COMPREHENSIVE COMPARISON OF DIFFERENT EARLY BEARING FAULT DETECTION TECHNIQUES

by
MOHAMED ELARABY ABDOU SOLIMAN ELGALLAD

Submitted to the Graduate School of of Engineering and Natural Sciences in partial fulfilment of the requirements for the degree of Master of Science

Sabancı University
July 2024

ABSTRACT

COMPREHENSIVE COMPARISON OF DIFFERENT EARLY BEARING FAULT DETECTION TECHNIQUES

MOHAMED ELARABY ABDOU SOLIMAN ELGALLAD

MECHATRONICS ENGINEERING M.Sc. THESIS, JULY 2024

Thesis Supervisor: Assoc. Prof. Kemalettin Erbatur

Keywords: Bearing Fault Detection, Optimization, SVM, Deep Learning, BiLSTM

Bearing fault detection is a part of predictive maintenance for rotating machinery to provide early warnings of pending breakdowns, preventing sudden stops in production. This study presents two advanced methods for bearing fault detection utilizing the Case Western Reserve University (CWRU) and the HUST Bearing Datasets: Support Vector Machines (SVMs) optimized by Grey Wolf Optimization (GWO), Particle Swarm Optimization (PSO), and the novel Kepler Optimization Algorithm (KOA), and a deep learning approach using Bidirectional Long Short-Term Memory (BiLSTM) networks.

The SVM parameters, box constraint and kernel scale were tuned with GWO, PSO, and KOA to improve fault detection efficiency. These results were compared with those of a BiLSTM-based deep learning model. Our comparison showed that the BiLSTM model significantly outperformed the optimized SVM m odels. Although the optimized SVMs achieved considerable improvements over non-optimized SVM models in fault detection accuracy, they were still inferior to the BiLSTM model.

Evaluated based on accuracy, the BiLSTM model consistently performed outstandingly across different fault types and sizes, reaching 100% accuracy on small fault sizes, and accuracies as high as 99.92% on bigger ones on the CWRU Dataset and accuracies as high as 99.58% on the HUST Dataset. The proposed model outperformed several modern models regularly utilized for bearing fault detection. This research highlights the potential of deep learning techniques, specifically BiLSTM, in bearing fault detection, demonstrating their advantage over traditional machine

learning models even when optimized with advanced algorithms. This study adds value to the field by showcasing the capabilities of deep learning to enhance predictive maintenance systems.

ÖZET

FARKLI RULMAN ERKEN ARIZA TESPİT TEKNİKLERİNİN KAPSAMLI KARŞILAŞTIRILMASI

MOHAMED ELGALLAD

Mekatronik Mühendisliği YÜKSEK LİSANS TEZİ, TEMMUZ 2024

Tez Danışmanı: Doç. Dr. Kemalettin Erbatur

Anahtar Kelimeler: Rulman arıza tespiti, Optimizasyon, Destek Vektör Makineleri (SVM), Derin Öğrenme, Çift Yönlü Uzun Kısa Süreli Bellek (BiLSTM)

Rulman arıza tespiti, dönen makineler için öngörücü bakımın bir parçası olup, beklenen arızalara ilişkin erken uyarılar sağlayarak üretimdeki ani duruşları önler. Bu çalışma, Case Western Reserve Üniversitesi (CWRU) ve HUST rulman veri kümelerini kullanarak rulman arıza tespiti için iki gelişmiş yöntem sunmaktadır: i) Grey Wolf Optimizasyonu (GWO), Parçacık Sürü Optimizasyonu (PSO) ve yeni Kepler Optimizasyon Algoritması (KOA) ile optimize edilmiş Destek Vektör Makineleri (SVM'ler) ve ii) Çift Yönlü Uzun Kısa Süreli Bellek (BiLSTM) ağlarını kullanan derin bir öğrenme yaklaşımı.

Arıza tespit verimliliğini artırmak için SVM kutu kısıtlaması ve çekirdek ölçeği parametreleri GWO, PSO ve KOA ile ayarlandı. Bu sonuçlar BiLSTM tabanlı derin öğrenme modelinin sonuçlarıyla karşılaştırıldı. Karşılaştırmamız BiLSTM modelinin optimize edilmiş SVM modellerinden önemli ölçüde daha başarılı olduğunu gösterdi. Optimize edilmiş SVM'ler, optimize edilmemiş SVM modellerine göre hata tespit doğruluğu açısından önemli gelişmeler elde etmesine rağmen, hala BiLSTM modelinden daha düşük performans gösterdiler.

Doğruluğa dayalı olarak değerlendirilen BiLSTM modeli, farklı hata türleri ve hata boyutlarında sürekli olarak üstün performans göstererek küçük hata boyutlarında %100 ve daha büyük olanlarda %99,92'ye varan doğruluğa ulaşmıştır. HUST Veri Kümesi'nde doğruluk oranı %99,58'e kadar çıkmaktadır. Önerilen model, rulman arıza tespiti için düzenli olarak kullanılan birçok son teknoloji ürünü modelden daha

iyi performans göstermiştir. Bu araştırma, rulman arızası tespitinde derin öğrenme tekniklerinin, özellikle de BiLSTM'nin potansiyelini vurgulamaktadir. Tez, gelişmiş algoritmalarla optimize edildikleri durumda dahi geleneksel makine öğrenimi modellerine göre BiLSTM'nin avantajlarını ortaya koymakta, derin öğrenmenin tahminsel bakım sistemlerini geliştirmedeki yeteneklerini sergileyerek alana değer katmaktadır.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my thesis advisor, Assoc. Prof. Kemalettin Erbatur, for his invaluable guidance, feedback, and unwavering support throughout this research. I would also like to thank Asst. Prof. Melih Turkseven and Assoc. Prof. Ahmet Onat for honoring me by judging my work. Finally, I want to extend my gratitude to all my professors and mentors from whom I have been fortunate to learn so much throughout my academic years.

I dedicate this thesis to my wife, without whom it would not have been possible, and to my parents and sister, who were always by my side.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

For an industry to flourish, bearing fault detection in mechanical machinery is obligatory, Holm-Hansen & Gao (2000). Bearing fault detection is a critical component in the maintenance and monitoring of rotating machinery, essential for ensuring operational reliability and preventing unexpected failures. Bearings are vital elements in machinery, facilitating smooth rotation and load-bearing capabilities. The importance of bearing fault detection cannot be exaggerated. In recent decades, the modern industry has increasingly incorporated critical and advanced machinery across various sectors, including power generation, aviation, oil and gas, chemicals, and manufacturing, Hui, Ooi, Lim & Leong (2016). In these industries, the failure of a single bearing can halt entire production lines, cause equipment to operate inefficiently, or lead to safety risks. Early detection of bearing faults allows for timely repairs and replacement, hence minimizing downtime and financial losses, Youcef Khodja, Guersi, Saadi & Boutasseta (2020), and preventing secondary damage to other machine components. This thesis explores two advanced methodologies for bearing fault detection: Support Vector Machines (SVM) optimized using Grey Wolf Optimization (GWO), Mirjalili, Mirjalili & Lewis (2014), Particle Swarm Optimization (PSO), Kennedy & Eberhart (1995), and the novel Kepler Optimization Algorithm (KOA), Abdel-Basset, Mohamed, Azeem, Jameel & Abouhawwash (2023), and deep learning with Bidirectional Long Short-Term Memory networks (BiLSTMs), Graves & Schmidhuber (2005).

Traditional maintenance strategies often relied on scheduled maintenance or reactive maintenance, which are not always very efficient. Scheduled maintenance may result in unnecessary downtime and costs, while reactive maintenance frequently resulted in unexpected failures and extensive repairs, Ran, Zhou, Lin, Wen & Deng (2019). These limitations emphasized the need for predictive maintenance techniques that can monitor the condition of bearings in real-time and predict faults long before they lead to failures.

1.1 Methods of Bearing Fault Detection

Acoustic Emission Analysis detects the high-frequency sound waves generated by bearing defects, Gholizadeh, Leman, Baharudin & others (2015). This method is sensitive to early-stage faults that may not yet produce significant vibration. By capturing and analyzing these acoustic signals, faults can be detected at an early stage.

Thermal Imaging involves using infrared cameras to detect temperature variations in the bearing, Azeez, Alkhedher & Gadala (2020). Faulty bearings often generate excessive heat due to increased friction. Thermal imaging can visualize these hot spots, providing a non-invasive way to detect bearing faults.

Electrical Signature Analysis monitors changes in the electrical signals of motors driving the bearings, Salomon, Ferreira, Sant'Ana, Lambert-Torres, Borges da Silva, Bonaldi, de Oliveira & Torres (2019). Faults in bearings can cause variations in current and voltage, which can be analyzed to detect anomalies.

Vibration Analysis is one of the most widely used techniques for machinery fault detection. It involves measuring the vibrations produced by a machine and analyzing these signals to identify abnormal patterns indicative of faults, Aherwar (2012). Vibration Analysis is the method of bearing fault detection used throughout this thesis.

