA algorithm (see section 2.2.3). The number of bins is equal to 2 ^ (wavelet level). For example, for a fault signature of wavelet level 4 like the one in the figure above, the number of bins will be 2 ^ 4 = 16 bins. The amplitude of each bin represents the error magnitude compared to healthy baselines.

The proposed method was designed to deliver the features and information that the fault signature carries to the Convolutional Neural Network (CNN) in the form of an image. The image is a top view map of the fault signature. The number of rows in the map is equal to the number of rows (i.e., sensors) in the fault signature. The number of columns in the map is equal to the columns in the fault signature (i.e., number of frequency bins per sensor). The total number of tiles in the map is equal to the total number of bins in the fault signature bar chart. The color intensity of each tile represents the amplitude (i.e., error magnitude) of its corresponding frequency bin. The final product of this method is a mosaic panel which carries fault signature patterns in a form of an image as an input to the Convolutional Neural Network (CNN). Figure 7.2 shows an example of the mosaic image which was generated for the same fault signature in Figure 7.1.

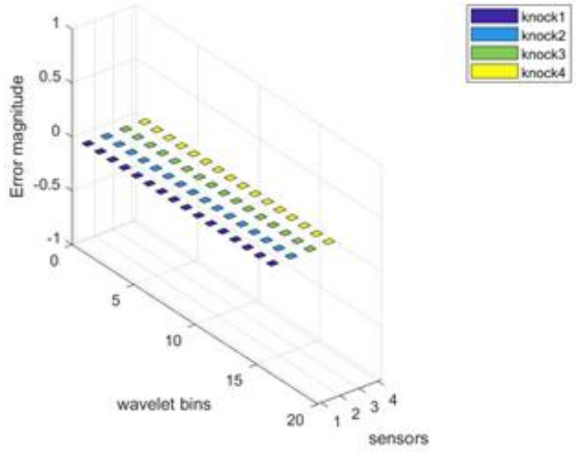


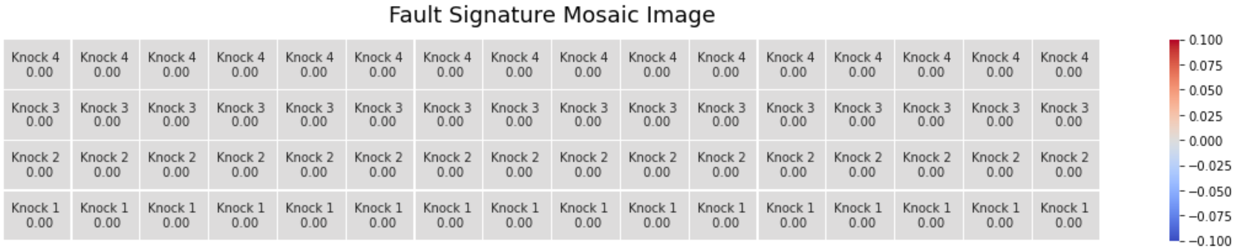


| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Knock 4<br>-0.00 | Knock 4<br>0.00 | Knock 4<br>-0.00 | Knock 4<br>-0.03 | Knock 4<br>-0.11 | Knock 4<br>0.01 | Knock 4<br>-0.11 | Knock 4<br>0.01 | Knock 4<br>-0.00 | Knock 4<br>-0.00 | Knock 4<br>0.01 | Knock 4<br>0.00 | Knock 4<br>-0.00 | Knock 4<br>-0.00 | Knock 4<br>-0.00 | Knock 4<br>0.00 |
| Knock 3<br>-0.00 | Knock 3<br>-0.00 | Knock 3<br>-0.00 | Knock 3<br>-0.05 | Knock 3<br>-0.03 | Knock 3<br>-0.01 | Knock 3<br>-0.02 | Knock 3<br>-0.02 | Knock 3<br>-0.00 | Knock 3<br>-0.00 | Knock 3<br>-0.00 | Knock 3<br>-0.00 | Knock 3<br>-0.00 | Knock 3<br>-0.01 | Knock 3<br>-0.00 | Knock 3<br>-0.00 |
| Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>-0.00 | Knock 2<br>0.11 | Knock 2<br>0.43 | Knock 2<br>-0.00 | Knock 2<br>-0.02 | Knock 2<br>0.01 | Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>0.01 | Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>-0.00 | Knock 2<br>0.01 | Knock 2<br>-0.00 |
| Knock 1<br>0.00 | Knock 1<br>0.00 | Knock 1<br>-0.00 | Knock 1<br>0.01 | Knock 1<br>-0.11 | Knock 1<br>-0.01 | Knock 1<br>-0.11 | Knock 1<br>0.01 | Knock 1<br>0.01 | Knock 1<br>-0.01 | Knock 1<br>-0.01 | Knock 1<br>0.00 | Knock 1<br>0.01 | Knock 1<br>0.01 | Knock 1<br>-0.01 | Knock 1<br>0.00 |

*Figure 7.2 Mosaic image of a 0.020" spark plug*

Figure 7.3 shows a fault signature of a healthy sample. When compared to a healthy baseline, a healthy test sample has an almost zero variance. Thus, the fault signature shows a flat pattern, which means there is the error magnitudes of all bins are almost zero.

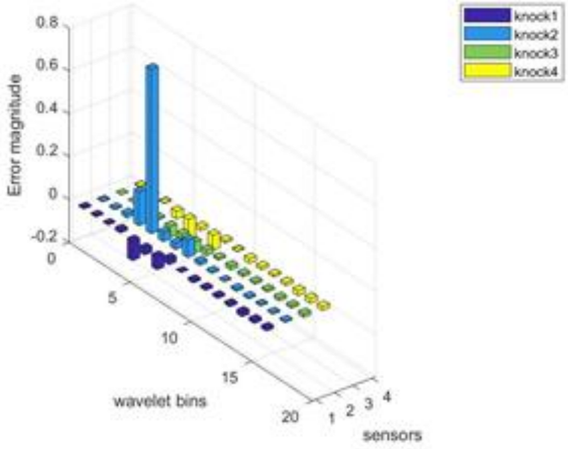


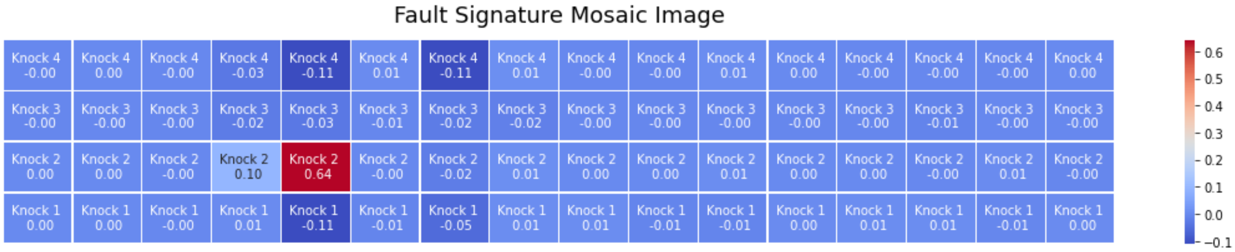


| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Knock 4<br>0.00 | Knock 4<br>0.00 | Knock 4<br>0.00 | Knock 4<br>0.00 | Knock 4<br>0.00 | Knock 4<br>0.00 | Knock 4<br>0.00 | Knock 4<br>0.00 | Knock 4<br>0.00 | Knock 4<br>0.00 | Knock 4<br>0.00 | Knock 4<br>0.00 | Knock 4<br>0.00 | Knock 4<br>0.00 | Knock 4<br>0.00 | Knock 4<br>0.00 |
| Knock 3<br>0.00 | Knock 3<br>0.00 | Knock 3<br>0.00 | Knock 3<br>0.00 | Knock 3<br>0.00 | Knock 3<br>0.00 | Knock 3<br>0.00 | Knock 3<br>0.00 | Knock 3<br>0.00 | Knock 3<br>0.00 | Knock 3<br>0.00 | Knock 3<br>0.00 | Knock 3<br>0.00 | Knock 3<br>0.00 | Knock 3<br>0.00 | Knock 3<br>0.00 |
| Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>0.00 |
| Knock 1<br>0.00 | Knock 1<br>0.00 | Knock 1<br>0.00 | Knock 1<br>0.00 | Knock 1<br>0.00 | Knock 1<br>0.00 | Knock 1<br>0.00 | Knock 1<br>0.00 | Knock 1<br>0.00 | Knock 1<br>0.00 | Knock 1<br>0.00 | Knock 1<br>0.00 | Knock 1<br>0.00 | Knock 1<br>0.00 | Knock 1<br>0.00 | Knock 1<br>0.00 |

***Figure 7.3 Mosaic image of a healthy gap (0.050”)***

Figure 7.4 represents the mosaic image generated for a fault signature of a large gap size. Since both fault conditions, small (i.e., 0.020”) and large (i.e., 0.080”) gaps,  lead to lower engine performance caused by engine vibrations. The patterns in the fault signature of both fault conditions are relatively close, compared to the healthy signature, but the amplitude of the most significant bin of the larger gap is higher than the small gap. Thus, the mosaic image of both fault conditions show similar patterns with different color intensities.

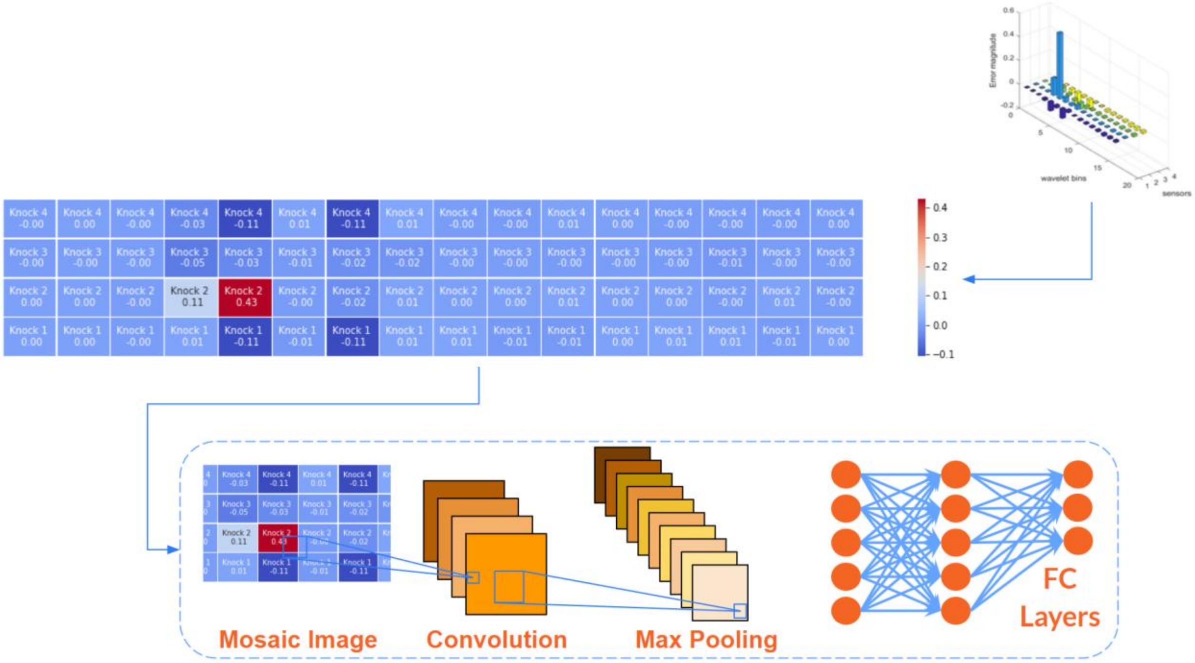




| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Knock 4<br>-0.00 | Knock 4<br>0.00 | Knock 4<br>-0.00 | Knock 4<br>-0.03 | Knock 4<br>-0.11 | Knock 4<br>0.01 | Knock 4<br>-0.11 | Knock 4<br>0.01 | Knock 4<br>-0.00 | Knock 4<br>-0.00 | Knock 4<br>0.01 | Knock 4<br>0.00 | Knock 4<br>-0.00 | Knock 4<br>-0.00 | Knock 4<br>-0.00 | Knock 4<br>0.00 |
| Knock 3<br>-0.00 | Knock 3<br>-0.00 | Knock 3<br>-0.00 | Knock 3<br>-0.02 | Knock 3<br>-0.03 | Knock 3<br>-0.01 | Knock 3<br>-0.02 | Knock 3<br>-0.02 | Knock 3<br>-0.00 | Knock 3<br>-0.00 | Knock 3<br>-0.00 | Knock 3<br>-0.00 | Knock 3<br>-0.00 | Knock 3<br>-0.01 | Knock 3<br>-0.00 | Knock 3<br>-0.00 |
| Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>-0.00 | Knock 2<br>0.10 | Knock 2<br>0.64 | Knock 2<br>-0.00 | Knock 2<br>-0.02 | Knock 2<br>0.01 | Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>0.01 | Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>-0.00 | Knock 2<br>0.01 | Knock 2<br>-0.00 |
| Knock 1<br>0.00 | Knock 1<br>0.00 | Knock 1<br>-0.00 | Knock 1<br>0.01 | Knock 1<br>-0.11 | Knock 1<br>-0.01 | Knock 1<br>-0.05 | Knock 1<br>0.01 | Knock 1<br>0.01 | Knock 1<br>-0.01 | Knock 1<br>-0.01 | Knock 1<br>0.00 | Knock 1<br>0.01 | Knock 1<br>0.01 | Knock 1<br>-0.01 | Knock 1<br>0.00 |

***Figure 7.4 Fault signature of a large gap (0.080”)***

The mosaic method was able to reconstruct patterns from spark plug fault signatures in form of images. However, it was not easy for us humans to distinguish similar patterns using individual data samples. This is because of the randomness inside the engine cylinder during the combustion process may result in showing different patterns within the same fault condition. This explains how difficult the spark plug degradation problem was. The CNN model consisted of 4 hidden layers: 3 Convolutional layers and 1 Max Pooling layer. The output was taken to 1 Fully Connected (FC) layer which consisted of 3 neurons: healthy, fault 1 and fault 2. A Relu activation function was used in all hidden layers, while a Softmax function was used in the FC layer. An Adam optimizer was used along with a learning rate of 0.001 in this network.



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Knock 4<br>-0.00 | Knock 4<br>0.00 | Knock 4<br>-0.00 | Knock 4<br>-0.03 | Knock 4<br>-0.11 | Knock 4<br>0.01 | Knock 4<br>-0.11 | Knock 4<br>0.01 | Knock 4<br>-0.00 | Knock 4<br>-0.00 | Knock 4<br>0.01 | Knock 4<br>0.00 | Knock 4<br>-0.00 | Knock 4<br>-0.00 | Knock 4<br>-0.00 | Knock 4<br>0.00 |
| Knock 3<br>-0.00 | Knock 3<br>-0.00 | Knock 3<br>-0.00 | Knock 3<br>-0.05 | Knock 3<br>-0.03 | Knock 3<br>-0.01 | Knock 3<br>-0.02 | Knock 3<br>-0.02 | Knock 3<br>-0.00 | Knock 3<br>-0.00 | Knock 3<br>-0.00 | Knock 3<br>-0.00 | Knock 3<br>-0.00 | Knock 3<br>-0.01 | Knock 3<br>-0.00 | Knock 3<br>-0.00 |
| Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>-0.00 | Knock 2<br>0.11 | Knock 2<br>0.43 | Knock 2<br>-0.00 | Knock 2<br>-0.02 | Knock 2<br>0.01 | Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>0.01 | Knock 2<br>0.00 | Knock 2<br>0.00 | Knock 2<br>-0.00 | Knock 2<br>0.01 | Knock 2<br>-0.00 |
| Knock 1<br>0.00 | Knock 1<br>0.00 | Knock 1<br>-0.00 | Knock 1<br>0.01 | Knock 1<br>-0.11 | Knock 1<br>-0.01 | Knock 1<br>-0.11 | Knock 1<br>0.01 | Knock 1<br>0.01 | Knock 1<br>-0.01 | Knock 1<br>-0.01 | Knock 1<br>0.00 | Knock 1<br>0.01 | Knock 1<br>0.01 | Knock 1<br>-0.01 | Knock 1<br>0.00 |

***Figure 7.5 CNN analyzing mosaic images***

The mosaic images can be used as the input data to the Convolutional Neural Network (CNN) as Figure 7.5 shows. However, the performance would not be at its best. In data science, it is very important to visualize data before using it for training or testing. Visualization helps researchers figure out what pre-processing is needed in order to emphasize patterns. However, there is no pre-processing technique that solves all data issues. Every problem contains a different kind of data which requires unique strategies of feature engineering. Sometimes, data can be too noisy and needs filtration. In other cases, it may need averaging, normalization, or smoothing techniques.

## *7.2 Normalization*

To properly see similarities within the 3 fault conditions of the spark plug data, the entire dataset was plotted, as shown in Figure 7.6. The graph shows 1860x16 plot. The x-axis represents the number of data samples of the training dataset while the y-axis represents the number of frequency bins represented by mosaic images. The graph clearly shows that most peaks are concentrated in a certain range, which may prevent the learning algorithm from being able to classify properly.

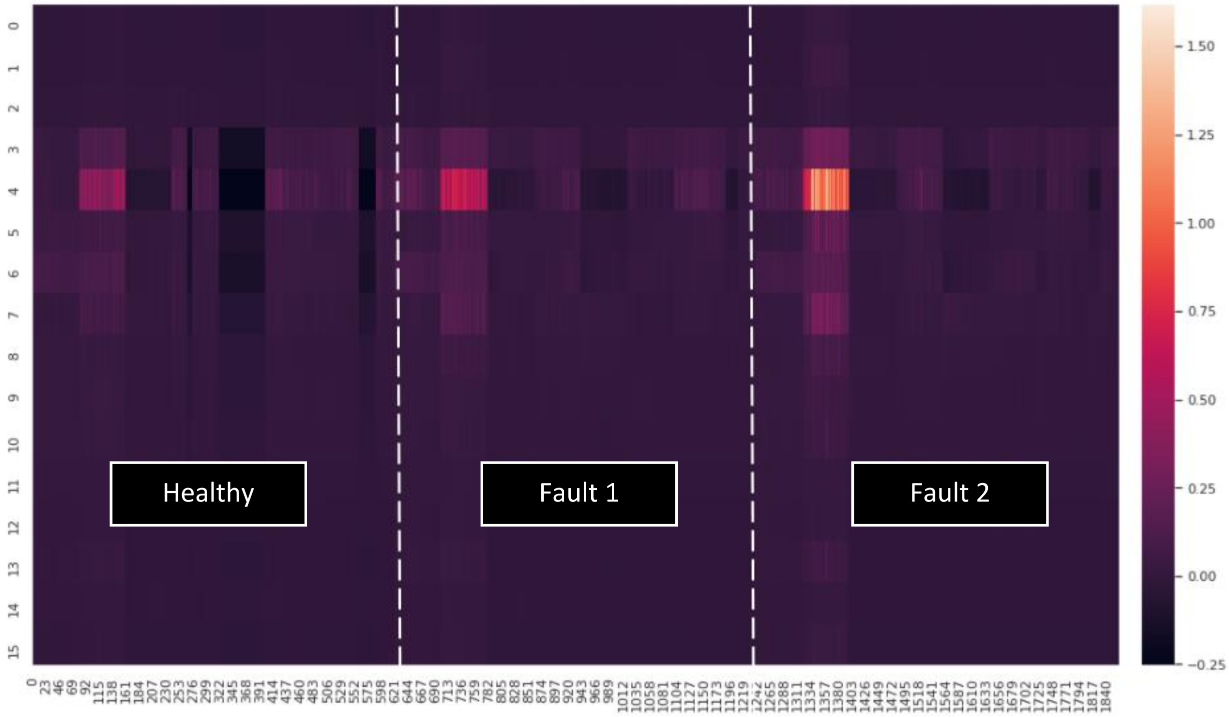


**Figure 7.6 Spark plug data before normalization**

Machine learning and Deep Learning algorithms were designed to find trends in the data by comparing features of data points, referred to as patterns. However, when features are on significantly different scales, the algorithm may not be able to distinguish patterns of different classes. Plotting data points of different scales on a single scale might result in a concentration and grouping of data points as shown in the figure above, although there are hidden differences. To solve this problem, the scale must be unified throughout the entire training dataset, which requires a normalization approach. In this method, a Min-Max normalizer was used to normalize the spark plug data.