1.2 Techniques of Bearing Fault Detection

Recent innovations in technology have led to the development of advanced methods for bearing fault detection. These methods include machine learning and deep learning techniques, which can analyze large amounts of data and detect complex patterns associated with bearing faults.

Machine learning algorithms, such as Support Vector Machines (SVM), Artificial Neural Networks, and K-Nearest Neighbors (K-NN), have been applied to bearing fault detection. These algorithms can be trained on labeled datasets to classify different types of bearing faults and predict their occurrence based on input data

such as vibration signals, Zhang, Zhang, Wang & Habetler (2020).

Deep learning, a subset of machine learning, employs neural networks with multiple layers to model complex relationships in data. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) networks, have shown great promise in bearing fault detection, Zhang et al. (2020). These models can automatically extract features from raw data and capture temporal dependencies, making them very effective for analyzing time-series data such as vibration signals.

The traditional machine learning algorithm, Support Vector Machines and the more advanced deep learning algorithm, Bidirectional Long Short-Term Memory, are both explored in this thesis.

1.3 Types of Bearing Faults

Three common classifications of bearing faults are inner race, outer race, and ball faults. As the names suggest, inner race faults refer to bearing faults on the inner bearing, while outer race faults refer to bearing faults on the outer bearing. Ball faults refer to faults on the rolling elements of the bearing. Figure 1.1 illustrates a typical bearing and Figure 1.2 illustrates the three common types of bearing faults mentioned above.

In addition to detecting the presence of a bearing fault, it is equally important to be able to classify the type of fault to quicken the process of fixing the fault. For this reason, the Case Western Reserve University (CRWU) Bearing Dataset, Wu (2024), and the HUST Bearing Dataset, Thuan & Hong (2023), were used in this thesis to appraise the performance of the presented models (the HUST Bearing Dataset was only used with the deep learning model). The Bearing Datasets contain accelerometer vibration data collected from non-faulty and faulty bearings.

1.4 Thesis Organization

The rest of this thesis is organized as follows: Chapter 2 reviews some related work. Following that, Chapter 3 presents our motivation for this thesis. Next, Chapter 4 covers our datasets and methodology. The results are presented in Chapter 5, and Chapter 6 concludes the thesis and highlights potential future work.

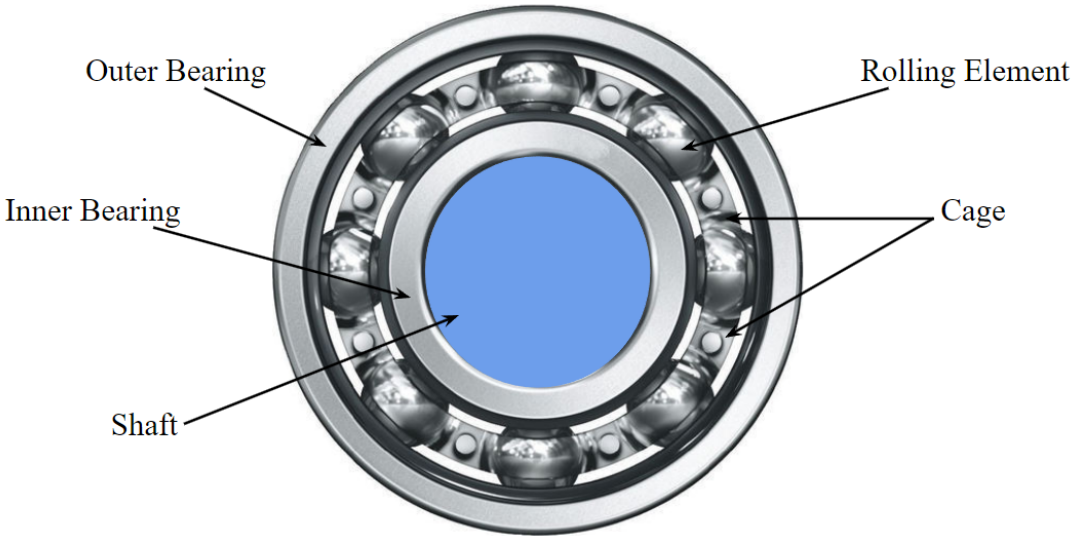


Figure 1.1 Typical Bearing

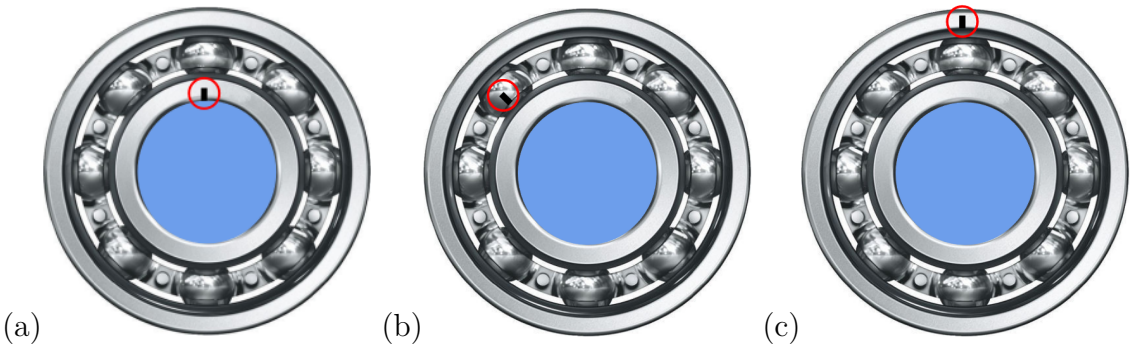


Figure 1.2 Fault Types: (a) Inner Race Fault (b) Ball Fault (c) Outer Race Fault

2. LITERATURE REVIEW

Bearing fault detection has been a hot topic for many years. With the recent rising popularity of predictive maintenance in Industry 4.0, Zonta, Da Costa, da Rosa Righi, de Lima, da Trindade & Li (2020), a lot of research has been conducted to apply predictive maintenance to bearing fault detection. Machine learning, as well as deep learning, are among the most transformative technologies in artificial intelligence. Their popularity has surged recently due to their capacity to predict outcomes, analyze vast amounts of data, and deliver insights that were once unattainable, Sharifani & Amini (2023). Naturally, since predictive maintenance relies mainly on analysing large amounts of data and detecting failures, machine learning and deep learning techniques have been widely applied to bearing fault detection.

2.1 Machine Learning In Bearing Fault Detection

Several Machie Learning techniques have been widely deployed for bearing fault detection. Schoen, Lin, Habetler, Schlag & Farag (1995) used Artifical Neural Networks (ANNs) to learn the characteristics of a good motor and used the learned model to detect faulty motors by comparing their characteristics with those of the good motor. Li, Chow, Tipsuwan & Hung (2000) used ANNs to detect bearing faults based on bearing vibration frequency features. Kanai, Desavale & Chavan (2016) combined ANNs with Mobel-based estimation (MBE) to diagnose faults of deep groove ball bearings. Li, Su, Wu, Wang & Chen (2017) diagnosed bearing faults using a method found on Kernel Extreme Learning Machine and Variational Mode Decomposition. Fei (2017) proposed an SVM model coupled with wavelet packet transform-phase space reconstruction-singular value decomposition for bearing fault diagnosis. Shen, Xiao, Wang & Song (2023) used Support Vector Machines com-

bined with the optimization algorithms Grey Wolf Optimization and Particle Swarm Optimization to detect and classify bearing faults using the CRWU bearing dataset. Yang, Hu & Zhang (2020) also used SVMs combined with IMF sample entropy and PSO to effectively identify rolling bearing faults. Hadden & Hadi (2023) combined Wavelet Analysis and K-Means Clustering for early fault detection in roller element bearings.

2.2 Deep Learning In Bearing Fault Detection

In addition to the many machine learning algorithms used, deep learning models also had their fair share of applications in bearing fault detection. Two of the most commonly used deep learning models are Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). Lu, Wang & Zhou (2017) applied intelligent rolling bearing diagnosis using hierarchical CNN-based health state classification. Hoang & Kang (2019) used CNNs to classify bearing faults based on vibrational signals. Wen, Li, Gao & Zhang (2017) proposed a novel CNN based on LeNet-5 for diagnosing bearing faults after transforming the signals into 2-D images. Zhang, Yi, Liang, Hongli, Xin & Hongliang (2020) also converted raw signals into two-dimensional images and used a CNN to classify the different types of bearing faults. Liu, Zhou, Zheng, Jiang & Zhang (2018) proposed a novel method for bearing fault detection using RNN in the form of an autoencoder. Shenfield & Howarth (2020) developed an intelligent fault diagnosis method using a novel dual-path RNN with a wide first kernel and deep CNN pathway capable of diagnosing bearing faults using real-time data streams. Chen, Zhang & Gao (2021) proposed an automatic feature learning neural network followed by an LSTM network to identify the type of fault.

2.3 Contributions of Thesis

While machine learning and deep learning techniques have been widely utilized for bearing fault detection, the vast majority of the proposed methods use some sort of preprocessing on the data before passing it to the model. What sets this thesis apart from the majority of the related work, is that no preprocessing is performed

on the data used in the proposed models. The advantage of this is that it simplifies the integration of the proposed model into any industrial setup where bearing fault detection is desired. The only inputs required by the model are the raw vibrational data from the accelerometers installed on the machines within close proximity to the bearings. The proposed model proved to be extremely accurate in detecting and classifying bearing faults, reaching an accuracy of 100% on small fault sizes and accuracies above 99.8% on bigger fault sizes. The proposed model reached an accuracy of 99.62% when all the fault sizes were combined.

3. MOTIVATION

In this chapter, we present the motivation for this thesis. The need for bearing fault detection is ever-growing due to the bloom in the number of machines used in every industry worldwide, from small workshops to large-scale factories. According to statistics, bearing failures are responsible for 40% of motor failures, Hakim, Omran, Ahmed, Al-Waily & Abdellatif (2023). A faulty bearing leads to the failure of the machine in which it is installed, which in turn leads to undesired outcomes. If a faulty bearing is not detected early, the failure of the machine cannot be prevented. This results in costly repairs and downtime. Generally, replacing a faulty bearing before complete breakdown costs significantly less time and money when compared with repairing the machine after it has broken down because the faulty bearing can lead to further complications within the machine. In cases where the bearing fault is not detected early and the machine completely fails, one might need to replace multiple parts of the machine, if not the whole machine.

Costly repairs and significant downtime are not the only reasons for the increasing interest in detecting faulty bearings early. Another equally, if not more, important reason is safety, Selcuk (2017). When a machine has a faulty bearing, it cannot be determined when and if it will break down. In many machines, especially large ones, this can pose a serious risk to the safety of the machines' operators. The machines could break down while an operator is handling them or while workers are in their proximity, which can result in serious injuries or even death. Therefore, identifying a faulty bearing ahead of a machine's failure is crucial for the well-being of the workers.

In addition to the reasons outlined above, the life of machines can be greatly extended with timely bearing fault detection. If a faulty bearing is detected and replaced before complete machine failure, the damage done by the bearing on the machine, due to the increased friction and repeated high-temperature contact, can be minimized, Xiao, Yang & Yang (2023). Moreover, the efficiency of the machines will increase because the increased friction also leads to higher energy

consumption by the machine. Finally, even if a machine still works, faulty bearings can lead to the production of subpar products. Therefore, detecting faulty bearings early helps maintain the expected quality of the products produced by the machines.

It is clear from the previous paragraphs that early bearing fault detection cannot just be an option, indeed it is a must. It needs to be easily available for anyone who wishes to avoid all the complications caused by faulty bearings. The objective of this thesis is to find a model that can accurately detect and classify bearing faults into inner race, outer race, and ball faults, using raw unprocessed vibration data from accelerometers. To do this, this thesis compares the performance of the machine learning model, Support Vector Machines (SVMs), with parameters optimized by the optimization algorithms GWO, PSO, and the novel KOA, and the deep learning model, Bidirectional Long Short-Term Memory (BiLSTM). Additionally, this thesis aims to come up with a set of hyper-parameters for the BiLSTM model that maximize the accuracy of detecting and classifying faulty bearings. The fact that the proposed model uses raw vibration data means that it is readily available for anyone who wishes to use it without the need for any kind of data preprocessing.

4. APPROACH AND METHODS

This chapter presents the approach and methodology of the thesis. In section 4.1, the Case Western Reserve University (CWRU) Bearing Dataset is inspected closely. Section 4.2 describes the HUST Bearing Dataset used for the BiLSTM model's performance verification. Sections 4.3 and 4.4 present Support Vector Machines (SVMs) and the three optimization algorithms: Grey Wolf Optimization Algorithm (GWO), Particle Swarm Optimization Algorithm (PSO), and Kepler Optimization Algorithm (KOA). In section 4.5, the deep learning models LSTM and BiLSTM are presented. Section 4.6 includes some definitions for the hyper-parameters that will be tuned for the BiLSTM model. The codes for the data preparation process, SVM models, and the deep learning model can be found in Appendix A.

4.1 The Case Western Reserve University Dataset

The Case Western Reserve University (CWRU) Bearing Data Center's datasets are used in this thesis to evaluate the models' performances in detecting and classifying bearing faults. The datasets contain accelerometer vibration data for normal and faulty bearings. The faulty bearings are of three kinds: inner race faults, outer race faults, and ball faults. Experiments were conducted using a 2 hp Reliance Electric motor, and acceleration data was measured at both the Drive End and the Fan End. The vibration data was collected using a 16 channel DAT recorder. The Drive End data was collected at two frequencies: 12 kHz and 48 kHz. The Fan End data was collected at 12 kHz only.

Motor bearings were intentionally damaged by electro-discharge machining (EDM). Defects with diameters between 0.007 inches and 0.028 inches were created individually on the inner raceway, outer raceway, and ball (rolling element). The

faulted bearings were then reinstalled into the test bench, and vibration data was collected under motor loads ranging from 0 to 3 horsepower. Throughout this thesis, a "dataset" would refer to a single column vector of accelerometer data that has a unique combination of these 5 characteristics:

- Location of the accelerometer:
  - Drive End
  - Fan End
- Frequency at which the data was collected:
  - 12 kHz
  - 48 kHz
- Fault type:
  - No Fault
  - Inner Race Fault
  - Ball Fault
  - Outer Race Fault
- Size of fault:
  - 0.007 inches
  - 0.014 inches
  - 0.021 inches
  - 0.028 inches (this fault size does not contain outer race faults)
- Motor load:
  - 0 Horsepower
  - 1 Horsepower
  - 2 Horsepower
  - 3 Horsepower

Throughout this thesis, the datasets used have the 3 common characteristics: accelerometer at the Drive End, data collected at 12 kHz, and 0 Horsepower motor

load. All 4 fault types and 4 fault sizes are used. Figure 4.1 shows the test stand used to collect the data and Figure 4.2 shows a simplified drawing of it. Table 4.1 displays the size specifications (in inches) of the bearings used for the data collection.