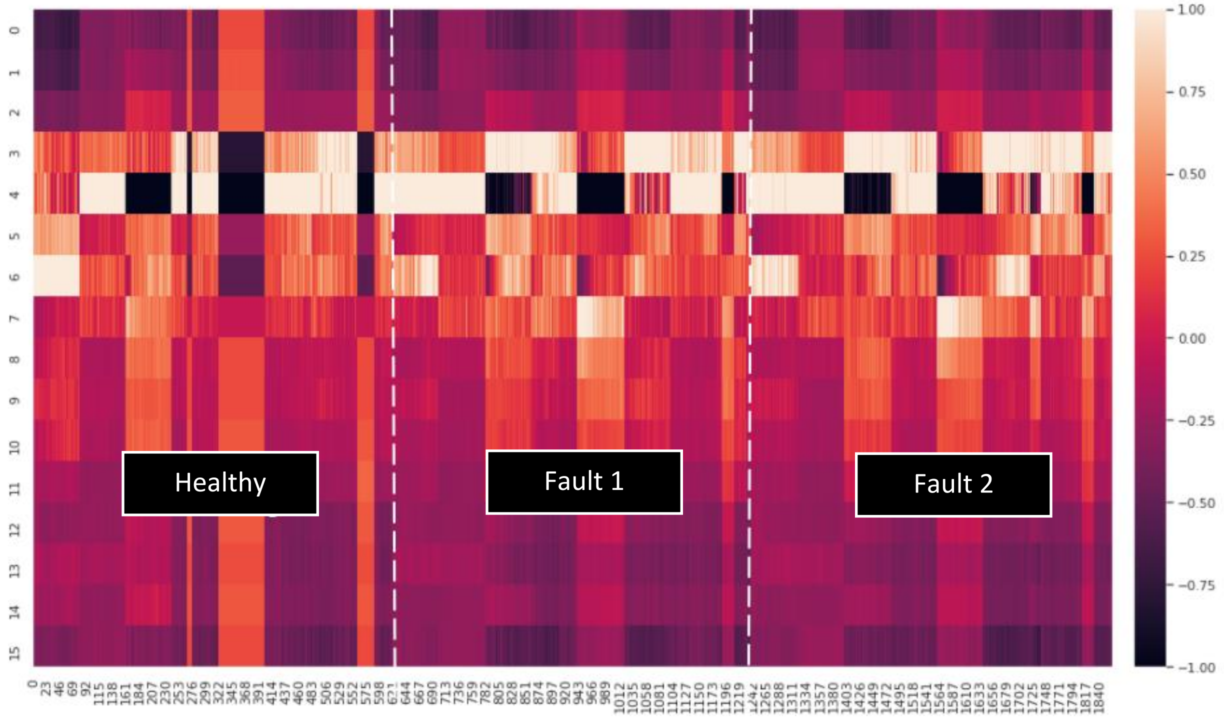


**Figure 7.7 Spark plug data after normalization**

The normalization step shown in Figure 7.7 clearly increased the granularity of the features within different classes and the difference between the data points can be visualized more easily.

## *7.3 Data Augmentation*

As shown in the previous figures, fault signatures typically consist of 16 frequency bins. Deep Learning algorithms normally require large amounts of data to achieve high classification accuracy. This includes both the number of data samples and the number of informative data points per sample. Having a small number of data points like in the fault signatures means more data samples are required. In the case of using all the available data samples without reaching the desired classification accuracy, data augmentation approaches are needed. Data augmentation can be conducted to increase either the size of each data sample (i.e., image), or the number of data samples, or sometimes both. However, data augmentation is only an attempt to increase the performance of the model, it does not always solve the problem if the output does not add valuable information for the model to learn from. In this method, a few data augmentation approaches were conducted.

### *7.3.1 Wavelet Levels*

As explained earlier in this chapter, the number of frequency bins of a fault signature depends on the wavelet level. Fault signatures which consist of 16 frequency bins were generated by applying wavelets level 4 to the raw signal. A higher wavelet level (i.e., level 6) was applied to the raw signal to generate fault signatures with more data points. Figure 7.8 shows an example of the difference between fault signatures generated under wavelet levels 4 and 6. Both signatures represent the same raw signals from 4 knock sensors. The shorter fault signature consists of 16x4 data points while the taller one consists of 64x4 data points.

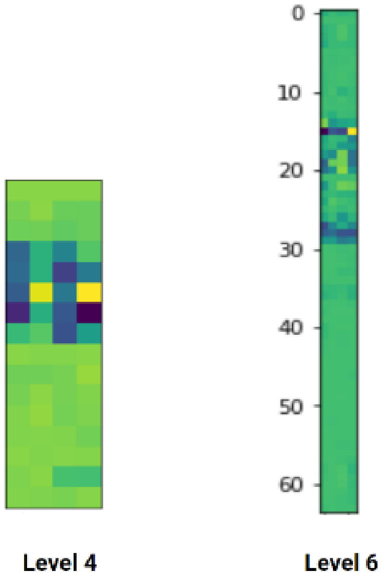


*Figure 7.8 Wavelet level 4 vs 6 fault signatures*

Higher wavelet levels could be achieved to generate bigger fault signatures. However, Wavelet Packet Transform (WPT) is a computationally heavy method. Thus, generating hundreds of fault signatures at higher wavelet levels (e.g., 8 and 10) was very time consuming. Also, bigger fault signatures would only help if the additional data points carried valuable information for the Deep Learning algorithm. To answer this question, different wavelet levels were tried, and it was observed that wavelet level 4 always provided the best results.

### *7.3.2 Image Size Augmentation*

Another way of data augmentation was to increase the size of mosaic images using common practical approaches. Although fault signatures carry information of 4 knock sensors, single sensors were also used individually to compare performances in all scenarios. The size of the fault signature dropped from 16x4 to 16x1 for single sensors as shown in Figure 7.9. In this case, image size augmentation was mandatory.

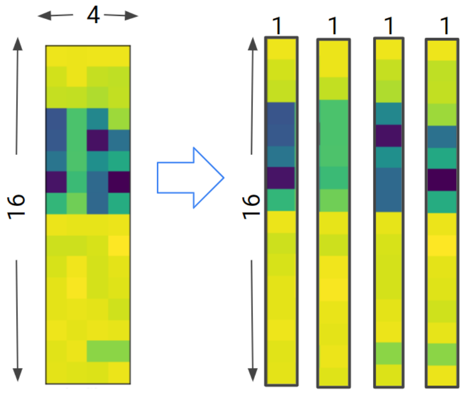


*Figure 7.9 Single sensor fault signature*

Increasing the size of singe-sensor mosaic images was an attempt to emphasize the patterns. This was done by repeating the pattern several times along the x-axis to create a bigger image. Figure 7.10 shows a horizontally augmented mosaic image of a single sensor.

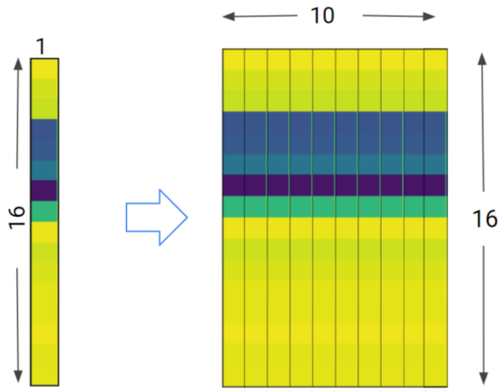


*Figure 7.10 Horizontally augmented single-sensor mosaic image*

Another way of augmenting the image was to repeat the pattern along y-axis on top of the x-axis. This has further increased the size of the image to emphasize the patterns by repetition. Figure 7.11 shows a horizontally and vertically augmented mosaic image of a single sensor.

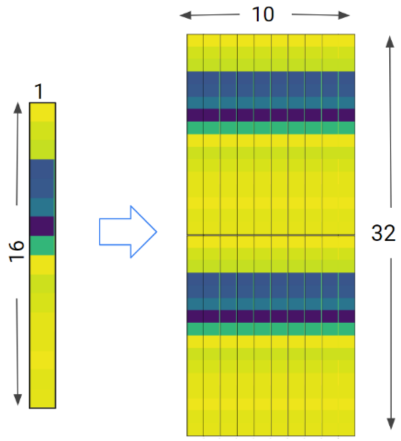


*Figure 7.11 Horizontally and vertically augmented single-sensor mosaic image*

### 7.3.3 Number of Samples Augmentation

As explained earlier in the chapter of experimental data and data collection, engine data was collected in form of data sets, each consisting of 200 engine cycles. In most experiments, these data sets were split into smaller portions of 20 engine cycles per training sample. This means that

10 training samples were constructed from each data set. Decreasing the number of engine cycles per training sample was another way to augment the number of data samples.

Several combinations (e.g., 10 cycles, 4 cycles and 1 cycle per sample) were tried to achieve the highest classification accuracy possible. Although the performance of the CNN model depended on the amount of information which each combination carries, it is not easy to expect the performance of a certain combination without trying. Less engine cycles per sample may result in less informative mosaic images. Also, loading each training sample with too much data can lead to overfitting or low classification accuracy. Figure 7.12, 7.13 and 7.14 show the difference between a single engine cycle per training sample and the one-cycle average of multiple engine cycles per sample for a 0.020", 0.050" and 0.080" respectively.

The figures show that a single engine cycle per training sample is much noisier than one-cycle averages of multiple engine cycles per sample. Noisy signals result in less informative fault signatures, which consequently means less informative mosaic images. However, using more engine cycles means a smaller number of training samples. Thus, a balanced decision had to be made. Filtering raw signals was also used as a pre-processing approach to eliminate the noise of training samples with low number of engine cycles. Eliminating noise discards redundant data points in noisy raw signals.

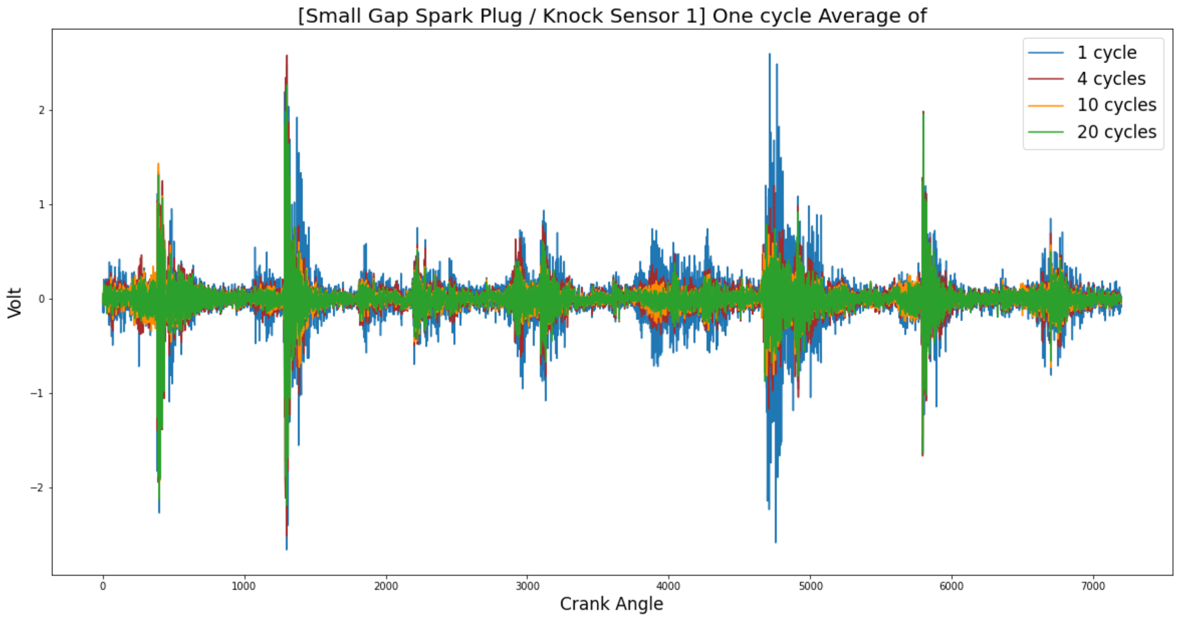


*Figure 7.12 Horizontally augmented single-sensor mosaic image*

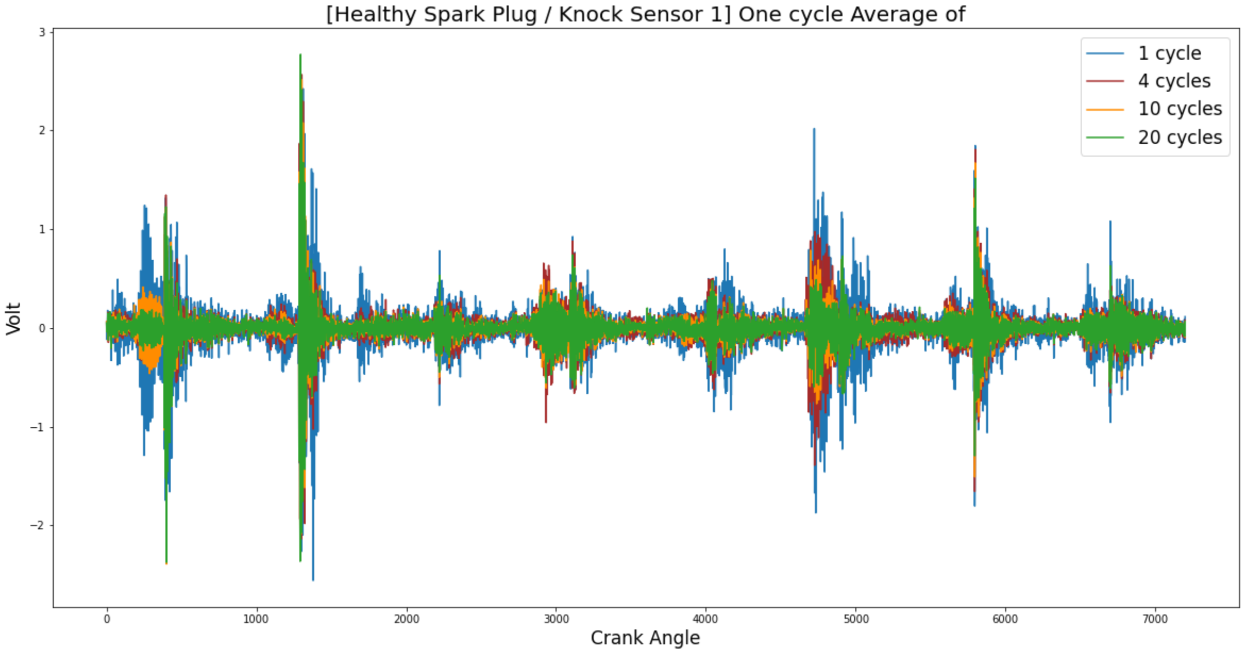


*Figure 7.13 Horizontally augmented single-sensor mosaic image*

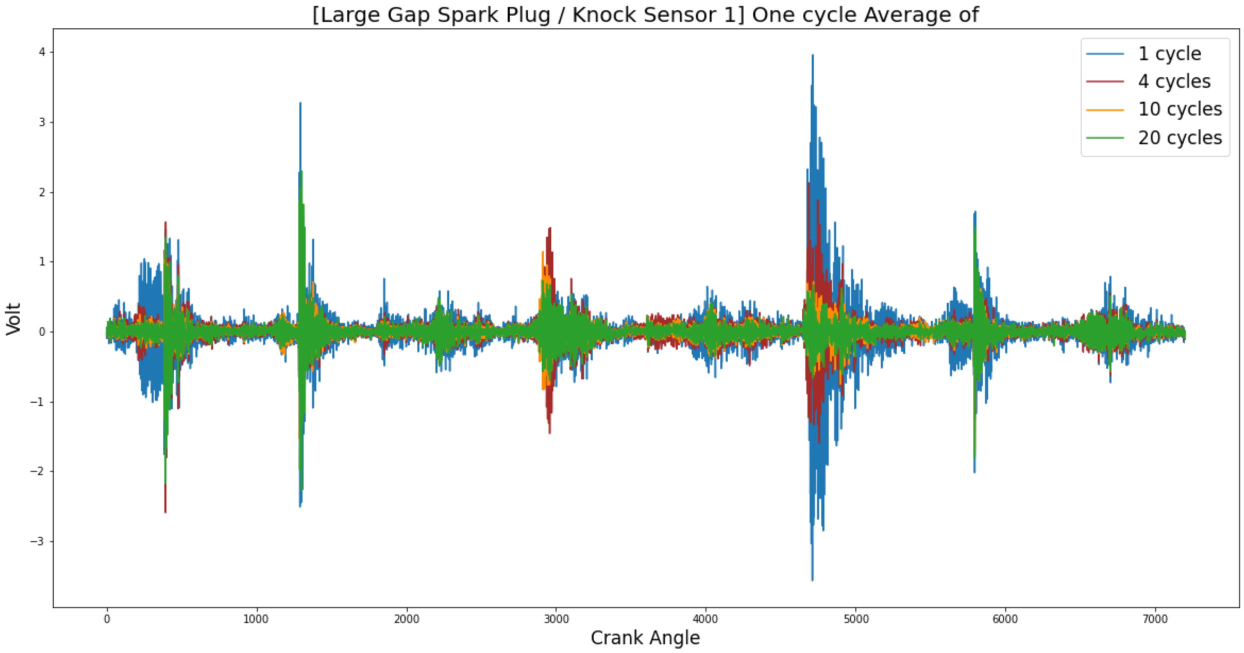


***Figure 7.14 Horizontally augmented single-sensor mosaic image***

### *7.3.4 Mosaic Image Workflow*

The workflow of this method can be summarized in the block diagram shown in Figure 7.15. The method started with importing both raw healthy and faulty sensory data. Signals were then normalized to unify varying scales of the data. Then the EMSPCA method (see section 2.2.3) was applied where fault signatures of both healthy and faulty data were generated. The next step is to graphically present fault signatures graphically to the Convolutional Neural Network (CNN) through mosaic images.

Since error magnitudes can significantly vary within different scales, the fault signatures were normalized again, which emphasized the granularity of the patterns. The mosaic images were then generated from the normalized fault signatures. Image augmentation approaches were applied to stretch the images because of the small size of the fault signatures. The output from the image augmentation process was the input to the convolutional neural network (see section 2.3.2).

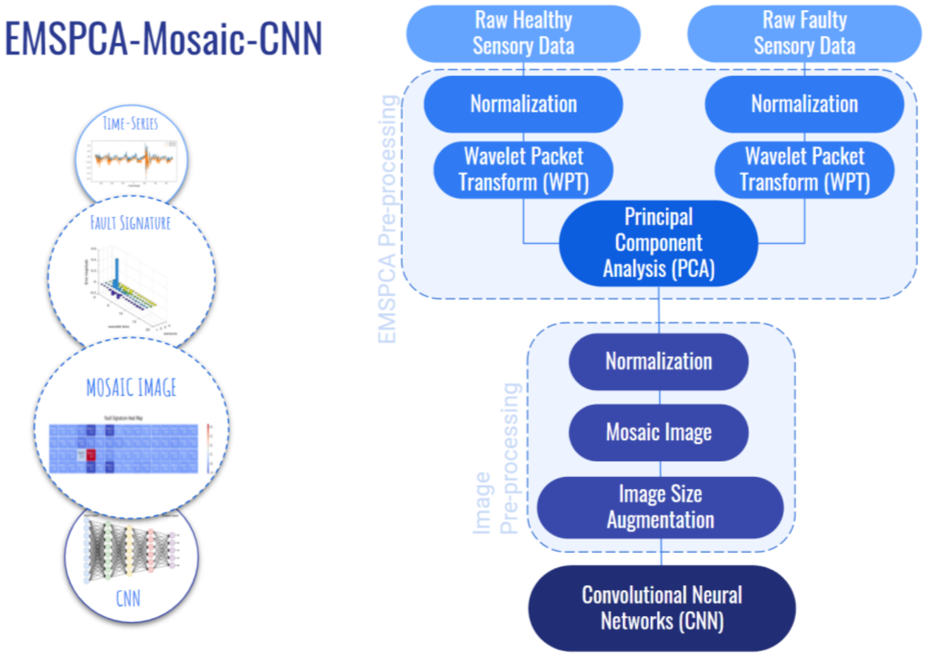


*Figure 7.15 Block diagram of EMSPCA-Mosaic-CNN*

# Chapter 8

# Internal Combustion Engine (ICE)

# FDD Results and Discussion

# *Chapter 8 – Internal Combustion Engine (ICE) FDD Results and Discussions*

In this chapter, the results of the six models which were built to detect spark plug (S1 to S4) and EGR valve (E1 and E2) degradation are discussed (see section 1.5). The structure and methodology of all the six Al-based models were discussed in chapter 6.

## *8.1 Results of Spark Plug Degradation Detection*

The spark plug degradation detection problem using standard engine sensors only was the most challenging problem in this research. Models S1 to S4 (see section 1.5) were developed to detect spark plug degradation. In Models S1 to S3, only 4 standard knock sensors of the Ford Coyote Engine (see section 5.1) were used. In Model S4, only a single optical encoder of the engine was used. Each model consisted of a pre-processing technique, along with a Machine Learning (ML) or Deep Learning (DL) classifier. The following results show the performance of each model.

### *8.1.1 Degradation Detection Using EMSPCA*

The theory and structure of the Extended Multi-Scale Principal Component Analysis (EMSPCA) were discussed earlier in this thesis (see section 2.2.3). The output from the EMSPCA method was fault signatures. Figures 8.1-8.3 show the average fault signatures of the 3 spark plug gaps, representing the 3 fault conditions:

a) Fault 1: 0.020”, incorrect spark plug setup.

b) Healthy: 0.040”, standard gap.

c) Fault 2: 0.080”, spark plug degradation.

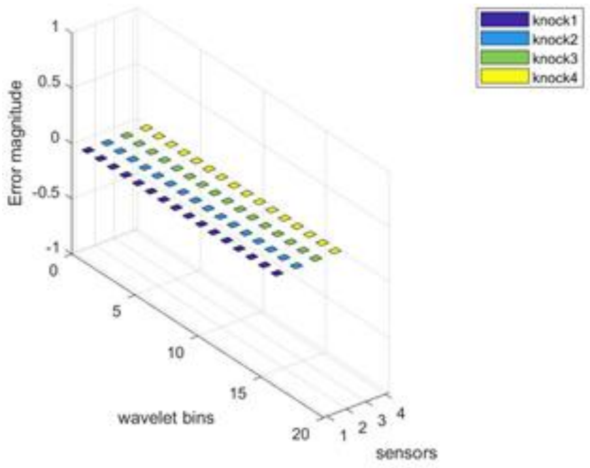


*Figure 8.1 Fault signature of a healthy gap (0.050”)*

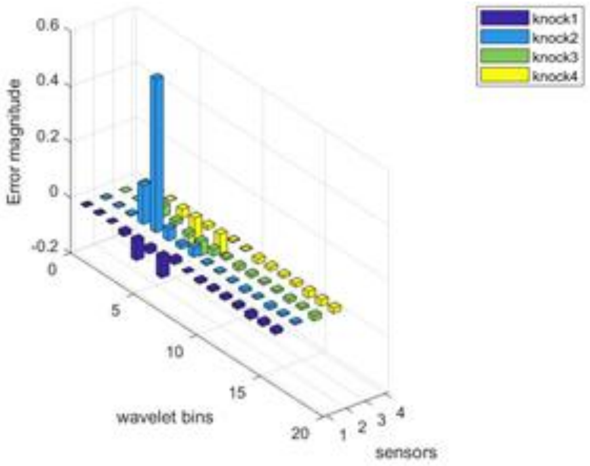


*Figure 8.2 Fault signature of a small gap (0.020”)*

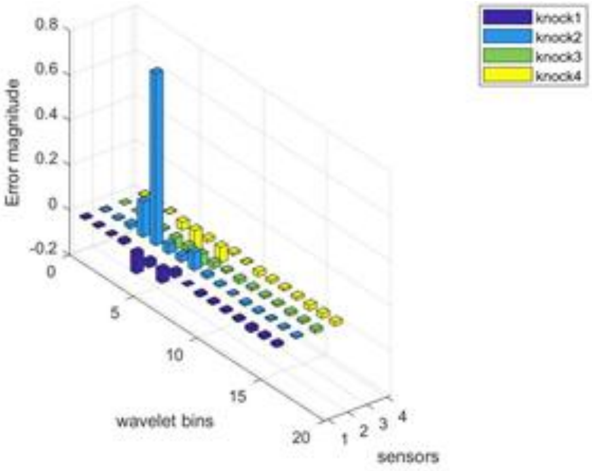


*Figure 8.3 Fault signature of a large gap (0.080”)*

The 3 fault signatures have distinct patterns in each of the 3 classes. These patterns were used as input to the Artificial Neural Network (ANN), representing the machine learning classifier. The output layer of the ANN consisted of 3 classes, representing 1 healthy and 2 fault conditions. The ML model consisted of a single input layer. The dimensions of the input layer were 4x16, which represented the number of sensors x the number of frequency bins. Three hidden layers were designed to calculate the feature weights. The output layer consists of 3 units per sample. Each unit provides a probability for the sample to belong to one of the 3 conditions.

The model was trained with 300 data samples for each of the 3 classes and was then tested using a set of unlabeled data to ensure high performance. The results were graphically presented using probability bar charts. A test sample that was predicted as category 1 (Healthy) with more than 50% probability would be considered as a healthy sample. A test sample that gets category 2 (XX gap) with more than 50% was a smaller gap. Similarly, category 3 (XX gap) with more than 50% goes for a large gap. Figures 8.4 – 8.6 show examples of predictions for 3 samples of class 1, class 2 and class 3, respectively.

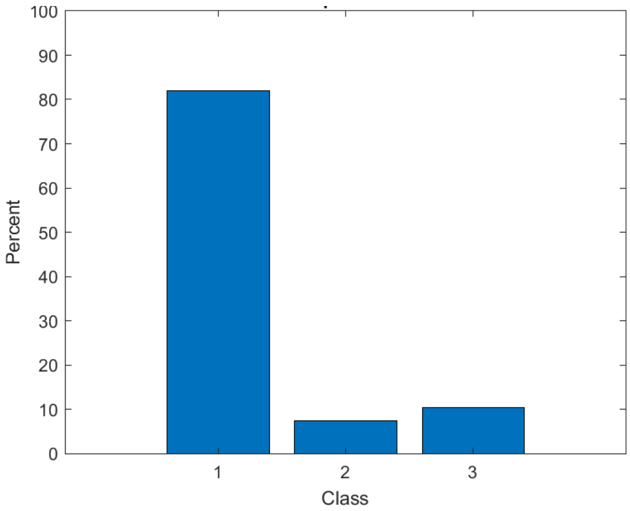


***Figure 8.4 EMSPCA results for spark plugs – class 1***

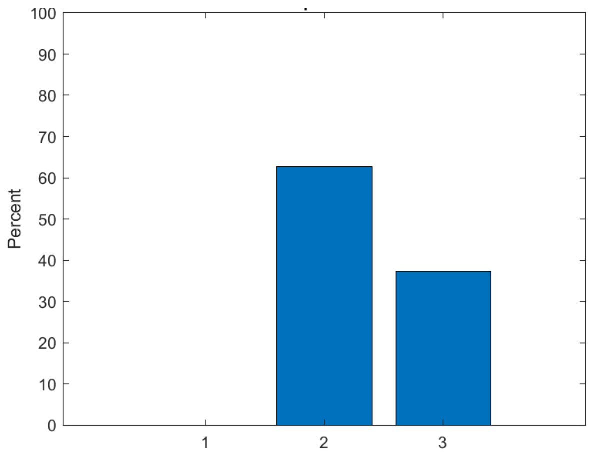


*Figure 8.5 EMSPCA results for spark plugs – class 2*

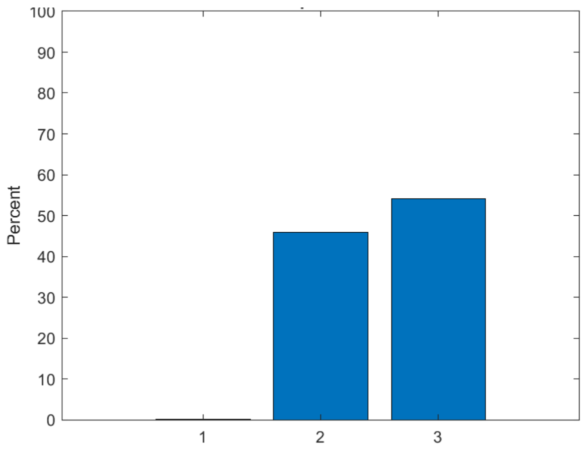
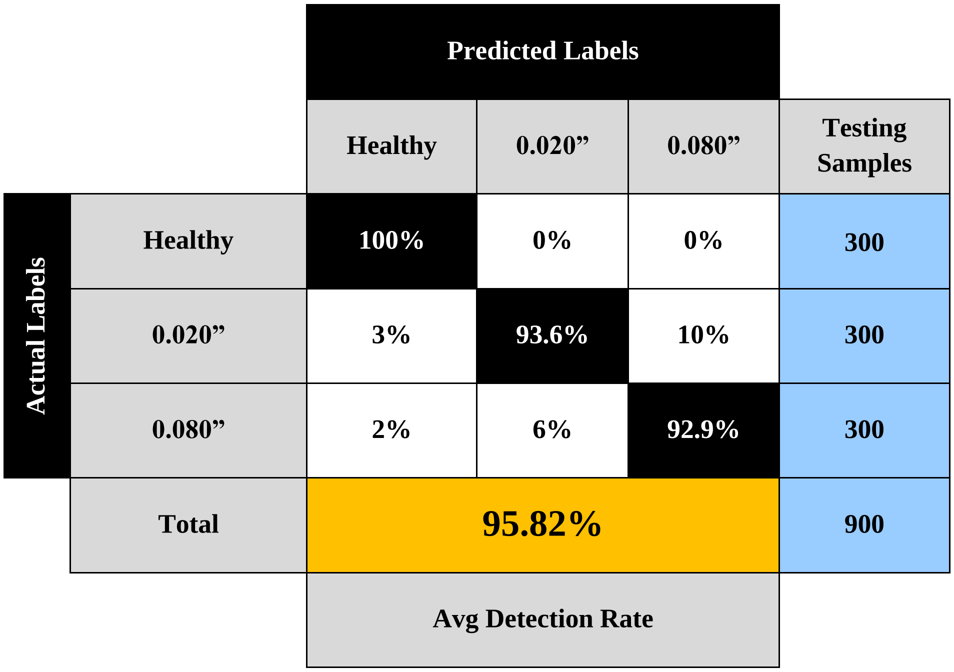


*Figure 8.6 EMSPCA results for spark plugs – class 3*

The total average detection rate represents the number of samples which were correctly predicted. For example, the charts above show 100% detection rate since all the predicted labels were equivalent to the actual labels. The algorithm was tested with data from different days and different times of the day to consider the reliability of the model. The model was able to detect and diagnose the degradation in 100% of the unseen test samples which were collected on the same day and the same morning when the training samples were collected. Another set of 100 test samples which were collected in the afternoon of the same day when the training samples were collected. All the test samples were detected and diagnosed correctly. Furthermore, the model was able to detect the degradation of 90 out of 100 samples which were collected on the 5th day. Figure 8.7 shows the summary confusion matrix of all the tests conducted to evaluate the performance of the EMSPCA model in detecting spark plug degradation on regular data of days 1, 2, 4 and 5 (see section 5.1).



| | | Predicted Labels | | | |
|---|---|---|---|---|---|
| | | Healthy | 0.020” | 0.080” | Testing Samples |
| Actual Labels | Healthy | 100% | 0% | 0% | 300 |
| | 0.020” | 3% | 93.6% | 10% | 300 |
| | 0.080” | 2% | 6% | 92.9% | 300 |
| | Total | 95.82% | | | 900 |
| | | Avg Detection Rate | | | |

*Figure 8.7 Test Confusion matrix of spark plug results using EMSPCA-ANN*

### *8.1.2 Degradation Detection Using Deep LSTM-RNN*

This section presents the results and discussions on the use of the deep Long Short-Term Memory Recurrent Neural Networks (LSTM-RNNs), a.k.a. Model S2 (see section 1.5). As mentioned in the previous chapters, all the sensory data were reduced to 720 data points through different feature extraction algorithms to ensure consistency of feature dimensions. Those 720 points were used as input to the first LSTM layer of the RNN model, then the weights were passed on to the hidden LSTM layers. The output from the hidden LSTM layers was taken to fully connected layers to connect all the information coming from each of the features. Figure 8.8 shows the structure of the LSTM-RNN model.

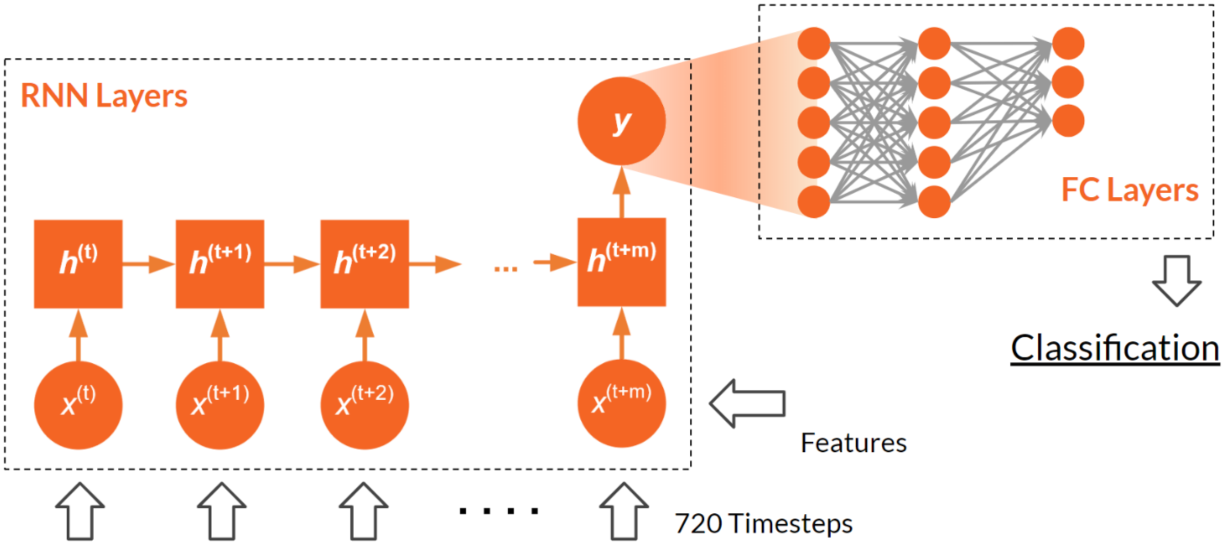
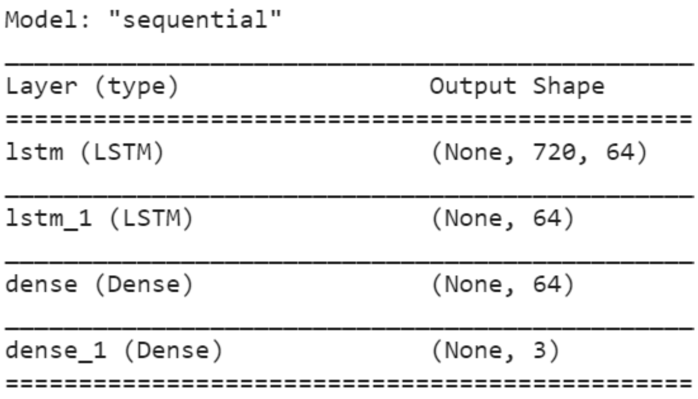


*Figure 8.8 Structure of the RNN (LSTM) model for spark plugs*

Table 8.1 shows the structure of the Long Short-Term Memory Recurrent Neural Network (LSTM-RNN).

*Table 8.1 Parameters of the LSTM-RNN model for spark plug degradation detection*



```
Model: "sequential"
_________________________________________________
Layer (type)                 Output Shape
=================================================
lstm (LSTM)                  (None, 720, 64)
_________________________________________________
lstm_1 (LSTM)                (None, 64)
_________________________________________________
dense (Dense)                (None, 64)
_________________________________________________
dense_1 (Dense)              (None, 3)
=================================================
```

The table shows the structure and hyperparameters of the model as follows:

- *Input layer: 720 units*
- *3 Hidden layers:*
  - *LSTM: 2 layers, 128 units*
  - *Fully Connected: 1 layer, 64 units (activation function = 'tanh')*
- *Output layer: 3 classes*
- *Optimizer: 'RMSProp'*
- *Loss: 'Sparse Categorical Cross-entropy'*
- *Epochs = 120*
- *Batch size = 32*

The dataset was split into 3 subsets: 60% for training, 20% for validation and 20% for testing. The testing set consisted of 1929 samples to evaluate the model. The entire training set was used to train the model. The validation set was not used for training but was used for refinement during the training process. After each epoch of training, the model predicts the validation set. The testing set was never used for training nor refinement but was held for post-training evaluation of the model.

The RNN model was able to converge very well throughout the training (Figure 8.9) and validation (Figure 8.10) processes and was able to achieve a test accuracy of 97.4%. Figure 8.11 shows the confusion matrix of the model which shows the predicted labels versus the actual labels of each class. These results were based on regular data from days 1, 2, 4 and 5 only (see section 5.1).

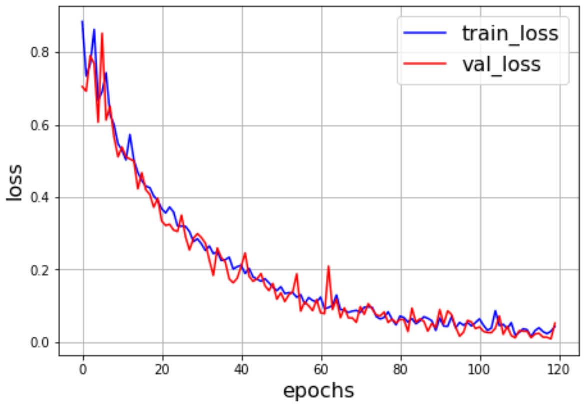


*Figure 8.9 Loss vs epochs – LSTM for spark plugs*

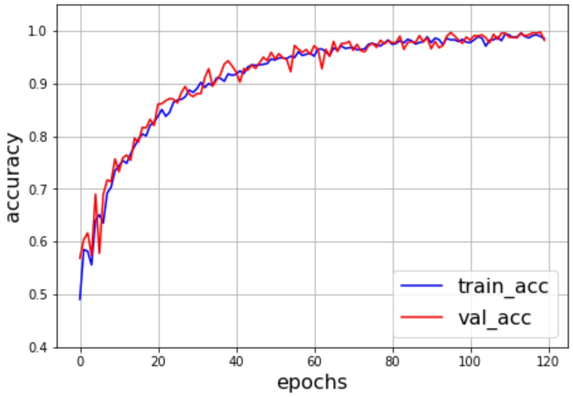


*Figure 8.10 Accuracy vs epochs -LSTM for spark plugs*

| | | Predicted Labels | | | |
|---|---|---|---|---|---|
| | | **Healthy** | **0.020”** | **0.080”** | **Testing Samples** |
| **Actual Labels** | **Healthy** | **99.9%** | **0.06%** | **0.04%** | **643** |
| | **0.020”** | **0.7%** | **96.5%** | **1.9%** | **643** |
| | **0.080”** | **1.8%** | **0.4%** | **97.8%** | **644** |
| | **Total** | **97.4%** | | | **1929** |
| | | **Avg Detection Rate** | | | |

*Figure 8.11 Test confusion matrix of spark plug results using LSTM-RNN*

### *8.1.3 Degradation Detection Using Deep CNNs*

Unlike sequential features, the input to the Convolutional Neural Network was 2-dimensional (20x15) heat maps as mentioned in the previous chapters. The heat maps were constructed using Mel-Frequency Cepstral Coefficients (MFCC) maps. The CNN model consisted of Convolution and Max Pooling layers from which the output was taken to fully connected layers to connect all the information coming from each of the 4 features. Figure 8.12 shows the CNN model structure.

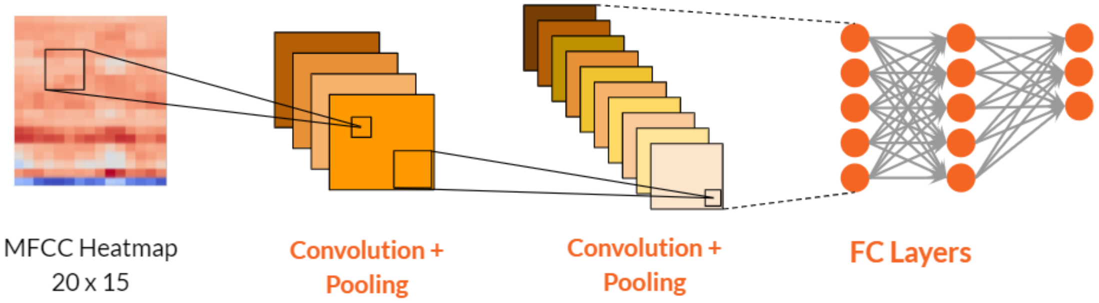
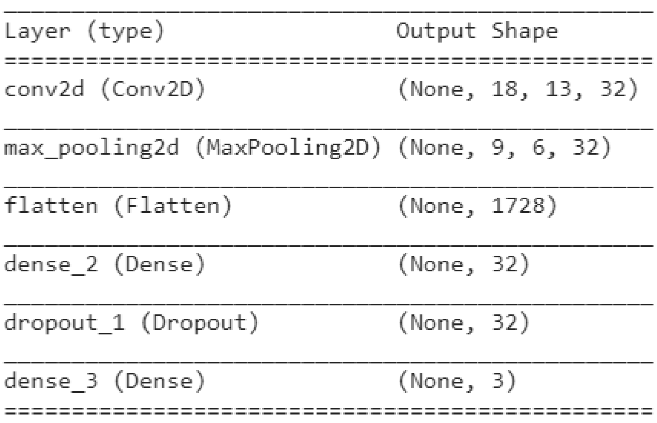


*Figure 8.12 Structure of the CNN model for spark plugs*

Table 8.2 shows the structure of the individual layers including the input and output shapes as well as the total and individual number of parameters processed per layer:

*Table 8.2 Parameters of the CNN model for spark plug degradation detection*



```
_________________________________________________
Layer (type)                 Output Shape
=================================================
conv2d (Conv2D)              (None, 18, 13, 32)
_________________________________________________
max_pooling2d (MaxPooling2D) (None, 9, 6, 32)
_________________________________________________
flatten (Flatten)            (None, 1728)
_________________________________________________
dense_2 (Dense)              (None, 32)
_________________________________________________
dropout_1 (Dropout)          (None, 32)
_________________________________________________
dense_3 (Dense)              (None, 3)
=================================================
```

The dataset was split into 60% for training, 20% for validation and 20% for testing. Like in the LSTM-RNN model, the training set was all used for training while the validation set was used for refinement. The test set was never seen by the model and contained 1929 samples to evaluate the model after the training process is over. Table 8.2 shows the structure and hyperparameters of the CNN model which contains the following:

- *Input layer: 720 units*
- *3 Hidden layers:*
  - *Conv2D: 2 layers, 32 units*
  - *Fully Connected: 1 layer, 64 units (activation function = 'relu')*
- *Output layer: 3 classes*
- *Optimizer: 'Adam'*
- *Loss: 'Sparse Categorical Cross-entropy'*
- *Epochs = 80*
- *Batch size = 32*

The CNN model was able to converge well through the training and validation process. As Figure 8.13 shows below.