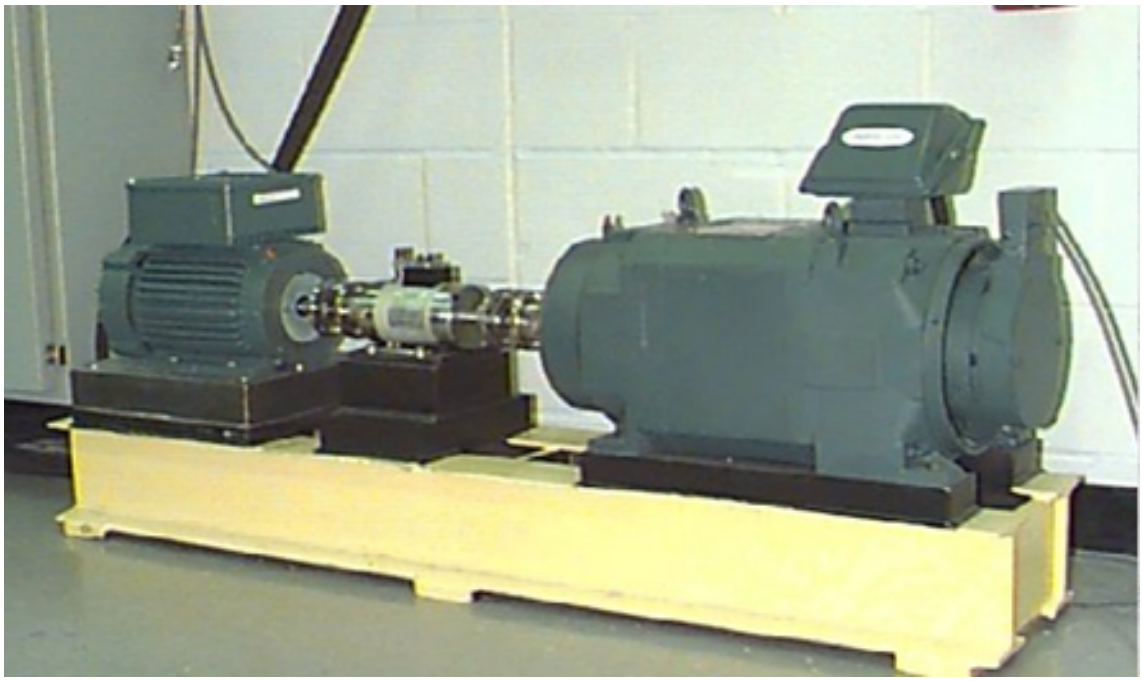


Figure 4.1 CWRU Bearing Test Stand

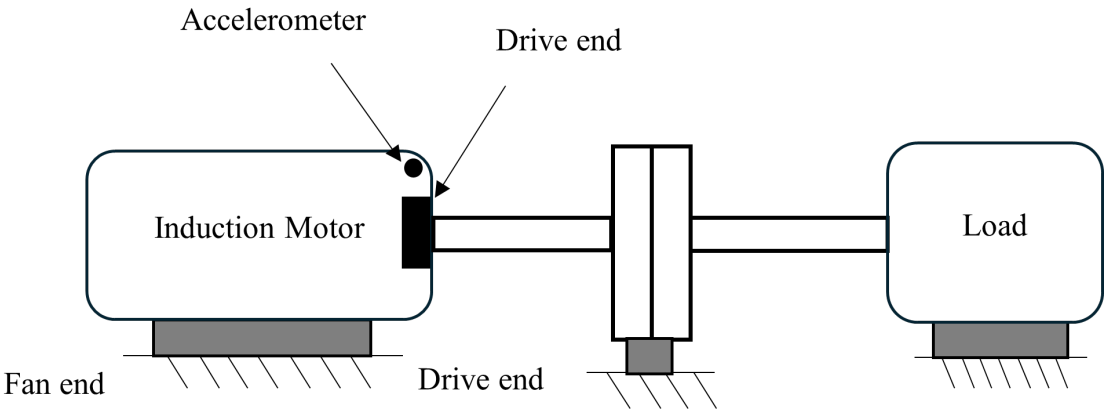


Figure 4.2 CWRU Bearing Test Stand Illustration

Table 4.1 CWRU - Bearing Size Specifications

Inside Diameter	**Outside Diameter**	**Thickness**	**Ball Diameter**	**Pitch Diameter**
0.9843	2.0472	0.5906	0.3126	1.537

4.2 The HUST Bearing Dataset

The HUST Bearing Dataset is used in this thesis to verify the performance of the BiLSTM model. The HUST Bearing Dataset consists of a large set of vibration data of different faulty bearings. The Dataset includes vibration data for 6 fault types: inner race fault, ball fault, outer race fault, and their 3 combinations. All fault sizes in this dataset have a size of about 0.0079 inches. The Dataset also includes vibration data for normal bearings. The vibration data is collected from the bearings at 3 motor loads: 0 Watts, 200 Watts, and 400 Watts. Five different bearings are used in this Dataset, their sizes (in inches) and other characteristics can be seen in Table 4.2. The test bench for the HUST Bearing Dataset can be seen in Figure 4.3 below.

Table 4.2 HUST - Bearing Size Specifications

Bearing	Inside Diameter	Outside Diameter	Ball Diameter	Ball Number
6204	0.7874	1.8504	0.2992	8
6205	0.9842	2.0472	0.3071	9
6206	1.1811	2.4409	0.3543	9
6207	1.3779	2.8347	0.4331	9
6208	1.5748	3.1496	0.4724	9

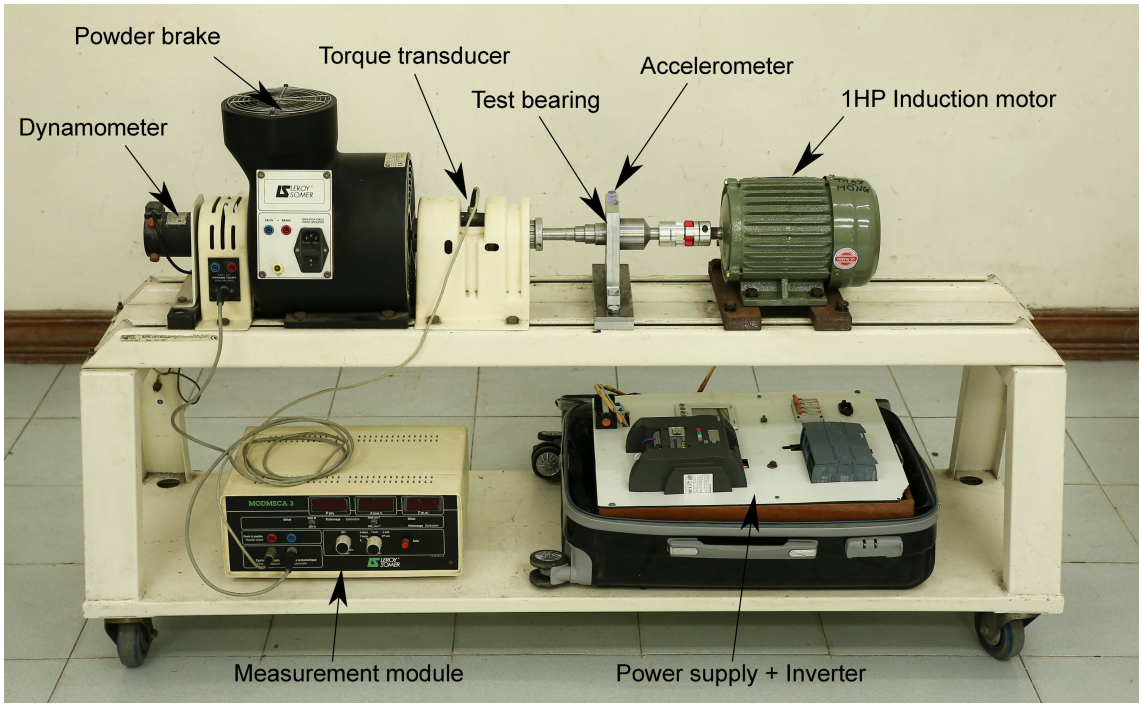


Figure 4.3 HUST Bearing Dataset Test Bench

In this thesis, the vibration data of the 4 bearings 6205, 6206, 6207, and 6208 are used. Bearing 6204's data was not used because it did not have vibration data for ball faults. The faults used are: inner race faults, ball faults, and outer race faults. The other 3 fault types were not used to keep the datasets comparable to the CWRU datasets. For the same reason, the vibration data collected at 0 Watts motor load was used.

4.3 Support Vector Machines (SVM)

Support Vector Machines (SVM) are powerful machine learning models that use supervised learning for regression, classification, and outlier detection. The fundamental concept of SVM is to find the optimal hyperplane that splits the data points of different classes in a high-dimensional space. The optimal hyperplane is defined as the one that maximizes the margin between the nearest points (support vectors) of the classes. Figure 4.4 illustrates the concept of finding the optimal hyperplane.

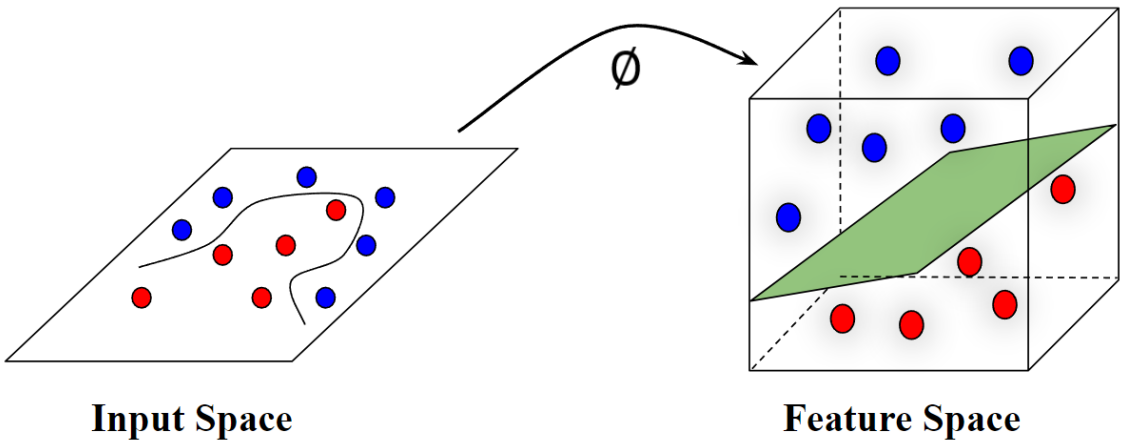


Figure 4.4 Input Plane to Optimal Plane

4.3.1 Mathematical Formulation of SVM

Given a set of training data $\{(x_i, y_i)\}_{i=1}^{n}$ where $x_i \in \mathbb{R}^n$ and $y_i \in \{-1, 1\}$, SVM aims to solve the following optimization problem:

$$min_{w,b} \frac{1}{2} \|w\|^2 \tag{4.1}$$

subject to the constraints:

$$y_i (w * x_i + b) \geq 1, \forall_i \tag{4.2}$$

Here, w represents the weight vector and b is the bias term. In scenarios where data is not perfectly separable, a slack variable ξ_i is introduced to allow some misclassification:

$$min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i}^{n} \xi_i \tag{4.3}$$

subject to:

$$y_i (w * x_i + b) \geq 1 - \xi_i, \xi \geq 0, \forall_i \tag{4.4}$$

where C (Box Constraint) is a regularization parameter that manages the trade-off between maximizing the margin and minimizing the classification error.

4.4 Optimization Algorithms for SVM

In this thesis, the optimization of the SVM models was done by finding the best values for the "Box constraint", C, and "Kernel Scale", g, to maximize classification performance. The "Kernel Scale" is a regularization parameter that influences the complexity of the model. A smaller kernel scale value results in a narrower kernel width, creating a more complex model capable of capturing finer details in the data, though it may increase the risk of overfitting. On the other hand, a larger kernel scale value produces a broader kernel, resulting in a simpler model that may generalize better but could potentially underfit the data. This thesis utilizes three optimization algorithms: Grey Wolf Optimization (GWO), Particle Swarm Optimization (PSO), and the Kepler Optimization Algorithm (KOA) to enhance SVM performance.

4.4.1 Grey Wolf Optimization (GWO)

Grey Wolf Optimization (GWO) is a nature-inspired algorithm based on grey wolves' social hierarchy and hunting behavior. The population is divided into four categories: alpha, beta, delta, and omega, which represent the leadership hierarchy. Figure 4.5 illustrates the different actors of the Grey Wolf Optimization Algorithm.

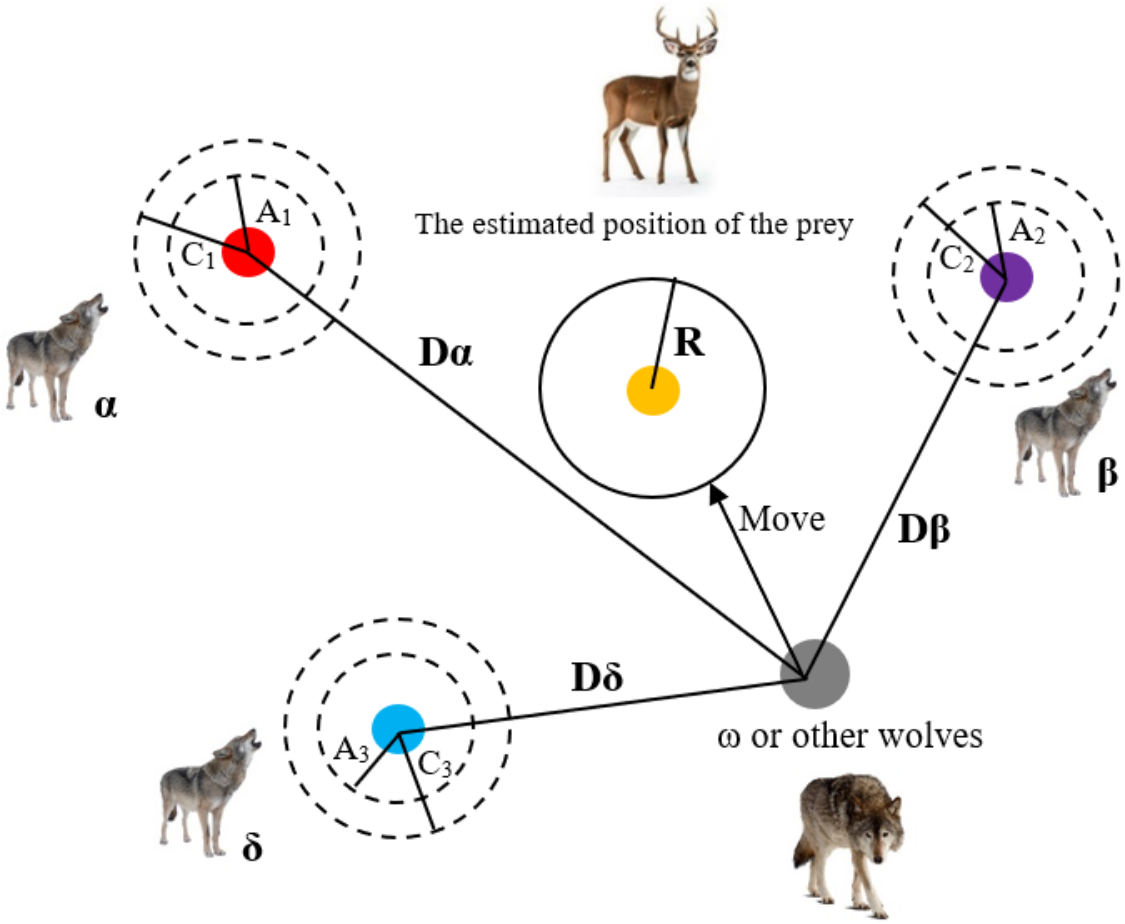


Figure 4.5 Grey Wolf Optimization

4.4.1.1 GWO Mathematical Model

The position update equations for GWO are:

$$\vec{D_\alpha} = \left|\vec{C} * \vec{X_p} - \vec{X_\alpha}\right| \tag{4.5}$$

$$\vec{X_1} = \vec{X_\alpha} - \vec{A} * \left(\vec{D_\alpha}\right) \tag{4.6}$$

$$\vec{D_\beta} = \left|\vec{C} * \vec{X_p} - \vec{X_\beta}\right| \tag{4.7}$$

$$\vec{X_2} = \vec{X_\beta} - \vec{A} * \left(\vec{D_\beta}\right) \tag{4.8}$$

$$\vec{D_\delta} = \left|\vec{C} * \vec{X_p} - \vec{X_\delta}\right| \tag{4.9}$$

$$\vec{X_3} = \vec{X_\delta} - \vec{A} * \left(\vec{D_\delta}\right) \tag{4.10}$$

where C_1, C_2, C_3 and A_1, A_2, A_3 are coefficient vectors, X_α, X_β, X_δ are the positions of the alpha, beta, and delta wolves, and X is the position of the grey wolf.

The new position of the grey wolf is then updated as follows:

$$\vec{X}\,(t+1) = \frac{\vec{X_1} + \vec{X_2} + \vec{X_3}}{3} \tag{4.11}$$

4.4.2 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) imitates the social behaviour of a flock of birds or a school of fish. Each particle adjusts its position according to its own experience and that of its neighbors. The basic working of the Particle Swarm Optimization Algorithm is shown in Figure 4.6.

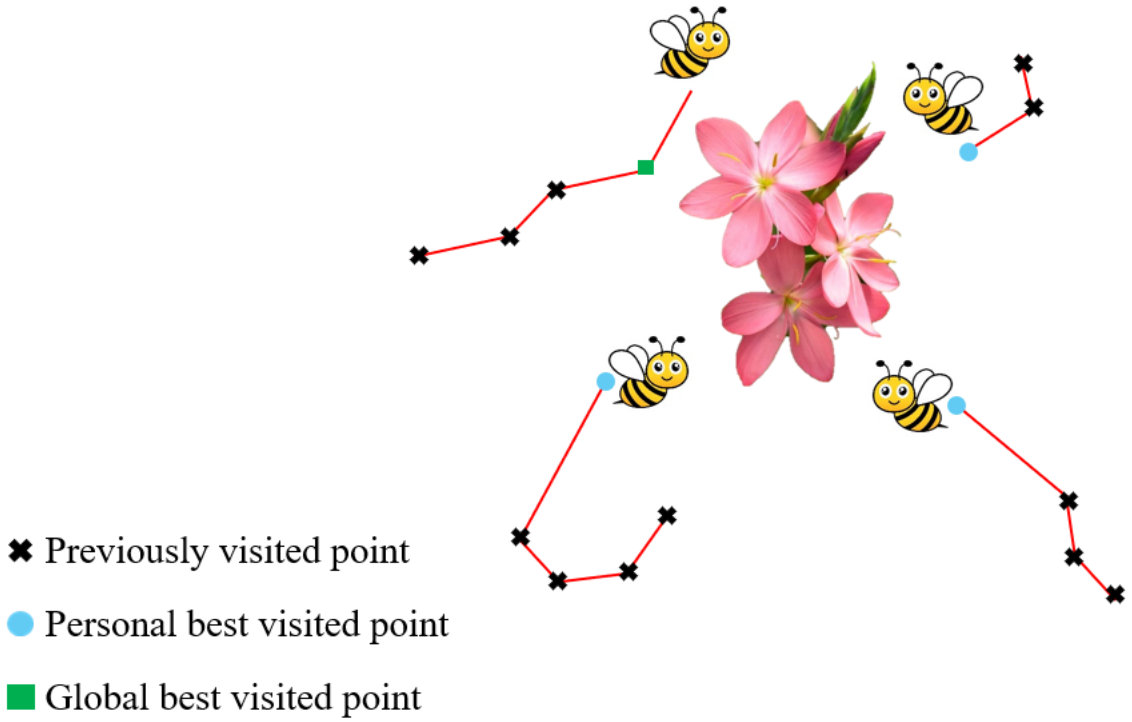


Figure 4.6 Particle Swarm Optimization

4.4.2.1 PSO Mathematical Model

The velocity and position of each particle are updated as follows:

$$v_i(t+1) = \omega v_i(t) + c_1 r_1 [p_i - x_i(t)] + c_2 r_2 [g - x_i(t)] \tag{4.12}$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{4.13}$$

where $v_i(t)$ is the velocity of particle i, $x_i(t)$ is the position of particle i, p_i is the best position found by particle i, g is the global best position found by the swarm, is the inertia weight, and c_1, c_2, r_1, and r_2 are constants and random variables that influence the cognitive and social components of the algorithm.