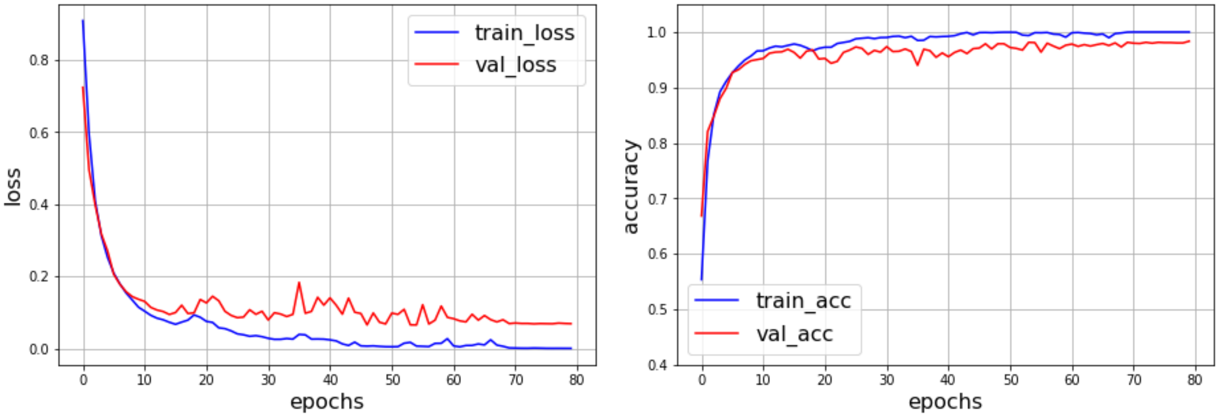
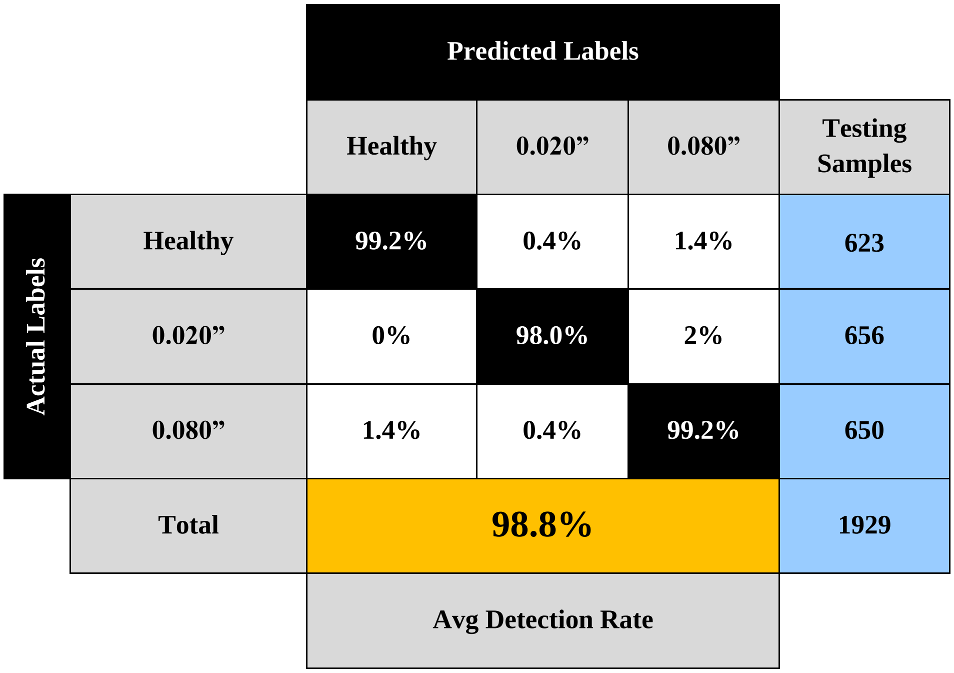


***Figure 8.13 Accuracy and loss – CNN for spark plugs***

The CNN model was able to achieve a test accuracy of 98.8%. Below is the test confusion matrix of the model which shows the predicted labels versus the actual labels of each class.



| | | Predicted Labels | | | |
|---|---|---|---|---|---|
| | | Healthy | 0.020” | 0.080” | Testing Samples |
| Actual Labels | Healthy | 99.2% | 0.4% | 1.4% | 623 |
| | 0.020” | 0% | 98.0% | 2% | 656 |
| | 0.080” | 1.4% | 0.4% | 99.2% | 650 |
| | Total | 98.8% | | | 1929 |
| | | Avg Detection Rate | | | |

***Figure 8.14 Test confusion matrix of spark plug results using MFCC-CNN***

Although the EMSPCA model was able to detect spark plug degradation using existing knock sensors at high detection rates, it was not able to do so using speed sensors. This was due to the nature of the raw speed sensor data for which the EMSPCA model was not a good fit. Therefore, another model was required to try and detect spark plug degradation using speed sensors.

### *8.1.4 Degradation Detection Using EMSPCA-Mosaic-CNN*

The EMSPCA-Mosaic-CNN model, which was explained in detail in Chapter 6, was developed particularly to solve the data variability problem (see section 5.1). The problem was that some of the collected data on specific days (day 3, 6 and 7) had different ranges of voltage amplitudes and frequencies due to experimental issues, namely variant data. These 3 days could not be diagnosed by any of the AI models in this research. The EMSPCA-Mosaic-CNN is an invariant AI model, inspired by a novel pre-processing method, namely mosaic images (see chapter 6 for description), which performs image pre-processing of time series data.

The training dataset for the EMSPCA-Mosaic-CNN model consisted of data from the standard 4 knock sensors only, with wavelet level 4 fault signatures, horizontally and vertically augmented 32x100 mosaic images, at 20 engine cycles per sample. Figure 8.15 shows the augmented mosaic image which was used as input to the EMSPCA-Mosaic-CNN model. Each mosaic image was augmented twice in the vertical direction and 25 times in the horizontal direction.

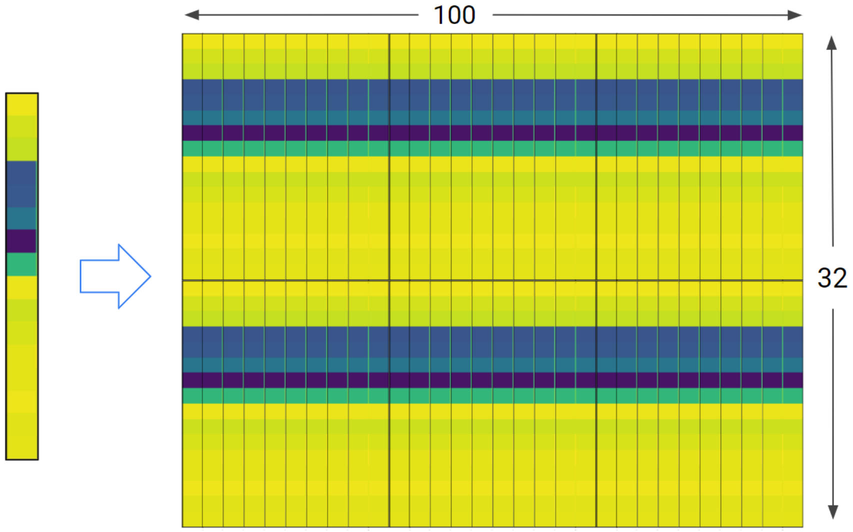


***Figure 8.15 Horizontally and vertically augmented mosaic image***

The dataset contained 1,860 samples which were split into 80% for training, 20% for validation. The model was able achieve a training accuracy of 100% and a validation accuracy of 96%. Figure 8.16 shows the training accuracy throughout 150 epochs against the validation accuracy. A set of 28 samples was kept aside for testing. The model was able to detect 25 out of 28 unlabeled test samples, which included variant data. Figure 8.17 shows the test confusion matrix.

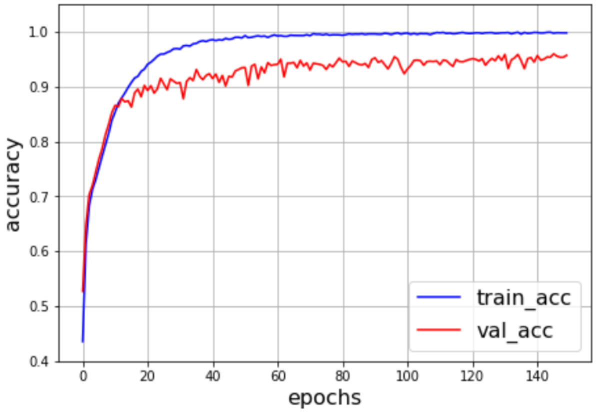
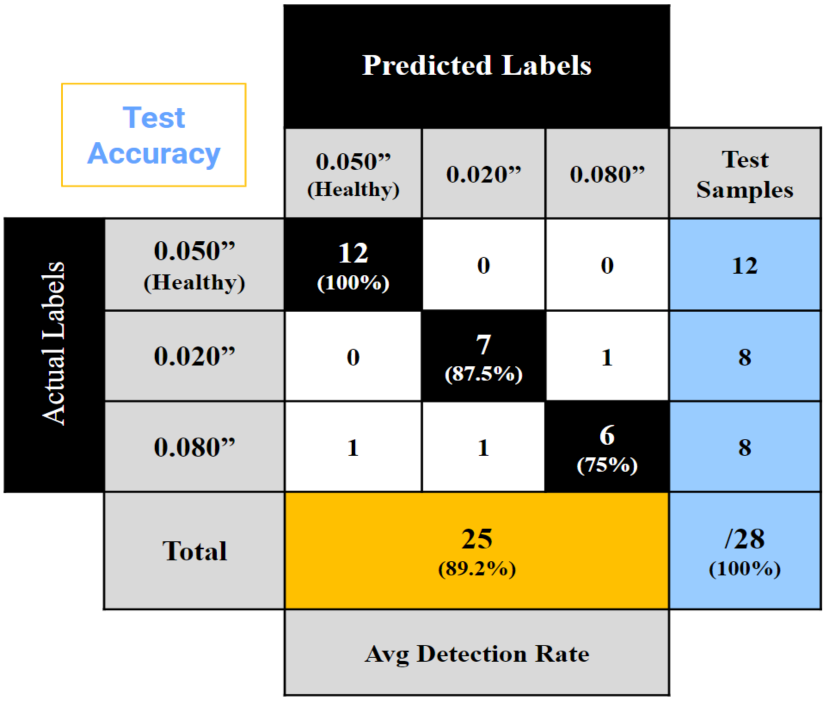


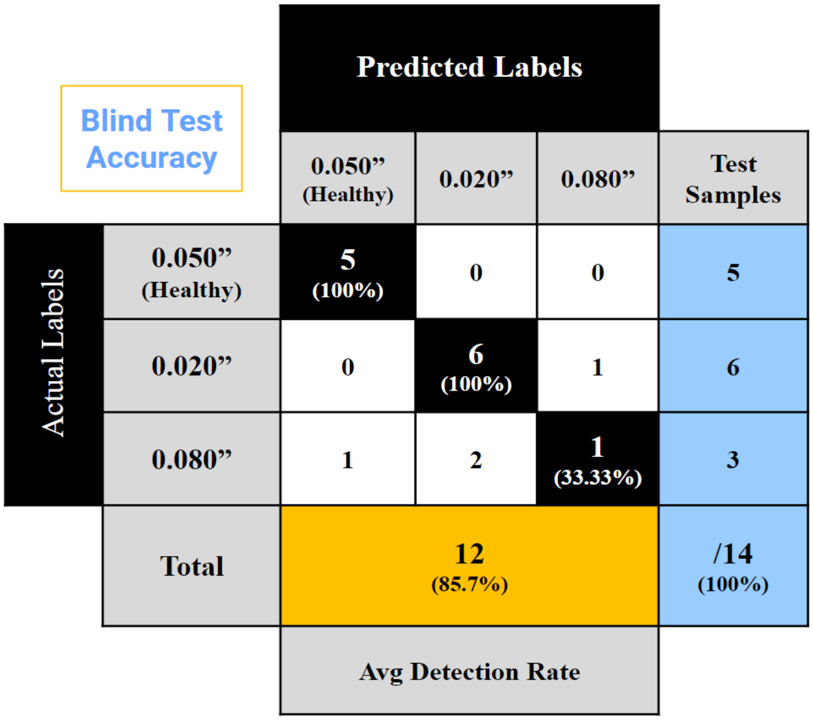
*Figure 8.16 Training vs validation accuracy of EMSPCA-Mosaic-CNN using 1-knock mosaic images*



| Test Accuracy | | Predicted Labels | | | |
|---|---|---|---|---|---|
| | | 0.050”<br>(Healthy) | 0.020” | 0.080” | Test Samples |
| Actual Labels | 0.050”<br>(Healthy) | 12<br>(100%) | 0 | 0 | 12 |
| | 0.020” | 0 | 7<br>(87.5%) | 1 | 8 |
| | 0.080” | 1 | 1 | 6<br>(75%) | 8 |
| | Total | 25<br>(89.2%) | | | /28<br>(100%) |
| | | Avg Detection Rate | | | |

*Figure 8.17 Training vs validation accuracy of EMSPCA-Mosaic-CNN*

**Blind Test:** An additional blind test was performed by a CMHT test engineer provided 14 completely unlabeled samples, which included variant data (see section 5.1) to confirm the performance of the EMSPCA-Mosaic-CNN model. The model was able to detect 100% of the samples and diagnosed 12 out of 14. Figure 8.18 shows the results of the blind test.



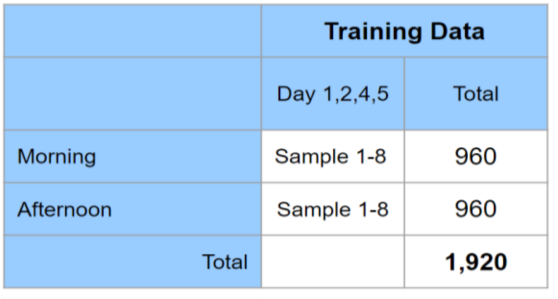
| Blind Test Accuracy | | Predicted Labels | | | |
|---|---|---|---|---|---|
| | | 0.050” (Healthy) | 0.020” | 0.080” | Test Samples |
| Actual Labels | 0.050” (Healthy) | 5 (100%) | 0 | 0 | 5 |
| | 0.020” | 0 | 6 (100%) | 1 | 6 |
| | 0.080” | 1 | 2 | 1 (33.33%) | 3 |
| | Total | 12 (85.7%) | | | /14 (100%) |
| | | Avg Detection Rate | | | |

*Figure 8.18 Blind test accuracy of EMSPCA-Mosaic-CNN*

The EMSPCA-Mosaic-CNN model was able to solve the variability problem in the engine data collected in this research, which was described in section 5.1. All data from days 1, 2, 4, and 5 were diagnosed with high detection rates by different models in this thesis. However, variant data from days 3, 6 and 7 could not be diagnosed by any of the models developed earlier. The EMSPCA-Mosaic-CNN, which was developed particularly for this problem, was the only model which was able to detect and diagnose variant data in different unlabeled tests.

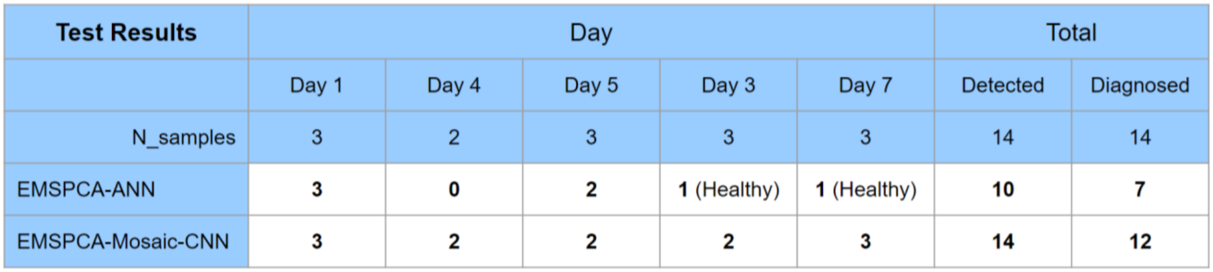
Below is a comparison between the results from the EMSPCA-ANN and the EMSPCA-Mosaic -CNN models of the blind test. Both models were trained on the same data provided in Table 8.3 and were tested blindly as described above. Table 8.4 shows in detail what day each sample belongs to. The EMSPCA-Mosaic-CNN has outperformed the EMSPCA-ANN model, especially in the classification of variant data. This explains the impact of the novel mosaic images in this research. Figure 8.19 shows a bar chart comparing the performance of both models.

*Table 8.3 Training data for the EMSPCA-Mosaic-CNN blind test on variant data*



| | **Training Data** | |
|---|---|---|
| | Day 1,2,4,5 | Total |
| Morning | Sample 1-8 | 960 |
| Afternoon | Sample 1-8 | 960 |
| Total | | **1,920** |

*Table 8.4 Results of EMSPCA-Mosaic-CNN blind test on variant data*



| **Test Results** | Day | | | | | Total | |
|---|---|---|---|---|---|---|---|
| | Day 1 | Day 4 | Day 5 | Day 3 | Day 7 | Detected | Diagnosed |
| N_samples | 3 | 2 | 3 | 3 | 3 | 14 | 14 |
| EMSPCA-ANN | **3** | **0** | **2** | **1** (Healthy) | **1** (Healthy) | **10** | **7** |
| EMSPCA-Mosaic-CNN | **3** | **2** | **2** | **2** | **3** | **14** | **12** |

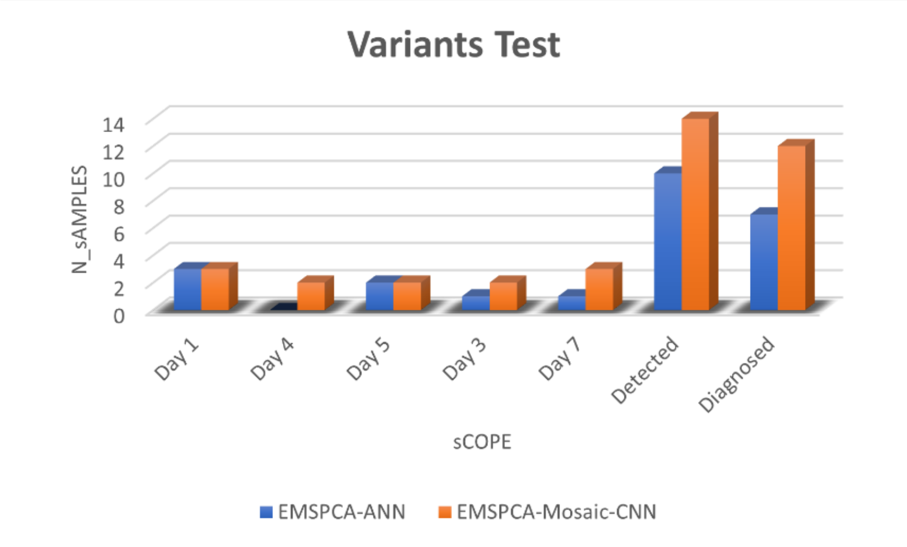


*Figure 8.19 EMSPCA-ANN vs EMSPCA-Mosaic-CNN on variant data*

### 8.1.5 Degradation Detection Using Random Forest

Unlike neural networks, the Random Forest algorithm does not work best with raw time-series data. As discussed in Chapter 2, it consists of several decision trees, each works as an individual classifier.

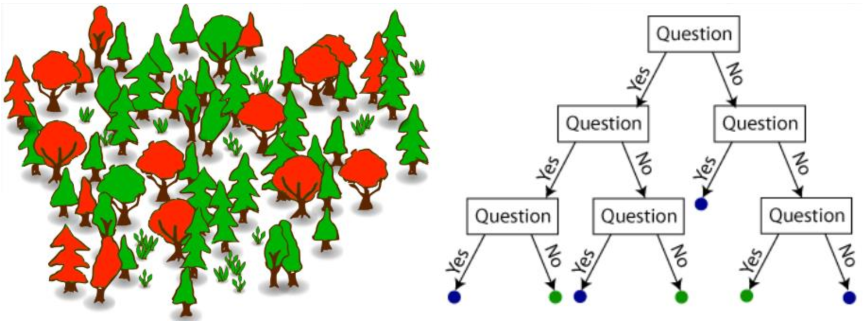


*Figure 8.20 Random Forest trees*

As shown in Figure 8.20, each tree looks at the data from a different perspective and answers a hierarchy of questions until it votes for a label. By the end of the training process, the number of predicted labels is equal to the number of trees. The label which gets most votes of all trees becomes the final label like Figure 8.21 presents.

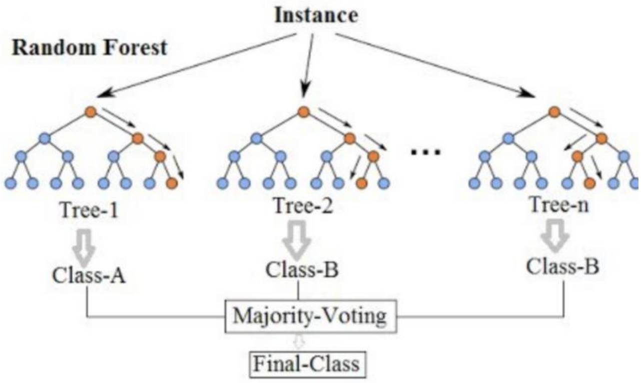


*Figure 8.21 Random forest majority voting*

The developer of a Random Forest model must be creative and focus on relevant features which emphasize differences between classes. Irrelevant features would make the model computationally heavier and will not add value to the final decision. That is because the model will eventually rely on relevant features which have the highest correlation with the target. Irrelevant features will have low participation in the voting process. For example, building a RF classifier to classify images of oranges and bananas, may require the shape, dimensions, and color of the fruit as relevant features. However, the background color of the image would not have low participation in the voting hierarchy. Random Forest (RF) has an important tool, namely feature importance. When the training process is over, this tool provides the user automatically with a list of all the input features along with a percentage which describes the weight of the feature towards the final vote.

Choosing the right model for each kind of data is key to success. In this model, only one existing speed sensor (i.e., optical encoder) was used to detect spark plug degradation. As discussed earlier, Random Forest (RF) works best with descriptive features rather than time series data. The input features to the RF model had to be extracted from the raw signal of the speed sensor. Figure 8.22 shows an example of a raw speed sensor signal when both small and large spark plug gaps were placed in the engine while running.

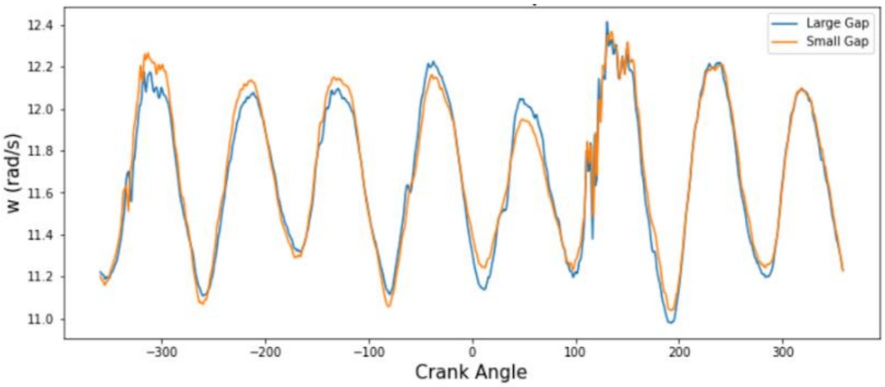


***Figure 8.22 Engine speed sensor single-cycle data***

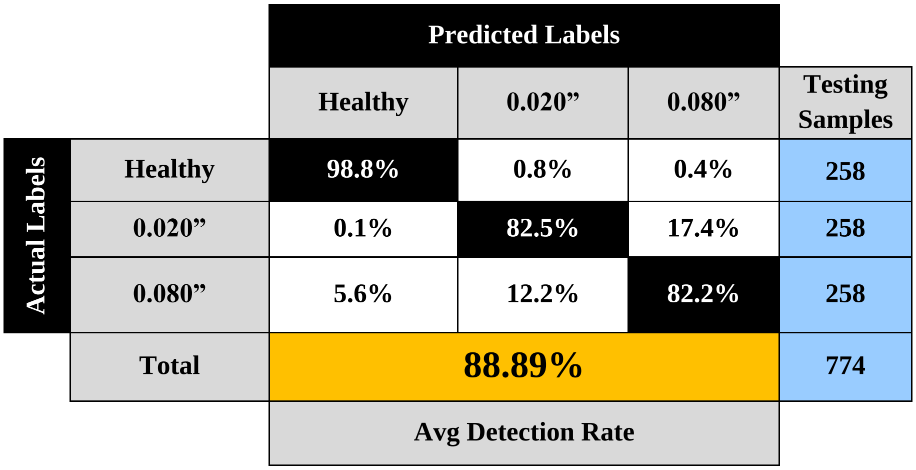
The raw speed sensor data consists of a sinusoidal signal. Both small and large spark plug gaps follow the same trend and the difference in amplitude is not big. Thus, the following curve-shape description features were calculated:

- First Voltage
- Last Voltage
- Minimum Voltage
- Maximum Voltage
- Average of peaks
- Average of dips
- Overall Mean Voltage
- First - Min
- Last - Min
- Area under the curve
- Maximum - Minimum
- Maximum - First

- Maximum - Last
- Maximum - Mean
- Average of peaks - Minimum
- Average peaks - First
- Average of peaks - Last
- Average of peaks - Mean
- Mean - Minimum
- Mean - First
- Crank Angle at Maximum
- Crank Angle at Minimum
- Maximum - Minimum (Angles)
- Maximum - First (Angles)
- Standard Deviation of the engine cycle
- 2* Standard Deviation of the engine cycle
- 3* Standard Deviation of the engine cycle
- Speed at 25% of the crank angles
- Speed at 50% of the crank angles
- Speed at 75% of the crank angles

Feeding the Random Forest model with curve-shape description features resulted in a detection rate of 88.89%. This means that in every 100 vehicles, the proposed model can accurately classify almost 89 vehicles as either healthy or degrading spark plug. Prior knowledge about vehicles with degrading spark plugs can prevent the vehicle from low performance and may result in serious engine damage. Figure 8.23 shows the confusion matrix of the Curve Description Random Forest

(CD-RF) model in detecting spark plug degradation using regular data from day 1, 2, 4, and 5 only (see section 5.1).



| | | Predicted Labels | | | |
|---|---|---|---|---|---|
| | | Healthy | 0.020” | 0.080” | Testing Samples |
| Actual Labels | Healthy | 98.8% | 0.8% | 0.4% | 258 |
| | 0.020” | 0.1% | 82.5% | 17.4% | 258 |
| | 0.080” | 5.6% | 12.2% | 82.2% | 258 |
| | Total | 88.89% | | | 774 |
| | | Avg Detection Rate | | | |

*Figure 8.23 Test confusion matrix of spark plug results using CD-RF*

## 8.2 Results of EGR Valve Degradation Detection

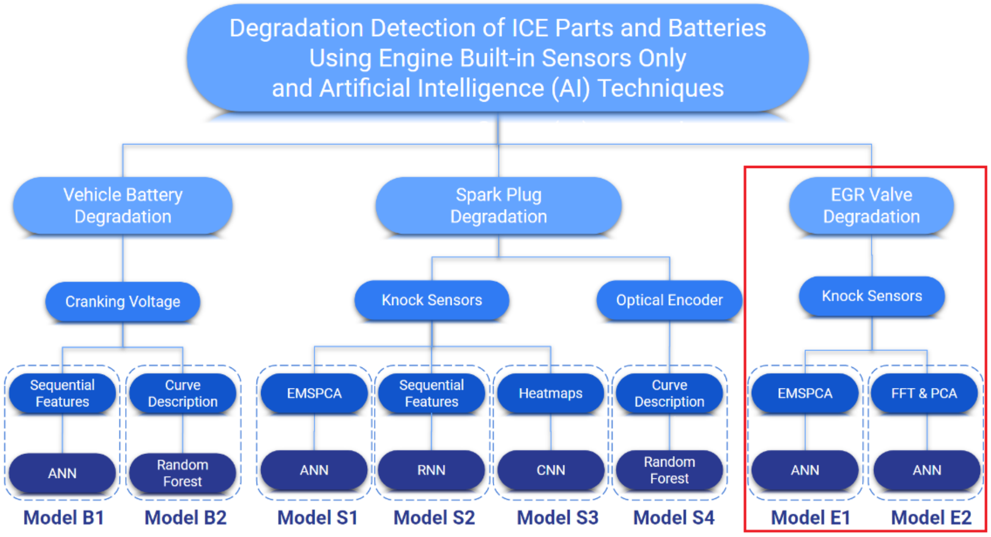


*Figure 8.24 EGR degradation detection - Models E1 and E2*

Two models were proposed to detect degradation of EGR valves: Model E1 and Model E2 (see section 1.5) as shown in Figure 8.24. Model E1 consists of the EMSPCA algorithm along with an Artificial Neural Network (EMSPCA-ANN). Model E2 starts with a pre-processing layer of Fast Fourier Transform and Principal Component Analysis (FFT-PCA), followed by an ANN. Both models used standard knock sensors to classify EGR faults.

### *8.2.1 EGR Degradation Detection Using EMSPCA*

Using the EMSPCA algorithm, fault signatures of EGR valve were generated for each of the EGR levels: Healthy, level 1 CO2 dilution (i.e., fault 1) and level 2 CO2 dilution (i.e., fault 2). Figure 8.25 – 8.27 show the average fault signatures of 3 fault conditions:

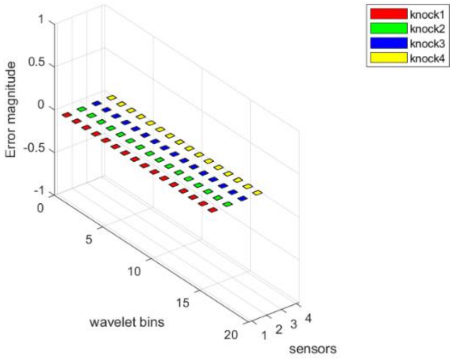


***Figure 8.25 Fault signature of a healthy EGR valve (Level 0)***

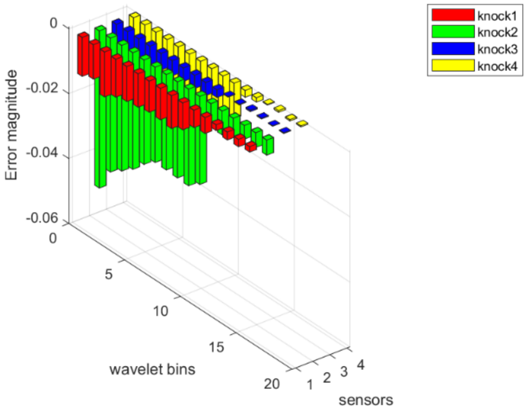


***Figure 8.26 Fault signature of EGR level 1***

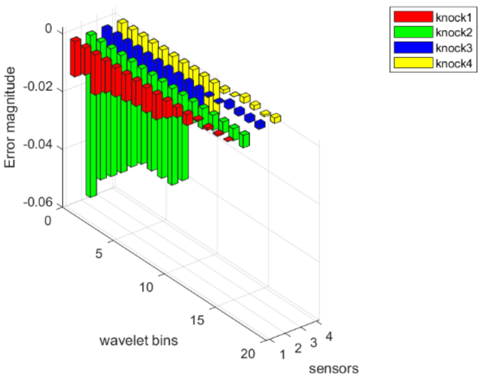


*Figure 8.27 Fault signature of EGR level 2*

Like the spark plug fault signatures, flat frequency bins were generated for healthy EGR valves (i.e., level 0). This confirms that the deviation of a healthy data sample from the healthy baseline is close to zero. The next fault signature presents a data sample of a level 1 EGR. Many bins in knock sensors 1,3, and 4 show deviations from the baseline. However, knock sensor 2 shows the most significant deviations. The bin shows a maximum error magnitude of about 0.5. The last fault signature represents level 2 EGR, which shows similar deviations to the level 1 EGR in knock sensors 1,3, and 4. Sensor 2 again shows a maximum error magnitude of almost 0.6. Although the difference between level 0 and 1 is very noticeable, the difference between levels 1 and 2 is not big, which makes the diagnosis of the fault more difficult than detecting it. These fault signatures were used as input to the Artificial Neural Network (ANN), representing the machine learning model. The output layer of the ANN consisted of 3 classes:

a) Class 1: Level 0 EGR (i.e., healthy).
b) Class 2: Level 1 EGR (i.e., 5% Co2 dilution).
c) Class 3: Level 2 EGR (i.e., 10% Co2 dilution).

The ML model shown in Figure 8.28 consisted of a single input layer. The dimensions of the input layer were 4x16, which represented the number of sensors x the number of fault signature bins.

Three hidden layers were designed to calculate the feature weights. The output layer consisted of 3 units per sample. Each unit provided a probability for the sample to belong to one of the 3 conditions.

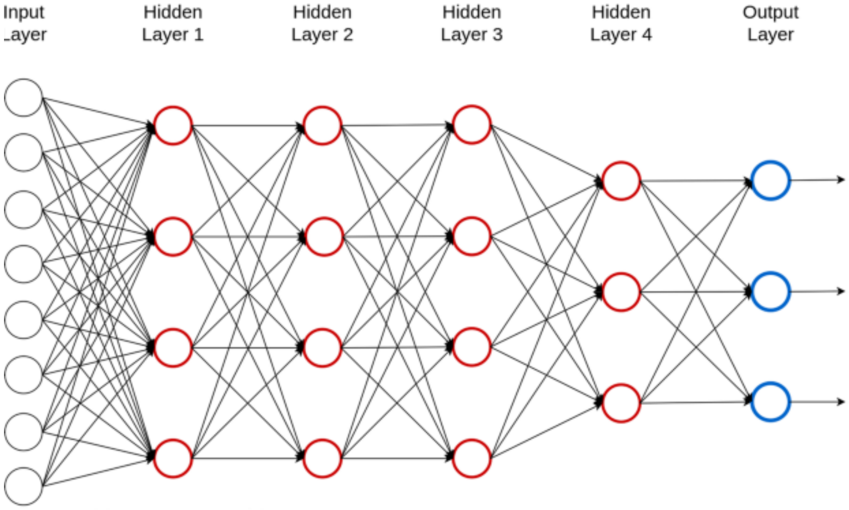


***Figure 8.28 ANN for EGR degradation detection***

The results of the EMSPCA-ANN model for EGR valve degradation detection are presented below. Figure 8.29, 8.30 and 8.31 show example of the probability bar charts provided for a healthy, fault 1, and fault 2 test samples, respectively. The sample with the highest probability becomes the final label to the test sample.

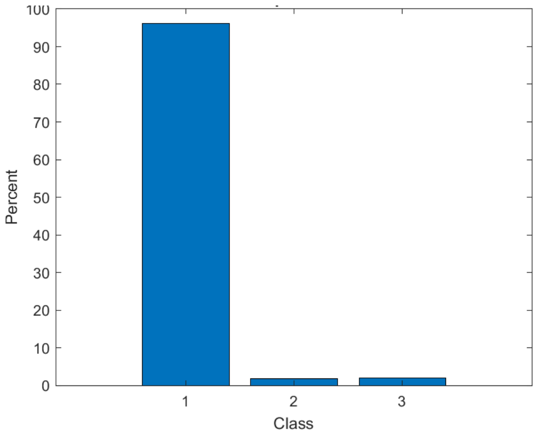


***Figure 8.29 EMSPCA predictions for EGR degradation detection – class 1***

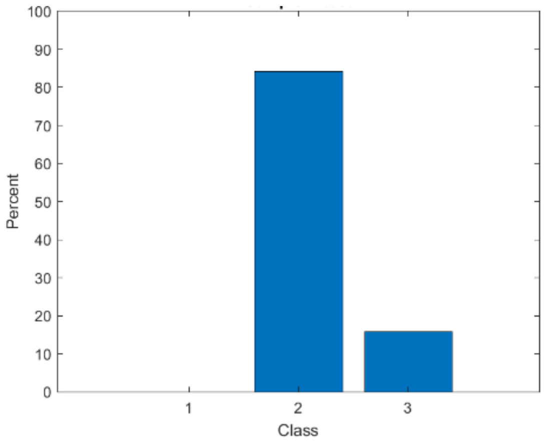


*Figure 8.30 EMSPCA predictions for EGR degradation detection – class 2*

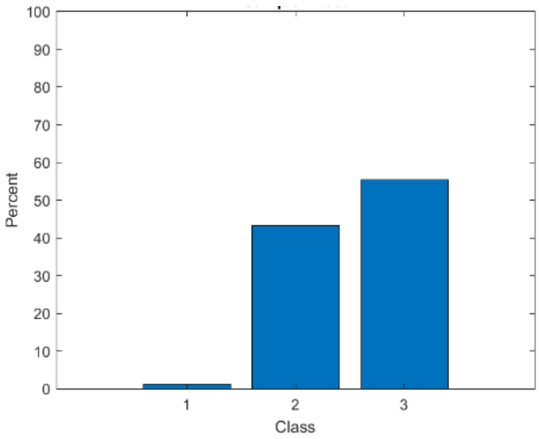


*Figure 8.31 EMSPCA predictions for EGR degradation detection – class 3*

The total average detection rate represents the number of samples which were correctly predicted. The EMSPCA-ANN model achieved 100% detection rate. The algorithm was tested with test data samples from the same day, same time of the day, different days, and different times of the day to consider the reliability of the model. The model was still able to detect and diagnose the degradation in 100% of the test samples, explained in the confusion matrix in Figure 8.32.

| | | Predicted Labels | | | |
|---|---|---|---|---|---|
| | | Healthy | Level 1 | Level 2 | Testing Samples |
| Actual Labels | Healthy | 100% | 0% | 0% | 300 |
| | Level 1 | 0% | 100% | 0% | 300 |
| | Level 2 | 0% | 0% | 100% | 300 |
| | Total | 100% | | | 900 |
| | | Avg Detection Rate | | | |

*Figure 8.32 Test accuracy for EGR valve using EMSPCA-ANN*

### 8.2.2 EGR Degradation Detection Using FFT-PCA

Although the FFT-PCA model is simpler than the EMSPCA algorithm, it was able to detect degradation in many of the EGR test samples. This could not happen with spark plug problem since their faults were more difficult to detect. In this model, the raw signal was converted to the frequency domain and then turned into ten unitless principal components which described significant features in the original signal. Figure 8.33 shows an example of the difference in amplitude between level 1 EGR and level 2 EGR in the frequency domain. The same tests which were conducted with the EMSPCA-ANN algorithm were repeated for the FFT-PCA model. The model was trained with 300 data samples for each of the 3 classes and was then tested using a set of unlabeled data to assess its performance. Figure 8.34 shows the confusion matrix of the EGR tests using the FFT-PCA method.

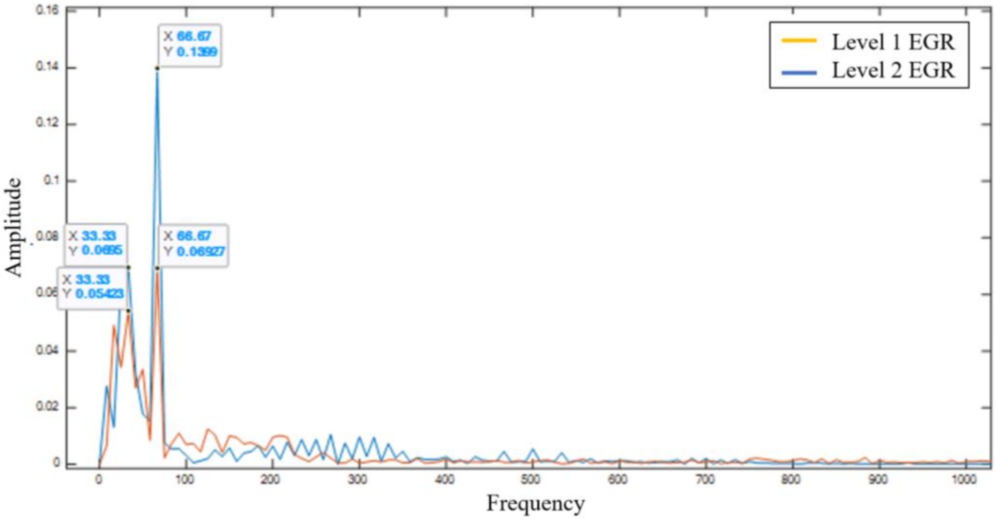
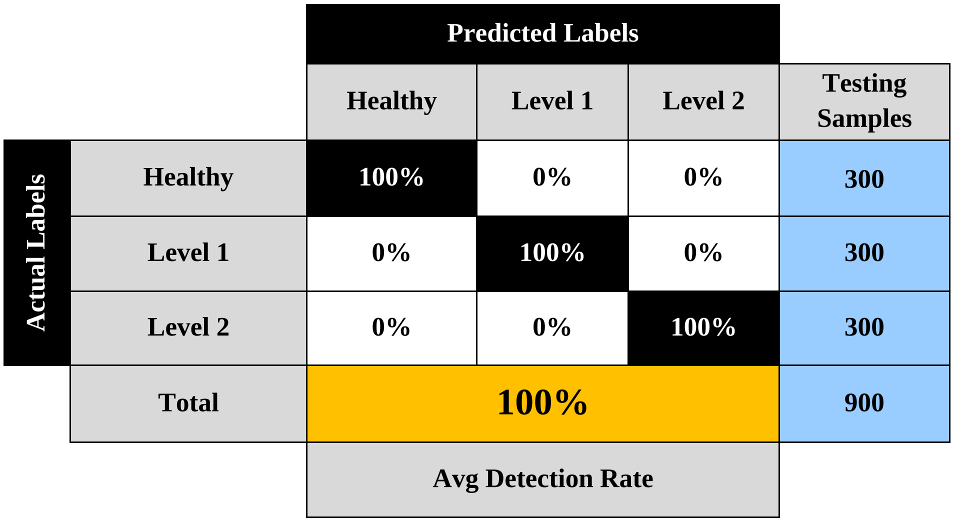


*Figure 8.33 Frequency analysis of level 1 vs level 2 EGR*



| | | Predicted Labels | | | |
|---|---|---|---|---|---|
| | | Healthy | Level 1 | Level 2 | Testing Samples |
| Actual Labels | Healthy | 100% | 0% | 0% | 300 |
| | Level 1 | 0% | 100% | 0% | 300 |
| | Level 2 | 0% | 0% | 100% | 300 |
| | Total | 100% | | | 900 |
| | | Avg Detection Rate | | | |

*Figure 8.34 Test accuracy for EGR valve – EMSPCA*

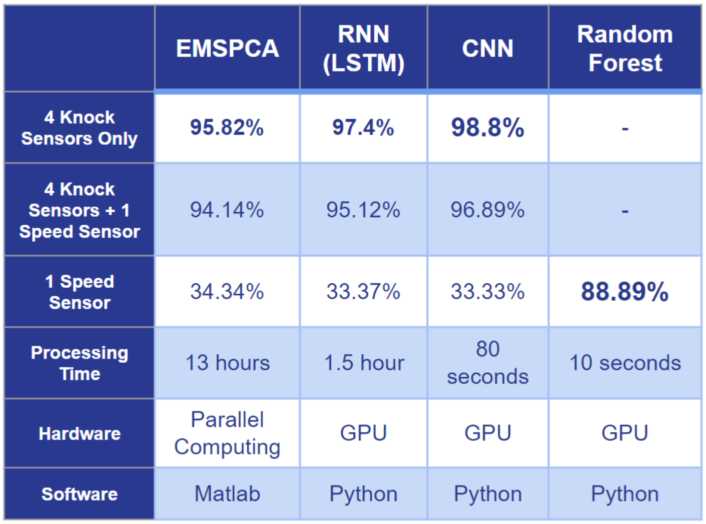
## *8.3 Summary of FDD Results*

This section provides a summary of the results of all the models developed and implemented to detect each of the following problems: Spark Plug Degradation and EGR degradation.