4.4.3 Kepler Optimization Algorithm (KOA)

The Kepler Optimization Algorithm (KOA) is inspired by the laws of planetary motion described by Johannes Kepler. It uses principles of gravitational interaction

and orbital mechanics to optimize search processes. Figure 4.7 shows a flowchart for the working of the Kepler Optimization Algorithm.

4.4.3.1 KOA Mathematical Model

The velocity update in KOA is based on gravitational forces:

$$v_i(t+1) = v_i(t) + \frac{GMm_i}{r_i^2}\Delta t \tag{4.14}$$

where G is the gravitational constant, M is the mass of the central body, m_i is the mass of the particle, r_i is the distance between the particle and the central body, and Δt is the time step.

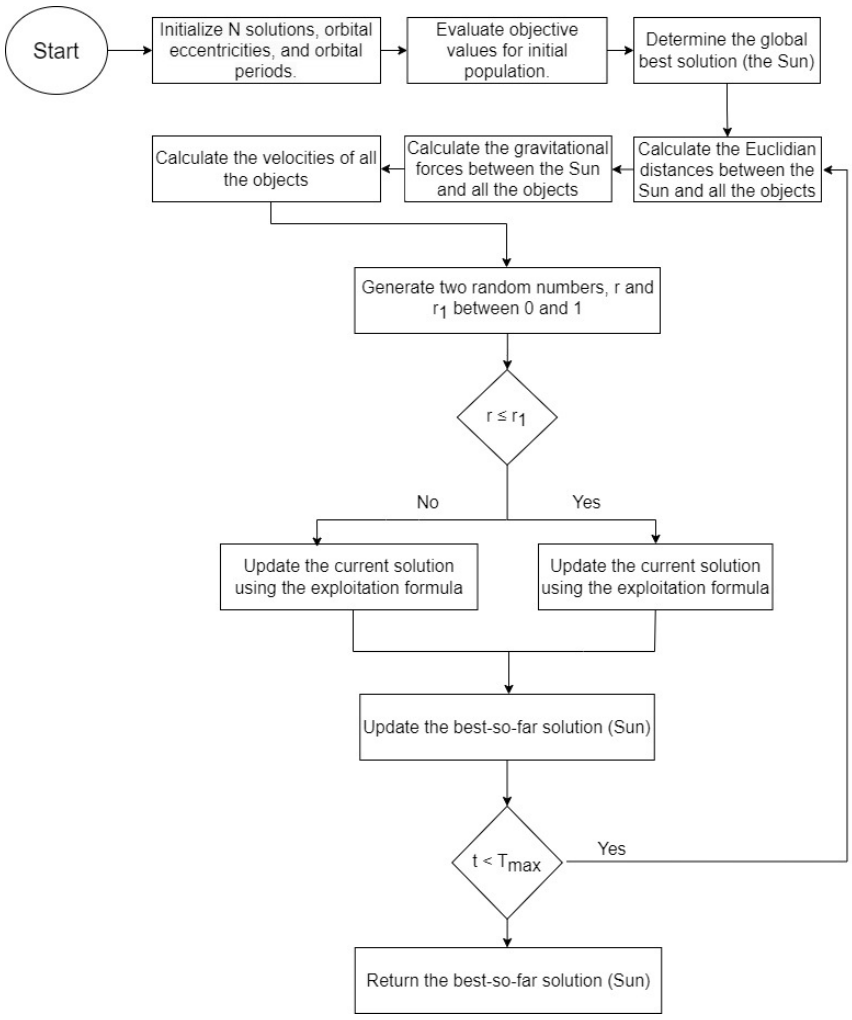


Figure 4.7 Kepler Optimization Algorithm

4.5 Deep Learning and Long Short-Term Memory Networks

Deep learning is a subset of machine learning that employs multi-layered neural networks to model complex patterns in data. It has proven effective in various applications, including speech and image recognition, time series analysis, and natural language processing.

4.5.1 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) created to capture long-term dependencies in sequential data. They address the vanishing gradient problem common in traditional RNNs by adding a memory cell that maintains its state over time. Figures 4.8 and 4.9 below show the difference in structure between traditional RNNs and the LSTM network.

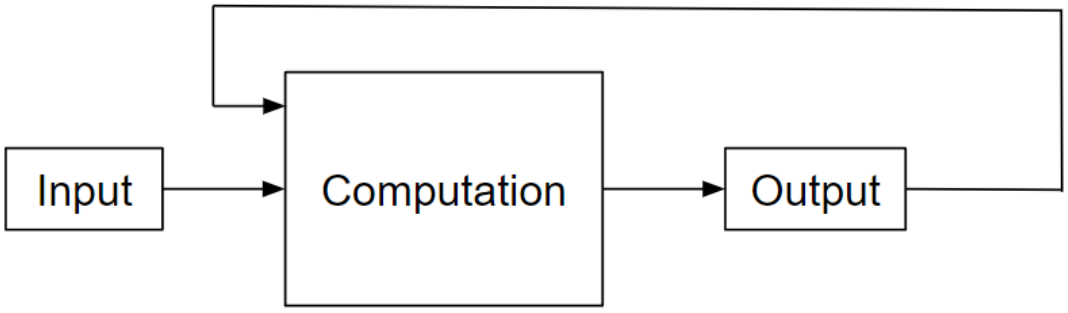


Figure 4.8 Traditional RNN

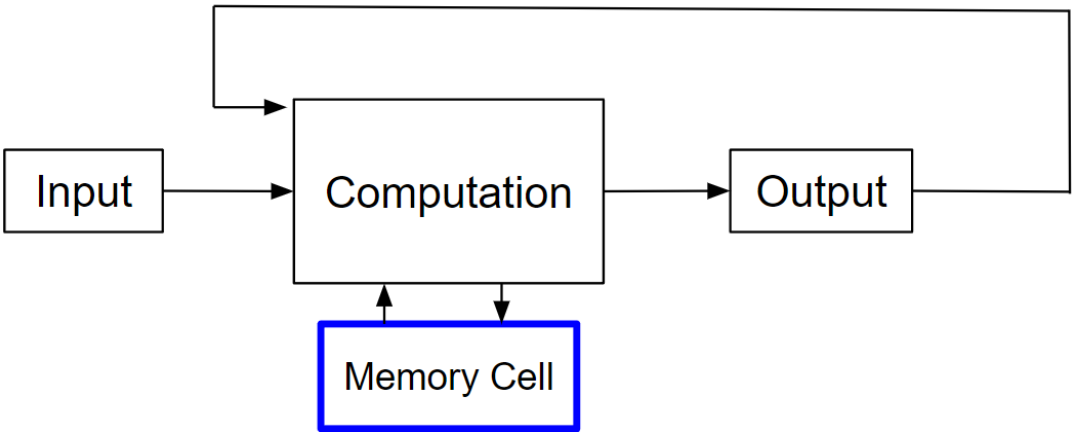


Figure 4.9 LSTM Network

4.5.1.1 LSTM Cell Equations

An LSTM cell consists of three gates: input gate, forget gate, and output gate. These gates manage the information flow into and out of the cell. An illustration of a memory cell and its gates can be seen in Figure 4.10.

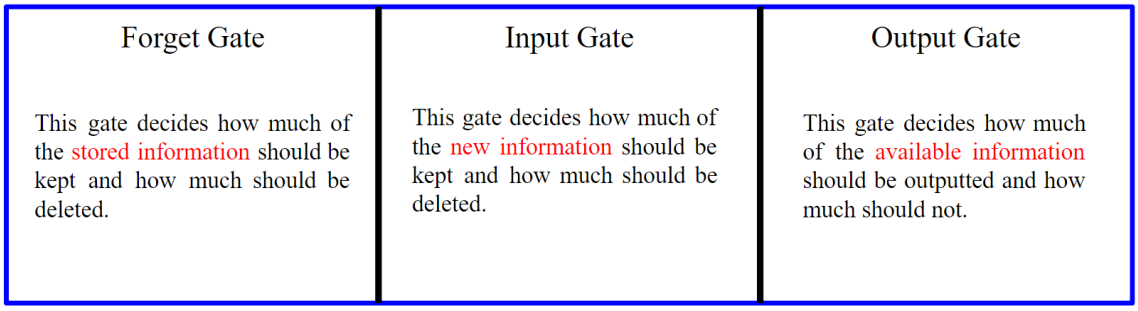


Figure 4.10 Memory Cell Structure

The LSTM Cell Equations:

$$f_t = \sigma\left(W_f * [h_{t-1}, x_t] + b_f\right) \tag{4.15}$$

$$i_t = \sigma\left(W_i * [h_{t-1}, x_t] + b_i\right) \tag{4.16}$$

$$\tilde{C}_t = tanh\left(W_C * [h_{t-1}, x_t] + b_C\right) \tag{4.17}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{4.18}$$

$$o_t = \sigma\left(W_o * [h_{t-1}, x_t] + b_o\right) \tag{4.19}$$

$$h_t = o_t * tanh\left(C_t\right) \tag{4.20}$$

where f_t is the forget gate, it is the input gate, $\tilde{C}_t$ is the candidate cell state, C_t is the cell state, o_t is the output gate, h_t is the hidden state, x_t is the input data, σ is an activation function, W_f, W_i, W_C, and W_o are linear transformation matrices, and b_f, b_f, b_C, and b_o are corresponding biases.