### *8.3.1 Summary of Spark Plug Degradation Results*

Four models were developed and discussed to detect spark plug degradation. Table 8.5 shows the performance of each of the models along with the sensors used and run times. Although EMSPCA, LSTM, and CNN all showed excellent performances using existing knock sensors, the Convolutional Neural Network (CNN) had the highest performance and run time. Random Forest was the only model which was able to detect spark plug degradation using the built-in engine speed sensor due to the nature of the data. The model showed a detection rate of almost 89%.

***Table 8.5 Summary of spark plug degradation detection results***



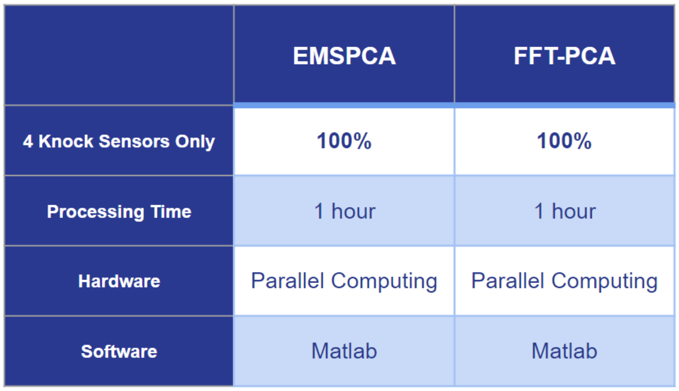
| | EMSPCA | RNN (LSTM) | CNN | Random Forest |
|---|---|---|---|---|
| **4 Knock Sensors Only** | **95.82%** | **97.4%** | **98.8%** | - |
| **4 Knock Sensors + 1 Speed Sensor** | 94.14% | 95.12% | 96.89% | - |
| **1 Speed Sensor** | 34.34% | 33.37% | 33.33% | **88.89%** |
| **Processing Time** | 13 hours | 1.5 hour | 80 seconds | 10 seconds |
| **Hardware** | Parallel Computing | GPU | GPU | GPU |
| **Software** | Matlab | Python | Python | Python |

All the results listed in the table were based on engine data of days 1, 2, 4 and 5 which had similar amplitudes and frequencies. These models were not able to diagnose variant data such as days 3, 6, 7 and 8. The novel method, referred to as the EMSPCA-Mosaic-CNN, were able to achieve 96% detection rate, using all the available dataset, including variant data.

### *8.3.2 Summary of EGR Results*

For the EGR problem, two models were developed to detect EGR valve degradation. Table 8.6 shows the performance of each model along with the sensors used, run time. Both models achieved a detection rate of 100%.

***Table 8.6 summary of EGR results***



| | EMSPCA | FFT-PCA |
|---|---|---|
| **4 Knock Sensors Only** | **100%** | **100%** |
| **Processing Time** | 1 hour | 1 hour |
| **Hardware** | Parallel Computing | Parallel Computing |
| **Software** | Matlab | Matlab |

# Chapter 9

# Conclusions and Future Work

# *Chapter 9 - Conclusions and Future Work*

This chapter provides a summary of the research, novel contributions, implementations, and results of the Fault Detection and Diagnosis (FDD) solutions proposed in this thesis. Also, recommendations for future work are suggested.

## *9.1 Conclusions of The Research*

This research proposed Artificial Intelligence (AI) based solutions for fault detection and diagnosis of three vehicle parts: 12v accessory battery, spark plugs and Exhaust Gas Recirculation (EGR) valve. While developing these solutions, useful datasets were collected, and novel pre-processing techniques were invented to solve specific yet important classification problems. Eight Machine Learning (ML) and Deep Learning (DL) based Fault Detection and Diagnosis (FDD) models were developed in this research, summarized in Figure 9.1:

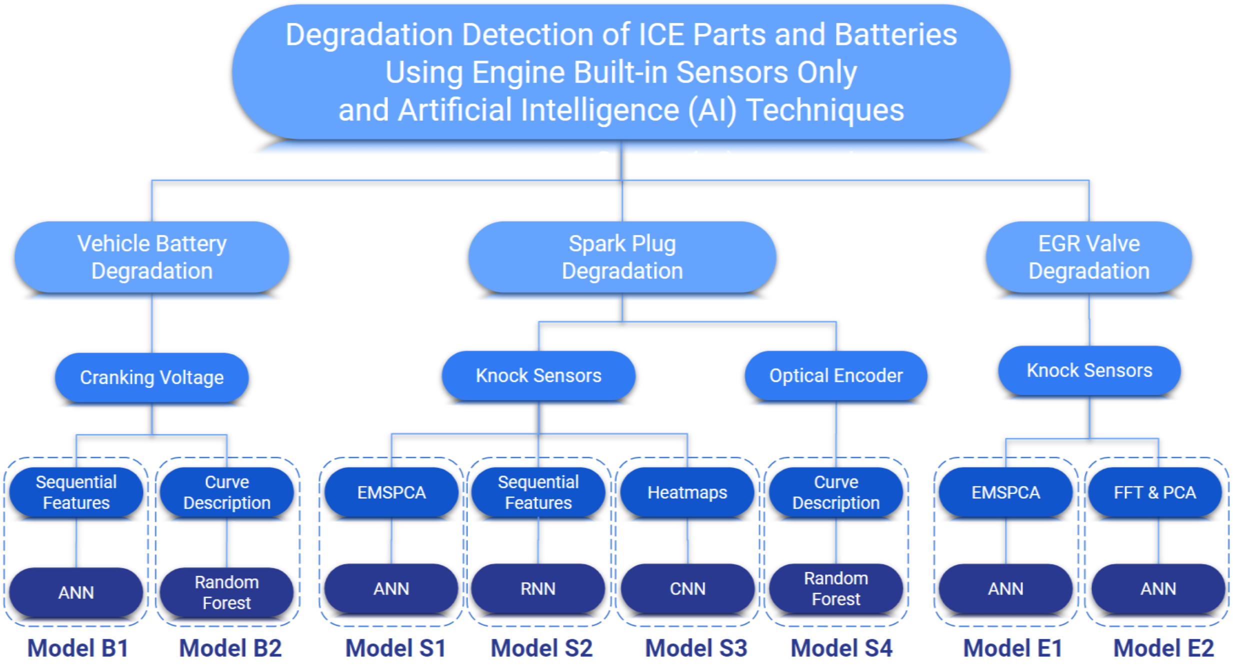


*Figure 9.1 AI-based FDD models*

The project started by a research collaboration between McMaster's Center for Mechatronics and Hybrid Technology (CMHT) and Geotab Inc., a world telematics leader. Geotab was looking for a research-based solution for their customers who suffered from on-duty failures of their commercial vehicle batteries. They wanted a solution which detects upcoming battery failures,

For this research, all the data was collected and stored by Geotab. The collected dataset contained both healthy and breakdown data. The healthy samples were cranking voltage history signals of vehicles which had healthy batteries, while breakdown data contained vehicles which had on-duty battery failures. Although data filtration was not easy, the lack of labels was a bigger problem, which were required to train the supervised learning models developed to detect battery degradation.

Two Machine Learning based models were developed as follows. Model B1 detected previous breakdowns in data samples and labeled them; and Model B2 detected ongoing battery degradation. Model B1 was able to detect 99.4% of the vehicles with previous breakdowns, while Model B2 was able to predict 95.8% of upcoming battery failures. The key to success of Model B2 was a novel pre-processing technique for Random Forest (RF) classifiers, which were not designed to classify time-series data. The pre-processing technique, namely Curve Description Random Forest (CD-RF), enabled time-series classification using Random Forest (RF) classifiers. The technique suggested a special set of features, namely Curve Description (CD) features which allowed the classifier to look at the data from a different point of view and extract distinguishable patterns.

Two ICE problems were also targeted in this research due to the limited scope of data gathering involving Geotab. These were Spark plug degradation detection and Exhaust Gas Recirculation

(EGR) valve failure. Since cost reduction was a main objective in this research, one of the biggest challenges was to solve these problems by only using built-in engine sensors. No external sensors or transducers were allowed to keep the cost of the solution to a potential user low. Thus, two kinds of standard sensors were selected for data collection: knock sensors and speed sensors. Four knock sensors and a single optimal encoder were used to collect data in this research. All data were collected by McMaster's Center for Mechatronics and Hybrid Technology (CMHT) test engineers using a 5.0L Ford Coyote Engine. An engine database was then built which contained data on both healthy and artificially induced fault conditions. The fault conditions contained both parts: spark plugs degradation and EGR valve fault, with each being operated under 3 conditions (two faulty and one healthy condition).

The engine database was then structured, uploaded, and stored on the cloud and was designed to import and export data directly to and from the AI-based models. The models were also operated on cloud virtual machines which enabled faster computation. This enabled a complete fault detection system implemented on the cloud. Sharing the cloud database which contains billions of engine data points can be as quick as few minutes.

Four models, including Deep Learning (DL) algorithms, were developed to detect spark plug degradation, three of which require 4 standard knock sensors, while the last one requires a single optical encoder. Model S1 (see Figure 9.1) consisted of the Extended Multi-Scale Principal Component Analysis (EMSPCA) along with an Artificial Neural Network (ANN). Model S2 started by simple sequential features, followed by a deep Long Short-Term Memory Recurrent Neural Networks (LSTM-RNN). Model S3, which achieved the highest performance, contained a new approach of image pre-processing time-series data using Mel-frequency Cepstral Coefficients

(MFCC) along with a deep Convolutional Neural Network (CNN). Model S4 used the novel Curve Description Random Forest [CD-RF] pre-processing technique, which allowed RF to detect spark plug degradation using the standard optimal encoder, after no other model was able to do so. Although all 4 models showed excellent performances, Model S3 (i.e., MFCC-CNN), achieved almost 99% detection and diagnosis rate.

All the previous models were able achieve excellent performance while classifying data with limited variability. Once some variability was introduced to some data samples, namely variant data, all 4 models failed to classify these samples. Therefore, an invariant AI model was developed to handle spark plug degradation detection with respect to data variability. CNN is a popular invariant Deep Learning (DL) model; however, CNN was designed to classify images. Thus, a novel pre-processing method was invented, namely mosaic images, to enable the use of CNN with EMSPCA. This model, namely EMSPCA-Mosaic-CNN, has outperformed all the others tried in this research by detecting spark plug faults in variant data with 96% detection rate. This model has solved the data variability issue.

Finally, the EGR problem was solved using 2 models: Model E1 and E2 (see Figure 9.1). Model E1 consisted of EMSPCA along with an ANN, while Model E2 contained a Fast Fourier Transform (FFT) with Principal Component Analysis (PCA) for pre-processing, along with an ANN. Both models were able to detect EGR condition with 100% success rate.

### *9.2 Recommendations and Future Work*

In this research, the developed ML and DL models were able to achieve high detection rates with experimental data gathered over 7 days of testing. Although thousands of samples were generated in these 7 days, Machine Learning algorithms can achieve better results if more data is available. More data does not always mean better results if the data is irrelevant, redundant, invalid, unlabeled, poor, or do not carry useful information. Thus, the first recommendation for upcoming researchers is to collect more labeled, relevant, diverse, and rich data. As a beginning, more days of data should be collected from the same engine. Then, different operating conditions should be included such as other engine speeds, loads temperatures and humidity levels. Switching sensors as well as data acquisition devices and cords would be useful to include more for inducing experimental variability. As a next step data from different engines should be collected. More data would result in the proposed models of this research achieving higher performances. Another recommendation is to explore more data augmentation strategies for the novel EMSPCA-Mosaic-CNN method proposed in this research.

# *Appendix A - Cloud-based Communications and Data Storage Strategies*

Data management is necessary when it comes to Big Data. Collecting large amounts of data requires proper data storage and governance. Building a clean and rich database of engine data is one of the contributions in this research. Not only that a cloud database will be useful for future research projects in the CMHT lab but can also help automotive researchers all over the world who suffer from the lack of data availability. Another great advantage of the cloud database is that huge amounts of data can be shared with anyone, anywhere in the world, just by adding the user email to the sharing list. No need to physically copy data or even sending them remotely. The owner of the database gets full control on what exactly will be shared with the other person.

## *A.1 Cloud-based Engine Database*

The Spark Plug and EGR data contain billions of data points which were stored in standard data folders on the CMHT server. The CMHT lab will continue collecting more engine data moving forward which include new engine faults and the amount of data points might reach trillions of data point. Copying all these data from the CMHT server to local computers to run classification models was a challenge. Therefore, the database was migrated to the cloud for better data storage and management. The top 3 cloud service providers in the world in 2020 were: Microsoft Azure, Google Cloud Platform (GCP), and Amazon Web Services (AWS). These providers offer different cloud services and cloud-based applications such as cloud storage, cloud database, live notebooks, virtual machines (VM), and data analysis tools.

Two main cloud services were needed in this project: Cloud database and virtual machines. The cloud database was needed for data storage and online querying while virtual machines were needed to run the classification models suggested in this project. Although Amazon Web Services (AWS) offers cloud storage, it did not offer online database querying at the time of this research. Thus, the options were narrowed down to Microsoft Azure and Google Cloud Platform (GCP). Both providers offered cloud databases and virtual machines (VM), however, GCP offered cheaper pricing. Therefore, Google Cloud Platform (GCP) was selected as the cloud service provider for this project. Migrating the local database to GBQ was a great solution for storing and managing big data on local computers since it has unlimited storage capacity. The cloud database application in GCP is called Google Big Query (GBQ). GBQ consists of a hierarchy of projects, datasets and tables as shown in Figure A.1:

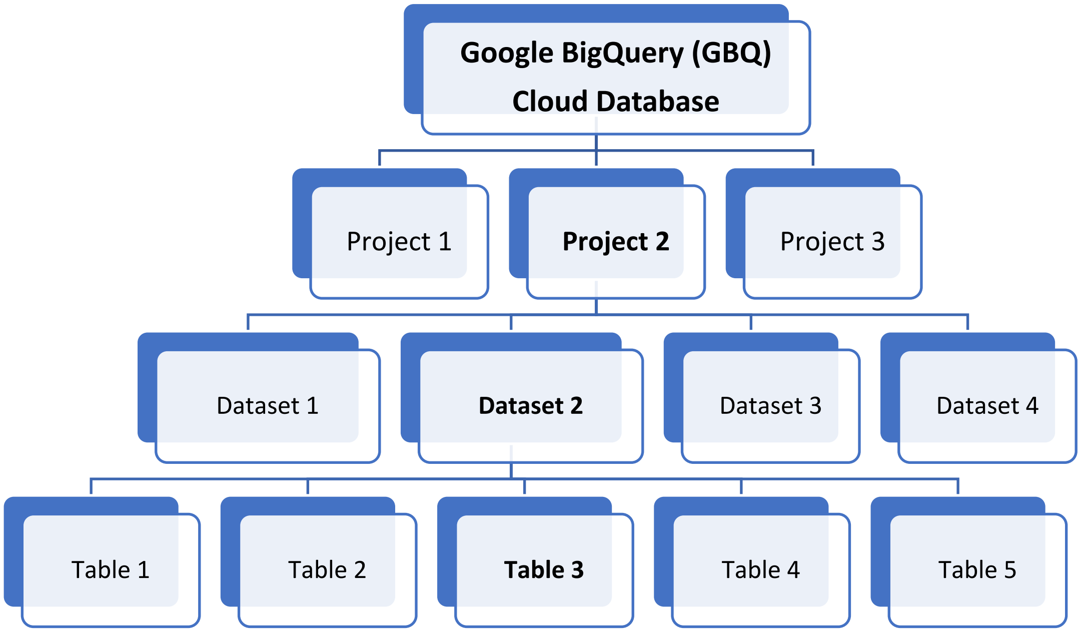


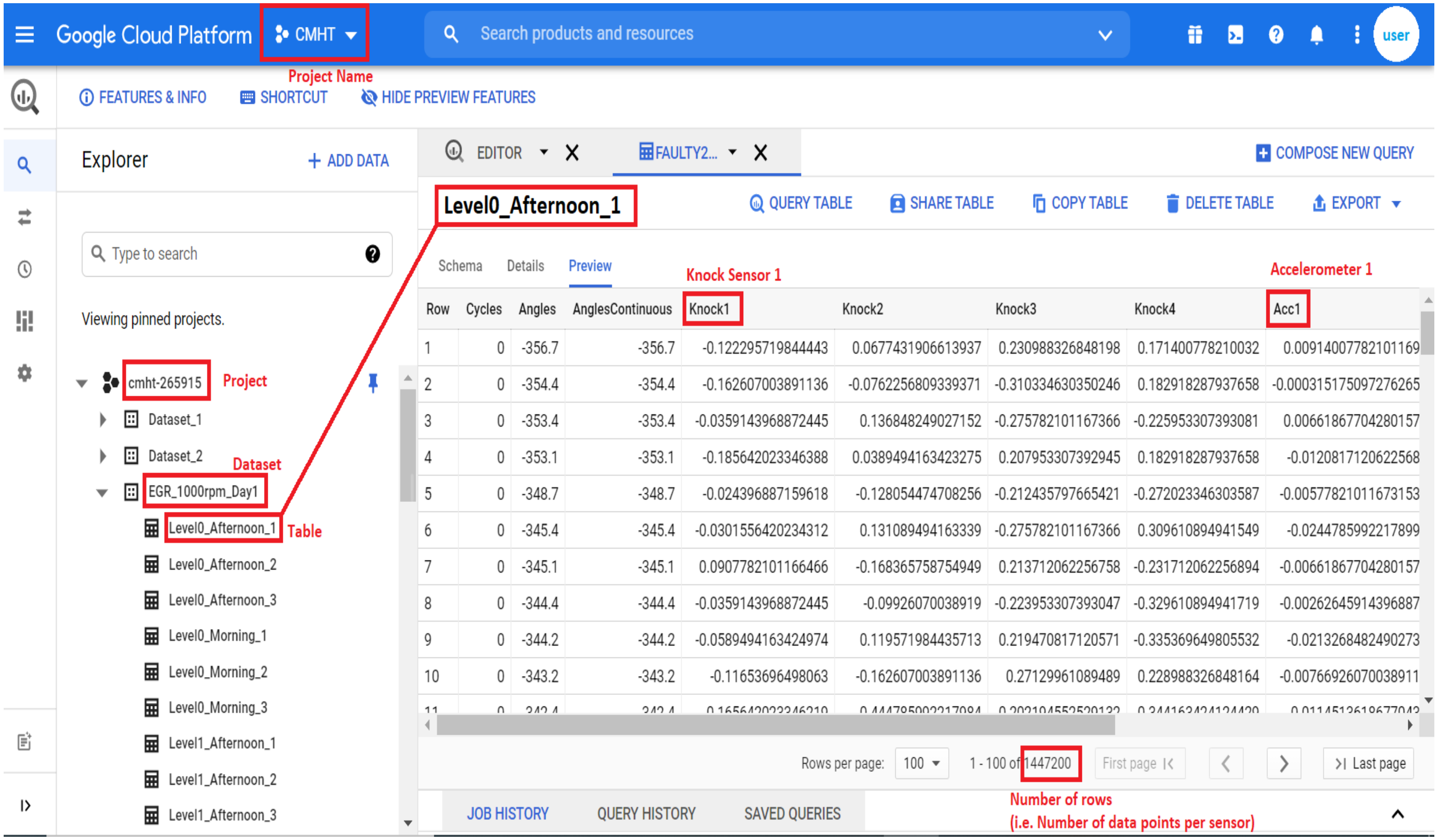
***Figure A.1 GBQ hierarchy***

## *A.2 Data Migration to Google Big Query Cloud Database*

Migrating a huge amount of data to the cloud would not be easy if done manually. The fresh data collected from the engine previously used to be stored in the CMHT server as multi-dimensional MAT files which are not accepted by Google Big Query (GBQ). Therefore, a cloud migration process was developed using MATLAB scripts and Command Prompt.

A MATLAB script was developed to convert all data files of the same category into CSV files after removing any redundant information. The CSV file contains single synchronized table that includes all the relevant sensory data which may be used for further analysis. The CSV files are created with the same names of the original data files to avoid confusions during the process. Users are only required to click the RUN button in MATLAB and all the work will be done automatically.

The next step is to use a single-line CMD command in Google Cloud SDK Shell, which connects local computers to the cloud platform. The CMD command takes all CSV files of the same category and uploads them directly to the corresponding GBQ datasets and tables, which are named with the same names of the original data files. The following Figure A.2 shows a screenshot from Google Big Query (GBQ) Cloud Database.



| Row | Cycles | Angles | AnglesContinuous | Knock1 | Knock2 | Knock3 | Knock4 | Acc1 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | -356.7 | -356.7 | -0.122295719844443 | 0.0677431906613937 | 0.230988326848198 | 0.171400778210032 | 0.00914007782101169 |
| 2 | 0 | -354.4 | -354.4 | -0.162607003891136 | -0.0762256809339371 | -0.310334630350246 | 0.182918287937658 | -0.000315175097276265 |
| 3 | 0 | -353.4 | -353.4 | -0.0359143968872445 | 0.136848249027152 | -0.275782101167366 | -0.225953307393081 | 0.00661867704280157 |
| 4 | 0 | -353.1 | -353.1 | -0.185642023346388 | 0.0389494163423275 | 0.207953307392945 | 0.182918287937658 | -0.0120817120622568 |
| 5 | 0 | -348.7 | -348.7 | -0.024396887159618 | -0.128054474708256 | -0.212435797665421 | -0.272023346303587 | -0.00577821011673153 |
| 6 | 0 | -345.4 | -345.4 | -0.0301556420234312 | 0.131089494163339 | -0.275782101167366 | 0.309610894941549 | -0.0244785992217899 |
| 7 | 0 | -345.1 | -345.1 | 0.0907782101166466 | -0.168365758754949 | 0.213712062256758 | -0.231712062256894 | -0.00661867704280157 |
| 8 | 0 | -344.4 | -344.4 | -0.0359143968872445 | -0.09926070038919 | -0.223953307393047 | -0.329610894941719 | -0.00262645914396887 |
| 9 | 0 | -344.2 | -344.2 | -0.0589494163424974 | 0.119571984435713 | 0.219470817120571 | -0.335369649805532 | -0.0213268482490273 |
| 10 | 0 | -343.2 | -343.2 | -0.11653696498063 | -0.162607003891136 | 0.27129961089489 | 0.228988326848164 | -0.00766926070038911 |
| 11 | 0 | -342.4 | -342.4 | 0.165642023346210 | 0.444785002217084 | 0.202194552529132 | 0.244163424124420 | 0.0114513618677043 |

*Figure A.2 Screenshot of Google BigQuery database*

## A.3 Pulling Data from Cloud Database

The following diagram in Figure A.3 presents the data uploading process which was used in this research:

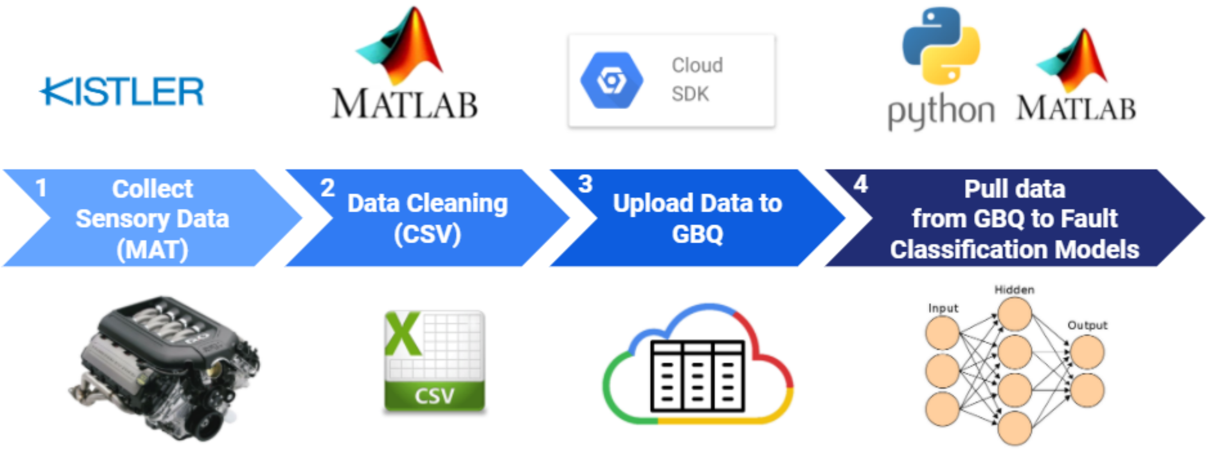


*Figure A.3 Data from engine to the cloud*

Python-based fault classification models, such as Deep Convolutional Neural Networks (CNNs), Deep Recurrent Neural Networks (RNNs) and Random Forest (RF) were designed to pull data from Google BigQuery (GBQ). The communications between Python and GBQ were all published and available online on the official website of Google Cloud Platform (GCP). However, pulling data from GBQ to MATLAB-based fault classification models, such as Extended Multi-Scale Principal Component Analysis (EMSPCA), was another challenge.

MathWorks Co. and Google Cloud Platform do not have communication protocols between their platforms. However, a GitHub repository which connects MATLAB to several cloud service providers was invented by other researchers [91]. The repository contains packages and instructions to connect MATLAB to Google Cloud Platform (GCP). Communications between

MATLAB and GCP were achieved as shown in Figure A.5 and GBQ tables were loaded into MAT-format. A few pre-requisites such as installing Git Desktop, JAVA, and Maven packages were required.

## *A.4 Cloud Execution of FDD Models on Virtual Machines (VM)*

Building a cloud database on GBQ was a great solution to the storage and data management issues on local computers. However, loading large data tables from GBQ back into MATLAB or Python to run FDD and classification models on local computers was not the best practice. Thus, virtual machines had to be created on Google Cloud Platform (GCP) to close the loop and conduct all communications on the cloud without needing local computers. Figure A.4 shows the workflow of the cloud-based model.

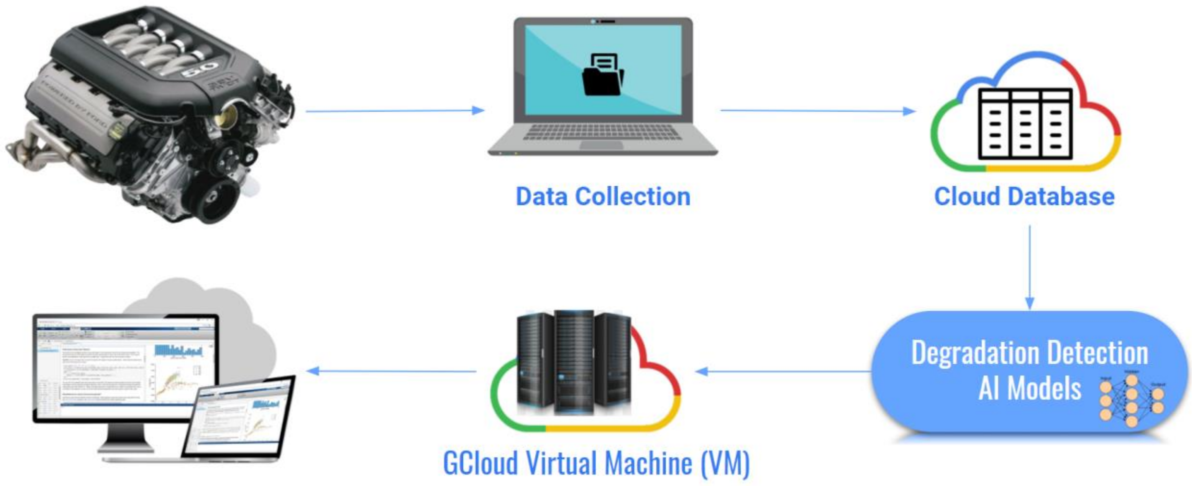
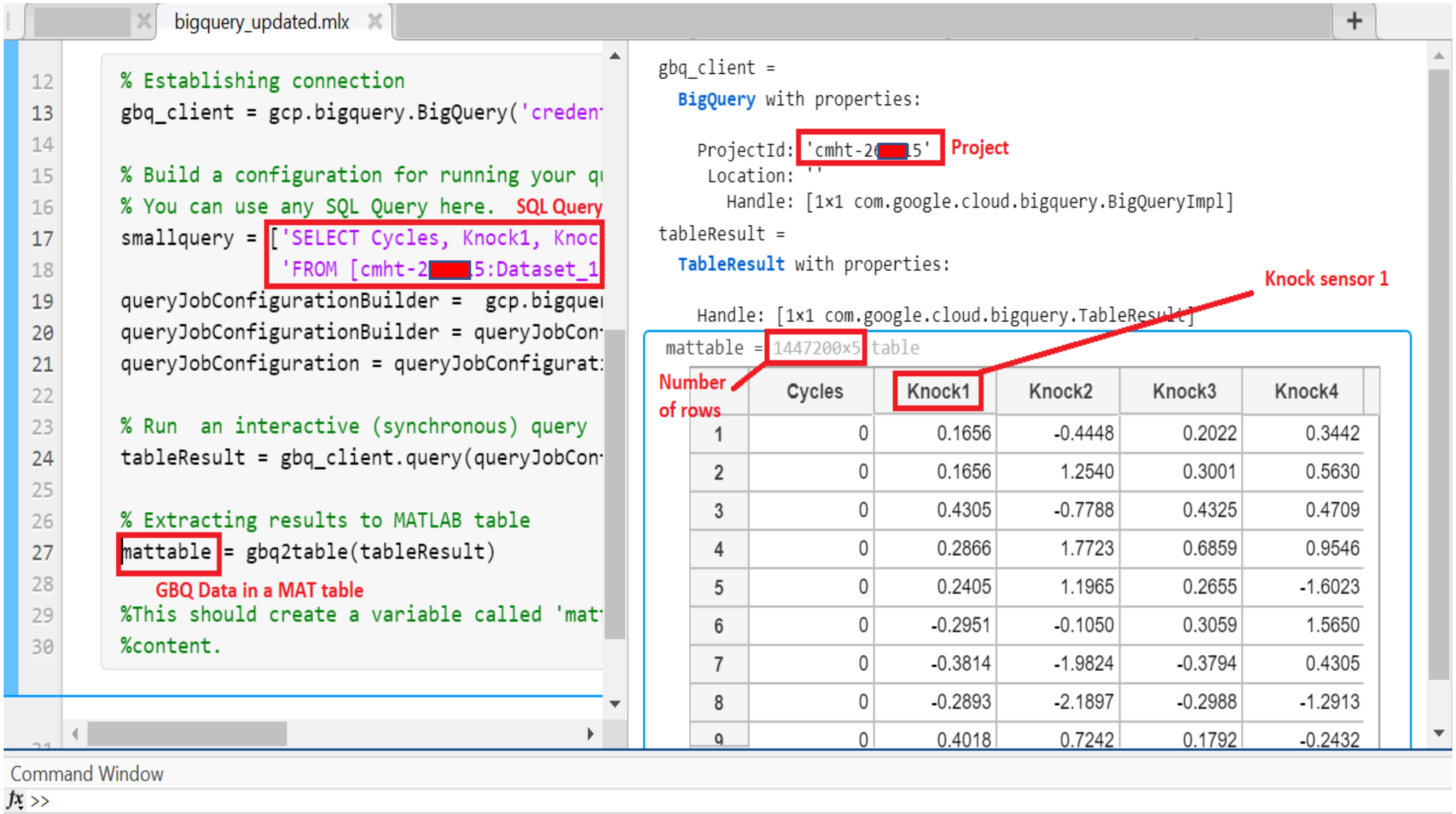


***Figure A.4 Data pipeline diagram***



```
% Establishing connection
gbq_client = gcp.bigquery.BigQuery('creden

% Build a configuration for running your qu
% You can use any SQL Query here.
smallquery = ['SELECT Cycles, Knock1, Knoc
              'FROM [cmht-2[redacted]5:Dataset_1
queryJobConfigurationBuilder =  gcp.bigque
queryJobConfigurationBuilder = queryJobCon
queryJobConfiguration = queryJobConfigurat

% Run  an interactive (synchronous) query
tableResult = gbq_client.query(queryJobCon

% Extracting results to MATLAB table
mattable = gbq2table(tableResult)

%This should create a variable called 'mat
%content.
```

SQL Query

GBQ Data in a MAT table

```
gbq_client =
  BigQuery with properties:

    ProjectId: 'cmht-2[redacted]5'
     Location: ''
       Handle: [1x1 com.google.cloud.bigquery.BigQueryImpl]
tableResult =
  TableResult with properties:

    Handle: [1x1 com.google.cloud.bigquery.TableResult]
mattable = 1447200x5 table
```

Project

Knock sensor 1

Number of rows

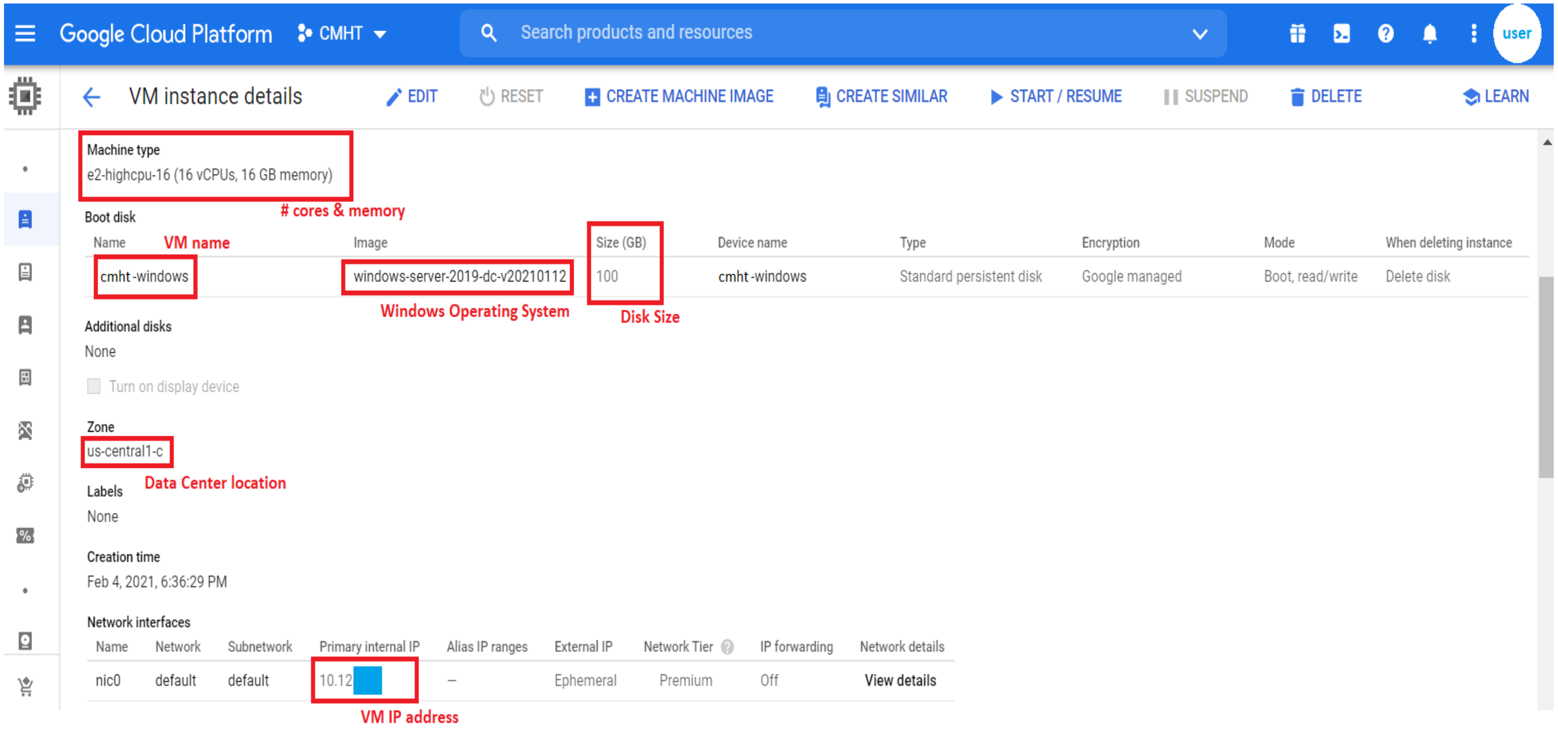
| | Cycles | Knock1 | Knock2 | Knock3 | Knock4 |
|---|---|---|---|---|---|
| 1 | 0 | 0.1656 | -0.4448 | 0.2022 | 0.3442 |
| 2 | 0 | 0.1656 | 1.2540 | 0.3001 | 0.5630 |
| 3 | 0 | 0.4305 | -0.7788 | 0.4325 | 0.4709 |
| 4 | 0 | 0.2866 | 1.7723 | 0.6859 | 0.9546 |
| 5 | 0 | 0.2405 | 1.1965 | 0.2655 | -1.6023 |
| 6 | 0 | -0.2951 | -0.1050 | 0.3059 | 1.5650 |
| 7 | 0 | -0.3814 | -1.9824 | -0.3794 | 0.4305 |
| 8 | 0 | -0.2893 | -2.1897 | -0.2988 | -1.2913 |
| 9 | 0 | 0.4018 | 0.7242 | 0.1792 | -0.2432 |

*Figure A.5 Loading GBQ table in Matlab*

Virtual machines (VM) are remote computers with fully customizable specifications which can be assembled and purchased on the cloud. VMs are physically created in Data Centers which are in specific countries around the world. Using Remote Desktop Connections, virtual machines can be accessed from any computer device, tablet, or even smart phone. The application on Google Cloud Platform (GCP) where virtual machines can be created and operated is called Google Compute Engine (GCE). Capacity of the Central Processing Unit (CPU), number of cores, disk size, memory and even operating system can all be adjusted based on the user needs. Even after creating and running the VM, most specifications can be changed according to current needs. VM operating system are available in Windows, Ubuntu, or Linux. Figure A.6 shows a screenshot of the virtual machine (VM) specifications.

All the FDD and classification models which were built in this thesis were executed on cloud virtual machines (VM) using Google Cloud Engine (GCE). This has enhanced the processing time and the ease of communications between the different stages of an FDD model. One of the best advantages of GCE is that disks can be copied to other virtual machines. This prevents the user from losing any installed packages or any stored data. However, communications are only allowed among disks and virtual machines in the same data center location. For example, disks which were created in a data center in a US zone can not be used in a VM in Asia.

Virtual machines give the user the flexibility to access their remote computers from any device anywhere in the world. This means that not only the cloud engine database can be easily shared with other researchers, but also the MATLAB-based or Python-based FDD and classification models. This can be done by sharing the VM credentials with the desired individual or institution. Figure A.7 shows a screenshot of a ML model running on a virtual machine (VM).