4.5.2 Bidirectional Long Short-Term Memory (BiLSTM)

Bidirectional LSTMs (BiLSTMs) extend the LSTM architecture by processing the input sequence in both forward and backward directions. This bidirectional approach captures information from past and future contexts, which is beneficial for tasks requiring comprehensive sequence understanding, as it significantly enhances the information accessible to the network, thereby enriching the context available to the algorithm.

4.5.2.1 BiLSTM Model

A BiLSTM network comprises two LSTM layers running in opposite directions:

Forward LSTM:

$$\overrightarrow{h} = \overrightarrow{LSTM}(h_{t-1}, x_t, C_{t-1}) \tag{4.21}$$

Backward LSTM:

$$\overleftarrow{h} = \overleftarrow{LSTM}(h_{t+1}, x_t, C_{t+1}) \tag{4.22}$$

The Final Output

$$H_t = \left[\overrightarrow{h_t}, \overleftarrow{h_t}\right] \tag{4.23}$$

Figure 4.11 below illustrates the BiLSTM model, Lin, Ji & Sun (2023).

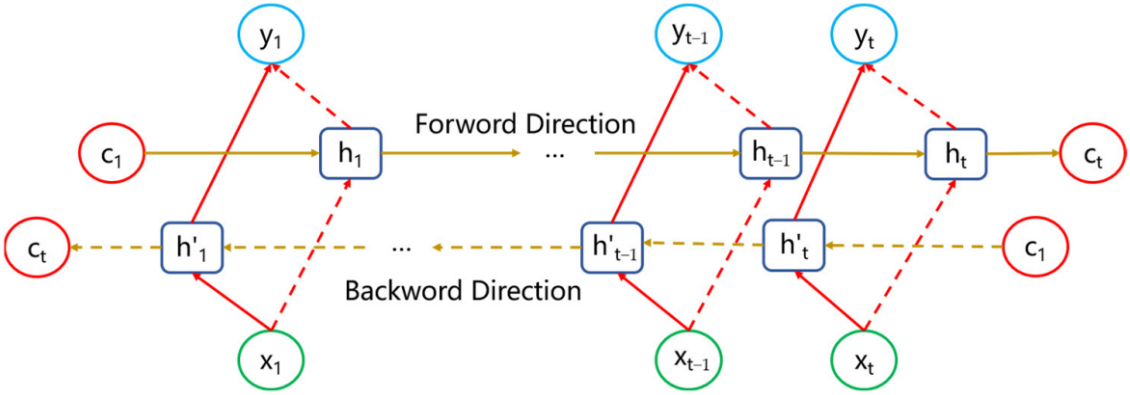


Figure 4.11 BiLSTM Network

4.6 BiLSTM Hyper-Parameters

This section provides short definitions of the hyper-parameters that were optimized for the BiLSTM model.

Array Size: The array size is the size of each data instance used in training and testing the model. It represents how each dataset is split into data instances.

Number of Epochs: It is the total number of training data iterations used in a single cycle to train the deep learning model.

Number of Hidden Layers: A hidden layer is a group of units that perform computations on the input or previous hidden layer.

Solver: The solver (also known an optimizer) is an algorithm used to update the parameters (weights and biases) of the model.

Learning Rate: The learning rate is a hyper-parameter that controls how rapidly a model picks up new information from the data while it is being trained. It regulates the step size taken in each optimization algorithm iteration.

5. RESULTS

In this chapter, we present and discuss the results of our models. In section 5.1, the results of the SVM models optimized by the three optimization algorithms, GWO, PSO, and KOA will be presented. Section 5.2 presents the results of the deep learning model using the Bidirectional Long Short-Term Memory network on the CWRU dataset. Additionally, the process of choosing the best hyper-parameters will be explained. Section 5.3 presents the results of the BiLSTM model on the HUST dataset. Finally, in section 5.4, the results of the SVM models will be compared with those of the deep learning models.

5.1 Results of the SVM Models

In this subsection, the results of the SVM models will be presented. The datasets used to train and test the SVM models had the following properties:

- Frequency: 12 kHz
- Fault size: 0.021"
- Motor load: 0 Horsepower
- Position of accelerometer: Drive End
- Classes:
  - Normal
  - Inner race fault
  - Ball fault
  - Outer race fault

Each dataset (Class) was split into 100 x 1 instances and the total instances were divided in the ratio 80:20 between train and test data. First, to obtain some reference benchmarks, the SVM model was tested on the datasets without any external optimization algorithms. The accuracy of the model was found to be 57.41%. The confusion matrix for this control experiment can be seen in Figure 5.1.

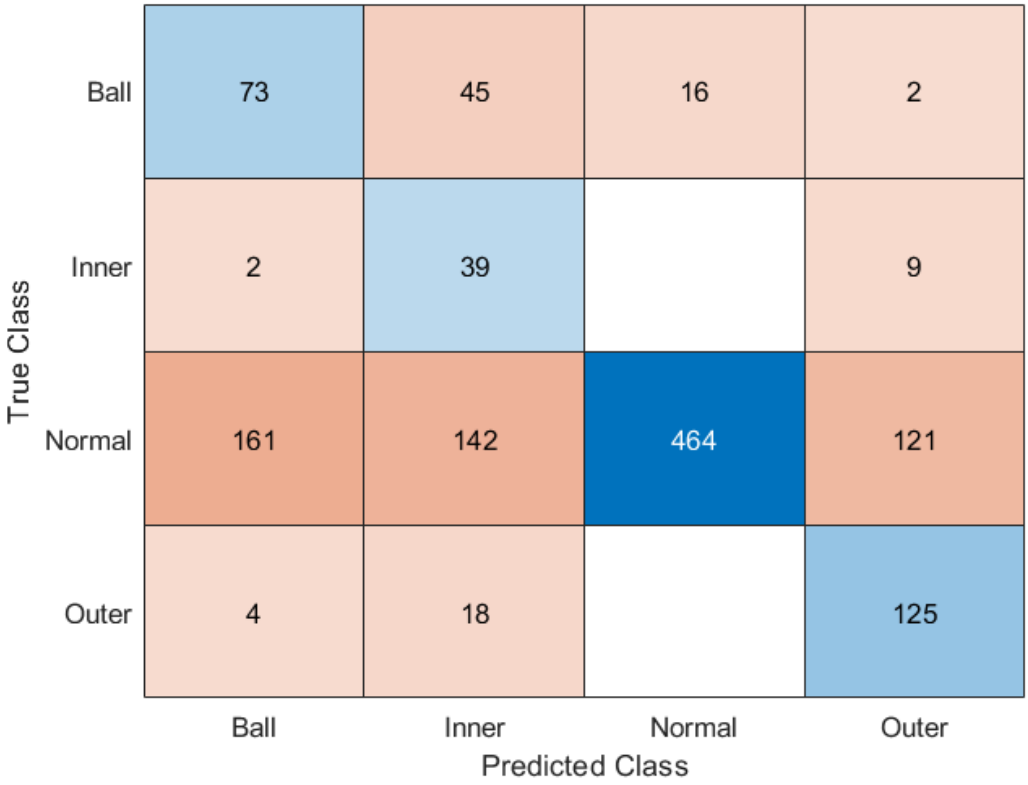


Figure 5.1 Confusion Matrix - SVM Model (Not Optimized)

Following the control experiment, three SVM models were created, each utilizing a different optimization algorithm. The first model was the GWO-SVM model. In this model, the SVM parameters "Box Constraint" and "Kernel Scale" were optimized using the Grey Wolf Optimization (GWO) algorithm. The number of iterations of the GWO algorithm was set to 300, and the number of wolves (search agents) was set to 10. The upper and lower bounds of the two parameters were set to 100 and 0.001, respectively. It took 60.425 hours to optimize the two SVM parameters using GWO. The accuracy of the model was 64.37%. The confusion matrix for the GWO-SVM model is shown in Figure 5.2 on the following page.