VM instance details

Machine type
e2-highcpu-16 (16 vCPUs, 16 GB memory)

Boot disk

| Name | Image | Size (GB) | Device name | Type | Encryption | Mode | When deleting instance |
|---|---|---|---|---|---|---|---|
| cmht-windows | windows-server-2019-dc-v20210112 | 100 | cmht-windows | Standard persistent disk | Google managed | Boot, read/write | Delete disk |

Additional disks
None

□ Turn on display device

Zone
us-central1-c

Labels
None

Creation time
Feb 4, 2021, 6:36:29 PM

Network interfaces

| Name | Network | Subnetwork | Primary internal IP | Alias IP ranges | External IP | Network Tier | IP forwarding | Network details |
|---|---|---|---|---|---|---|---|---|
| nic0 | default | default | 10.12[redacted] | – | Ephemeral | Premium | Off | View details |

*Figure A.6 Virtual machine specifications*

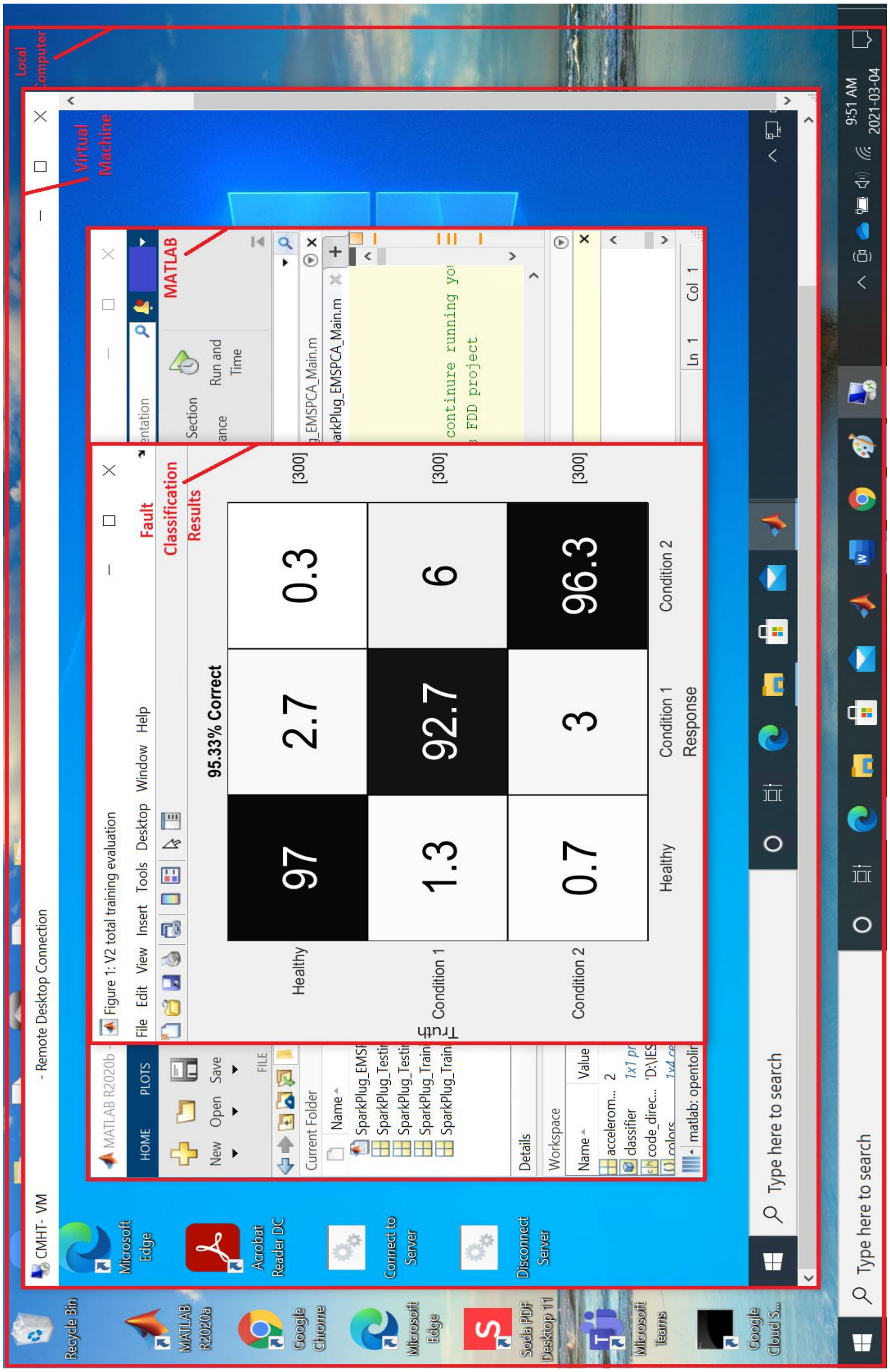


Figure 1: V2 total training evaluation

95.33% Correct

| Truth \ Response | Healthy | Condition 1 | Condition 2 | |
|---|---|---|---|---|
| Healthy | 97 | 2.7 | 0.3 | [300] |
| Condition 1 | 1.3 | 92.7 | 6 | [300] |
| Condition 2 | 0.7 | 3 | 96.3 | [300] |

*Figure A.7 Screenshot of a ML model on Virtual Machine (VM)*

# *References*

[1] R. Isermann, Fault-diagnosis systems: an introduction from fault detection to fault tolerance. Springer Science & Business Media, 2006.

[2] Feng Y., and Habibi S., “Fault Detection and Diagnosis of an Internal Combustion Engine”, Master’s thesis, McMaster University, Canada, 2016.

[3] V. Venkatasubramanian, R. Rengaswamy, S. N. Kavuri, and K. Yin, “A review of process fault detection and diagnosis: Part III: Process history-based methods,” Comput. Chem. Eng., vol. 27, no. 3, pp. 327–346, 2003.

[4] V. Venkatasubramanian, R. Rengaswamy, and S. N. Kavuri, “A review of process fault detection and diagnosis: Part II: Qualitative models and search strategies,” Comput. Chem. Eng., vol. 27, no. 3, pp. 313–326, 2003.

[5] You-Jin Park, Shu-Kai F. Fan, and Chia-Yu Hsu, “A Review on Fault detection and process diagnostics in Industrial Processes”, 2020.

[6] A. Dutka, H. Javaherian, and M. J. Grimble, “Model-based engine fault detection and isolation,” in 2009 American Control Conference, 2009, pp. 4593–4600.

[7] V. K. Rai and A. R. Mohanty, “Bearing fault diagnosis using FFT of intrinsic mode functions in Hilbert--Huang transform,” Mech. Syst. Signal Process., vol. 21, no. 6, pp. 2607–2615, 2007.

[8] Omar AlShorman, Muhammad Irfan, Nordin Saad, D. Zhen, Noman Haider, Adam Glowacz, Ahmad AlShorman, "A Review of Artificial Intelligence Methods for Condition

Monitoring and Fault Diagnosis of Rolling Element Bearings for Induction Motor", Shock andvibration, vol. 2020, ArticleID 8843759, 20 pages, 2020. https://doi.org/10.1155/2020 /8843759

[9] S. Habibi et al., "A Real Time Fault Detection and Diagnosis System for Automotive Applications," 2019.

[10] F. Kimmich, A. Schwarte, and R. Isermann, "Fault detection for modern Diesel engines using signal-and process model-based methods," Control Eng. Pract., vol. 13, no. 2, pp. 189–203, 2005.

[11] Paul M. andThomas B., "A Review on Google Self Driving Car Technology", International Journal of Engineering Research and Technology (IJERT), ISSN: 2278-0181, NSDMCC – 2015 Conference Proceedings, 2015.

[12] Shweta N. Dethe, Varsha S. Shevatkar, and R. P. Bijwe, "Google Driverless Car", 2016 IJERSET, vol. 2, Issue 2, Print ISSN: 2395-1990, Online ISSN: 2394-4099, 2016.

[13] Seddik E., and Habibi S., "Fault Detection and Identification of Vehicle Starters and Alternators using Machine Learning Techniques", Master's thesis, McMaster University, Canada, 2016.

[14] B. Samantha, "Gear fault detection using artificial neural networks and support vector machines with genetic algorithms," Mech. Syst. Signal Process., vol. 18, no. 3, pp. 625–644, 2004.

[15] Laftah, Rafil et al., "Neural networks of Engine Fault Diagnosis based on Exhaust gas analysis," Thi-Qar University Journal for Engineering Sciences, vol. 4, no. 1, 2013.

[16] Kiplimo R., and Ndiritu H., “Effect of Exhaust Gas Recirculation on Performance and Emission Characteristics of a Diesel-Piloted Biogas Engine”, January 2015, Smart Grid and Renewable Energy 06(04):49-58, DOI: 10.4236/sgre.2015.64005, License: CC BY 4.0

[17] T. P. Carvalho, F. A. Soares, R. Vita, R. da P. Francisco, J. P. Basto, and S. G. Alcalá, “A systematic literature review of machine learning methods applied to predictive maintenance,” Computer & Industrial Engineering, 106024, 2019.

[18] R. Ounanena, L. Oukhellou, D. Candusso, et al., “Estimation of fuel cell operating time for predictive maintenance strategies”, International Journal of Hydrogen Energy, vol. 35, Issue 15, August 2010, Pages 8022-8029.

[19] T. Praveenkumar, M. Saimurugan, P. Krishnakumar, and K. I. Ramachandran, “Fault Diagnosis of Automobile Gearbox Based on Machine Learning Techniques,” Procedia Engineering, vol. 97, 2014, Pages 2092-2098.

[20] Eriksson J., “Machine Learning for Predictive Maintenance on Wind Turbines”, Master’s thesis, Linkoping University, 2020.

[21] Chen K., Pashami S., Fan Y., Nowaczyk S. (2019) Predicting Air Compressor Failures Using Long Short Term Memory Networks. In: Moura Oliveira P., Novais P., Reis L. (eds) Progress in Artificial Intelligence. EPIA 2019. Lecture Notes in Computer Science, vol 11804. Springer, Cham. https://doi.org/10.1007/978-3-030-30241-2_50.

[22] Maashi M., Alwhibi N., Alamr F., et al., “Industrial Duct Fan Maintenance Predictive Approach Based on Random Forest”, 9th International Conference on Information Technology Convergence and Services (ITCSE 2020), 2020.

[23] Fausing Olesen, Jonas, and Hamid Reza Shaker. “Predictive Maintenance for Pump Systems and Thermal Power Plants: State-of-the-Art Review, Trends and Challenges.” Sensors (Basel, Switzerland) vol. 20,8 2425. 24 Apr. 2020, doi:10.3390/s20082425.

[24] Ran Y., Zhou X., Lin P., et al., “A Survey of Predictive Maintenance: Systems, Purposes and Approaches”, IEEE Communications Surveys & Tutorials vol. xx, no. xx, Nov. 2019. arXiv: 1912.07383v1, 2019.

[25] S. Hwang, J. Jeong and Y. Kang, "SVM-RBM based Predictive Maintenance Scheme for IoT-enabled Smart Factory," 2018 Thirteenth International Conference on Digital Information Management (ICDIM), 2018, pp. 162-167, doi: 10.1109/ICDIM.2018.8847132.

[26] Krenek J., Kuca K., Blazek P., et al., “Application of Artificial Neural Networks in Condition Based Predictive Maintenance”, Studies in Computational Intelligence, doi: 10.1007/978-3-319-31277-4-7.

[27] Uhlmann E., Pontes R., Geisert C., and Hohwieler E., “Cluster identification of sensor data for predictive maintenance in a Selective Laser Melting machine tool”, Procedia Manufacturing, vol. 24, 2018, Pages 60-65.

[28] S. Haykin, Cognitive Dynamic Systems, Cambridge University Press, 2012.

[29] D. J. Mackay, Information Theory Inference and Learning Algorithms, Cambridge: Cambridge University Press, 2003.

[30] S. Haykin, Neural Networks and learning Machines, Third Edition ed., New Jersey: Pearson, 2009.

[31] University of Toronto, "Coursera," [Online]. Available: www.coursera.org/learn/neural-networks.

[32] I. Velsiev, "A Deep Learning Tutorial: From Perceptron to Deep Networks," 2010. [Online]. Available: https://www.toptal.com/machine-learning/an-introduction-to-deep-learning-fromperceptrons-to-deep-networks.

[33] S. Theodoridis and K. Koutroumbas, Pattern Recognition, 4th edition ed., Vols. ISBN: 978-1- 59749-272-0, Elsevier, 2009.

[34] S. Haykin, Neural Networks: A Comprehensive Foundation 2nd, vol. ISBN:0132733501, New Jersey, USA: Prentice Hall PTR Upper Saddle River, 1998.

[35] M. A. Nielson, Neural Networks and Deep Learning, Determination Press, 2015.

[36] D. D. Team, "Deeplearning4j: Open-source distributed deep learning for the JVM," Apache Software Foundation License 2.0, [Online]. Available: http://deeplearning4j.org.

[37] Y. Benjio, J. Goodfellow and A. Courville, “Deep Learning”, Book in preparation for MIT Press, 2016.

[38] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," Neural Networks, vol. 61, no. arXiv:1404.7828v4 [cs.NE], pp. 85-117, 2015.

[39] Henry J. Kelly, “Gradient Theory of Optimal Flight Paths”, 1960.

[40] Arthur E. Bryson, “A gradient method for optimizing multi-stage allocation processes”, In Proceedings of the Harvard Univ. Symposium on digital computers and their applications, 1961.

[41] Rumelhart, and McClelland, “Parallel Distributed Processing”, ISBN: 9780262181204567, July 1986.

[42] Y. Bengio, "Learning Deep Architectures for AI," Foundations and Trends in Machine Learning, Vols. Vol.2, No.1, no. DOI: 10.1561/2200000006, p. 1–127, 2009.

[43] L. Gomes, "Machine-Learning Maestro Michael Jordan on the Delusions of Big Data and Other Huge Engineering Efforts," IEEE Spectrum, 2014.

[44] Parmar A., Katariya R., Patel V., “A Review on Random Forest: An Ensemble Classifier”, In: Hemanth J., Fernando X., Lafata P., Baig Z. (eds) International Conference on Intelligent Data Communication Technologies and Internet of Things (ICICI) 2018, ICICI 2018, Lecture Notes on Data Engineering and Communications Technologies, vol 26. Springer, Cham. https://doi.org/10.1007/978-3-030-03146-6_86, 2019.

[45] Goel E., Ablhilasha Er., “Random Forest: A Review”, IJARCSSE, DOI: 10.23956/IJARCSSE/V7|1/01113, 2017.

[46] Gilles Louppe, “Understanding Random Forests: from theory to practice”, PhD Dissertation, University of Liege, arXiv:1407.7502v3, 2015.

[47] S. Bernard, S. Adam and L. Heutte, "Using Random Forests for Handwritten Digit Recognition," Ninth International Conference on Document Analysis and Recognition (ICDAR 2007), 2007, pp. 1043-1047, doi: 10.1109/ICDAR.2007.4377074.

[48] N Azizah, L. S. Riza, Y. Wihardi, "Implementation of random forest algorithm with parallel computing in R", Journal of Physics Conference Series 1280:022028, doi: 10.1088/1742-6596/1280/2/022028

[49] Wu Y., He J., Ji Y., et al., "Enhanced Classification Models for Iris Dataset", 7th International Conference on Information Technology and Quantitative Management (ITQM 2019), Procedia Computer Science 162 (2019) pp 946-954, 2019.

[50] R. A. Fisher, "Iris Data Set", 1936. (Online): https://archive.ics.uci.edu/ml/datasets/iris.

[51] Mei J., He D., Harley R.G., et al., "A random forest method for real-time price forecasting in New York electricity market", 2014 IEEE Power & Energy Society General Meeting, doi: 10.1109/PESGM.2014.6939932, 2014.

[52] Dudek G., "Short-Term Load Forecasting Using Random Forests", Advances in Intelligent Systems and Computing book series, AISC, vol. 23, pp 821-828. Doi: 10.1007/978-3-319-1130-4_71.

[53] Buchwitz B., Falkenberg A., and Kusters U., "Time-Series Event Forecasting in Consumer Electronic Markets using Random Forests", Proceedings of the 2019 Pre-ICIS SIGDSA Symposium, 2019.

[54] Rodriguez J., and Kuncheva L., "Time series classification: Decision forests and SVM on Interval and DTW features", doi:10.1.1.73.8695, 2007.

[55] Benjamin Goehry, Hui Yan, Yannig Goude, Pascal Massart, and Jean-Michel Poggi, "Random Forests for Time Series", 2021. hal-03129751.

f[56] Liu Y., Wang Y., and Zhang J., "New Machine Learning Algorithm: Random Forest", International Conference on Information Computing and Applications, ICICA 2012. Lecture Notes in Computer Science, vol 7473. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-34062-8_32

[57] Chris Smith, and Mark Koning, "Decision Trees and Random Forest: A visual introduction for beginners", October 2017.

[58] Genuer, Robin, Poggi, and Jean-Michel, "Random Forests with R", Springer International Publishing, doi: 10.1007/978-3-030-56485-8, 2020.

[59] Titanic Data Set, "Kaggle". Online: https://www.kaggle.com/c/titanic.

[60] Sklearn Library, Python programming language. Online: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[61] M. Ismail, "Industrial Extended Multi-Scale Principal Components Analysis for fault detection and diagnosis of car alternators and starters", M.Sc. Thesis, McMaster University, 2015.

[62] M. Ismail, Habibi S., and Ziada A., "An enhanced system and method for conducting PCA analysis on data signals", Canadian Intellectual Property Office, Patent: CA2965340A1, 2017.

[63] Stein S. A. M., Loccisano A., et al., “Chapter 13 Principal Components Analysis: Review of its Application on Molecular Dynamics Data”, Annual Reports in Computational Chemistry, vol. 2, pp 233-261, 2006.

[64] S. Zhang, S. Zhang, B. Wang, and T. G. Habetler, “Deep Learning Algorithms for Bearing Fault Diagnostics – A Comprehensive Review,” IEEE, vol. 3, arXiv:1901.08247, 2020.

[65] F. Jia, Y. Lei, L. Guo, J. Lin, and S. Xing, “A neural network constructed by deep learning technique and its application to intelligent fault diagnosis of machines,” Neurocomput., vol. 272, pp. 619–628, 2018.

[66] Alex Sherstinsky, “Fundamentals of Recurrent Neural Networks (RNN) and Long Short-term Memory (LSTM) Network”, ELSEVIER Physica D Nonlinear Phenomena 404(8): 132306, doi: 10.1016/j.physd.2019.132306, 2020.

[67] Yu Y., Si X., Hu S., and Zhang J., “A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures”, Neural Computation 2019, vol. 31, issue 7, doi: 10.1162/neco_a_01199, pp 1235-1270, 2019.

[68] Zhang S., and Wang B., “Deep Learning Algorithms for Bearing Fault Diagnosis – A comprehensive review”, arXiv:1901.08247v3, 2020.

[69] H. Pan, X. He, S. Tang, F. Meng, “An improved bearing fault diagnosis method using one-dimensional CNN and LSTM,” J. Mech. Eng., vol. 64, no. 7–8, pp. 443–452, 2018.

[70] H. Jiang, X. Li, H. Shao, and K. Zhao, “Intelligent fault diagnosis of rolling bearings using an improved deep recurrent neural network,” Meas. Sci. Technol., vol. 29, no. 6, art. no 065107, May 2018.

[71] Rui Yang, Mengjie Huang, Qidong Lu, Maiying Zhong, “Rotating Machinery Fault Diagnosis Using Long-short-term Memory Recurrent Neural Network”, IFAC-Papers Online, Volume 51, Issue 24, 2018, Pages 228-232, ISSN 2405-8963, https://doi.org/10.1016/j.ifacol.2018.09.582.

[72] Dizhi Long, Xin Wen, Junhong Wang, Bingyi Wei, "A Data Fusion Fault Diagnosis Method Based on LSTM and DWT for Satellite Reaction Flywheel", Mathematical Problems in Engineering, vol. 2020, ArticleID 2893263, 15 pages, 2020. https://doi.org/10.1155/2020/2893263

[73] Ashok Tak, “Deep Learning based Fault Diagnosis in Transmission Lines via Long Short-term Memory Networks”, Master’s thesis, University of Toronto, 2021.

[74] Yixin Huangfu, Seddik E., Habibi S., and Wassyng A., “Fault Detection and Diagnosis of Spark plugs using Deep Learning Techniques”, SAE Int. Journal of Engines Mechatronics, 2021.

[75] Liang X., Ge Z., et al., “LSTM with Wavelet Transform Based Data Preprocessing for Stock Price Prediction”, Mathematical Problems in Engineering, vol. 2019, https://doi.org/10.1155/2019/1340174, Article ID 1340174, 2019.

[76] Jarrah M., and Salim N., “A Recurrent Neural Network and a Discrete Wavelet Transform to predict the Saudi stock price trends”, International Journal of Advanced Computer Science and Applications 10(4), DOI: 10.14569/IJACSA.2019.0100418, 2019.

[77] Ghosh A., Sufian A., Sultana F., et al., "Fundamental Concepts of Convolutional Neural Network", Recent trends and Advances in Artificial Intelligence and Internet of Things. DOI: 10.1007/978-3-030-32644-9_36, 2020.

[78] Yadav R., "Light-weighted CNN for Text Classification", arXiv: 2004.07922v1, 2020.

[79] Amin M., and Nadeem N., "Convolutional Neural Network: Text Classification Model for Open Domain Question Answering System", https://arxiv.org/ftp/arxiv/papers/1809/1809.02479.

[80] Michael Nielsen, "Neural Networks and Deep Learning", 2019. Online: http://neuralnetworksanddeeplearning.com/.

[81] Serradilla O., Zugasti E., and Zurutuza U., "Deep learning models for predictive maintenance: a survey, comparison, challenges and prospect", arXiv: 2010.03207v1, 2020.

[82] Khorram A., Khalooei M., and Rezghi M., "End-to-end CNN+LSTM Deep Learning Approach for Bearing Fault Diagnosis", Electrical Engineering and Systems Science, Signal Processing, doi: 10.1007/s10489-020-01859-1, arXiv: 1909.070801, 2020.

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

[84] C. Lu, Z. Wang, and Bo Zhou, "Intelligent fault diagnosis of rolling bearing using hierarchical convolutional network-based health state classification," Adv. Eng. Informat., vol. 32, pp. 139–157, 2017.

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

[86] L. Wen, X. Li, L. Gao and Y. Zhang, “A new convolutional neural network-based data-driven fault diagnosis method,” IEEE Trans. Ind. Electron., vol. 65, no. 7, pp. 5990–5998, July 2018.

[87] W. Zhang, F. Zhang, W. Chen, Y. Jiang and D. Song, “Fault state recognition of rolling bearing based fully convolutional network,” Comput. in Sci. & Eng., vol. PP, no. PP, pp. PP–PP, 2018.

[88] Z. Zhuang and Q. Wei, “Intelligent fault diagnosis of rolling bearing using one-dimensional multi-scale deep convolutional neural network-based health state classification,” in Proc. 2018 IEEE 15th Int. Conf. Netw., Sens. & Control (ICNSC), Zhuhai, China, 2018, pp. 1–6.

[89] W. Zhang, C. Li, G. Peng, Y. Chen, Z. Zhang, “A deep convolutional neural network with new training methods for bearing fault diagnosis under noisy environment and different working load,” Mech. Syst. Signal Process., vol. 100, pp. 439–458, 2018.

[90] Geotab corporation (09/04/2021). https://geotab.github.io/sdk/software/guides/go-device-logging/

[91] Github. Online: https://github.com/mathworks-ref-arch/mathworks-gcp-support.