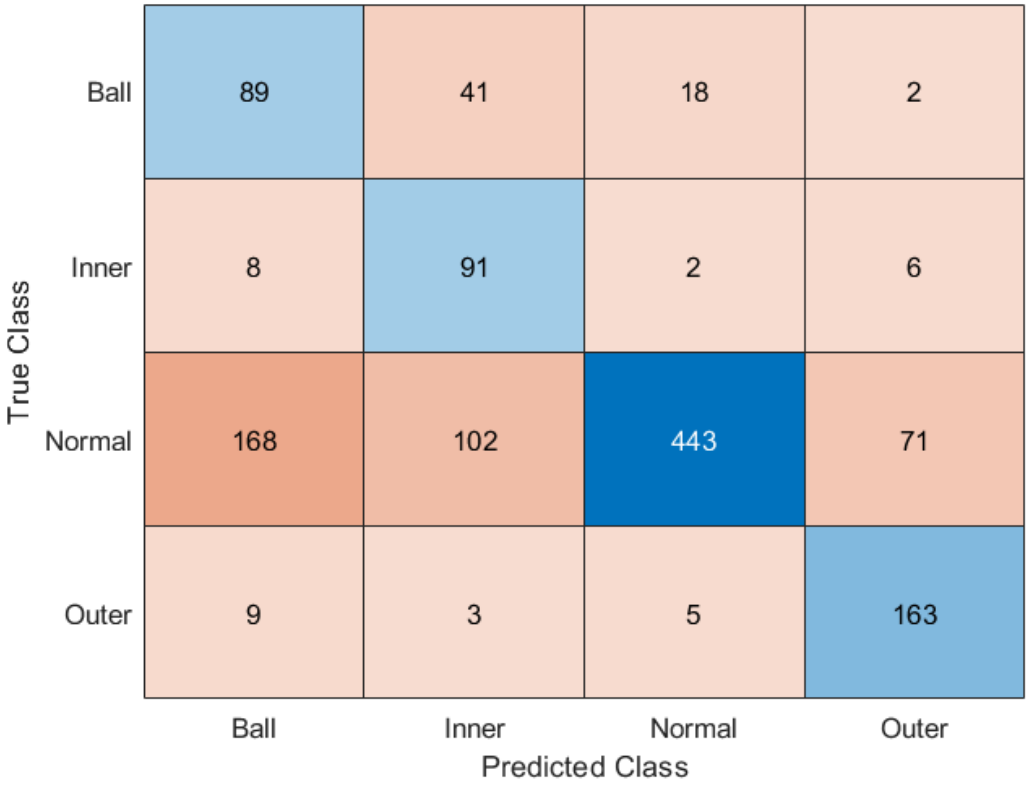


Figure 5.2 Confusion Matrix - GWO-SVM Model

The second model was the PSO-SVM model. This model optimized the same two SVM parameters using the Particle Swarm Optimization (PSO) algorithm. The number of iterations of the PSO algorithm was set to 300, and the swarm size (search agents) was set to 10. The upper and lower bounds of the two parameters were set to 100 and 0.001, respectively. It took 54.625 hours to optimize the two SVM parameters using PSO. The accuracy of the model was 62%. The confusion matrix for the PSO-SVM model is shown in Figure 5.3 on the following page.

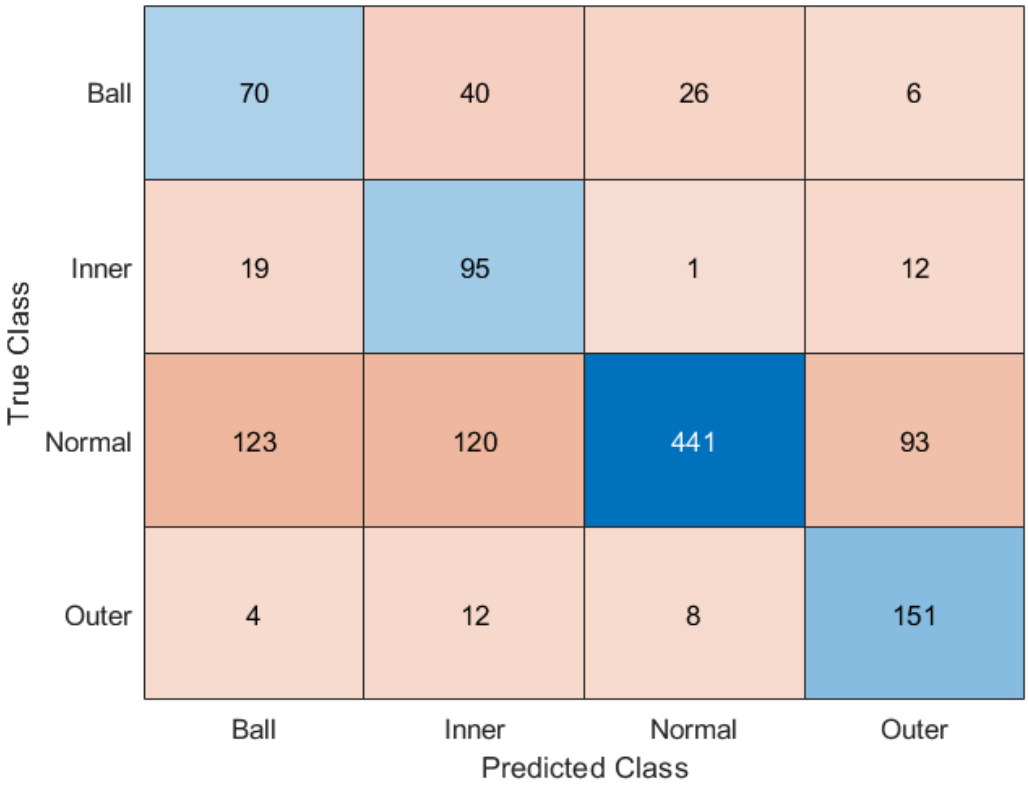


Figure 5.3 Confusion Matrix - PSO-SVM Model

The third model was the KOA-SVM model. In this model, the same two SVM parameters were optimized using the Kepler Optimization Algorithm (KOA). The number of iterations of the KOA was set to 1500, and the number of planets (search agents) was set to 30. The upper and lower bounds of the two parameters were set to 100 and 0.001, respectively. The higher numbers of iterations and search agents compared to the GWO-SVM and PSO-SVM models are because the KOA is a lot faster than GWO and PSO. It took 15.725 hours to optimize the two SVM parameters using KOA. The accuracy of the model was 62.9%. The confusion matrix for the KOA-SVM model is shown in Figure 5.4 below.

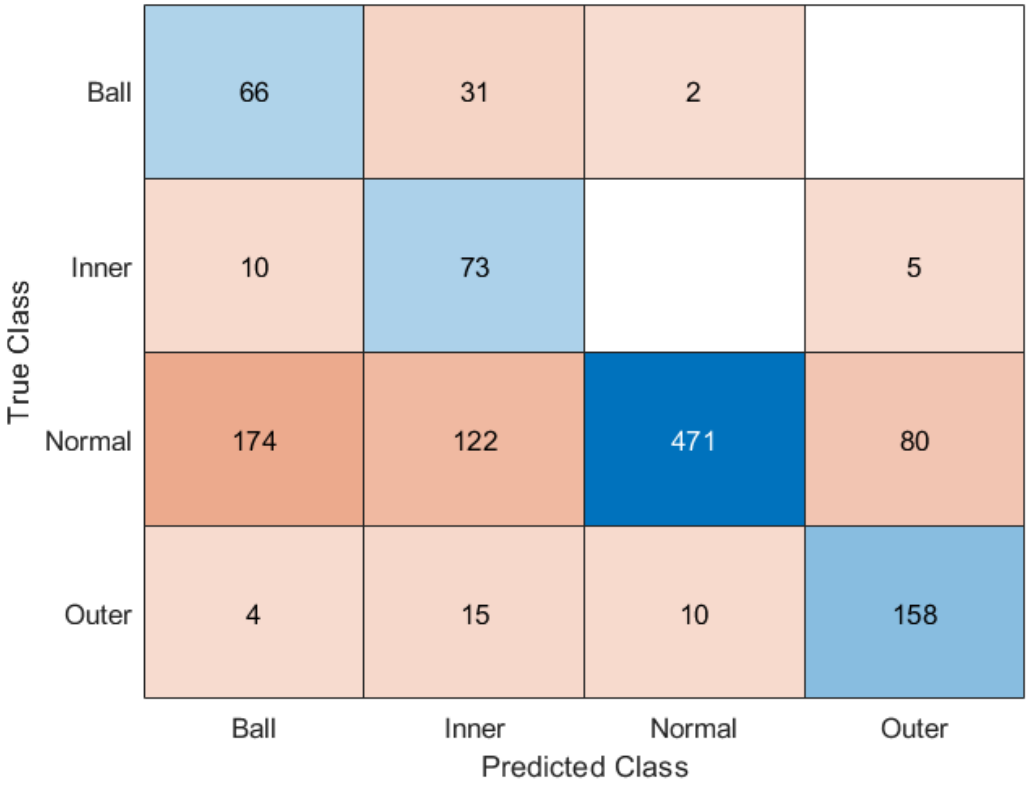


Figure 5.4 Confusion Matrix - KOA-SVM Model

The GWO-SVM model achieved the highest accuracy, and it also had the longest optimization time. The KOA-SVM achieved the second-highest accuracy despite having the shortest optimization time. Despite achieving better accuracies than the non-optimized model, None of the three SVM models could achieve high accuracies on the raw, unprocessed datasets. This is expected of SVM models because SVM models are generally suitable for small to medium sized datasets, and the CRWU datasets are too big for SVM to handle them as they are. Therefore, the datasets' sizes were reduced from 100 x 1 instances to 8 x 1 instances. This was done by calculating 8 time-domain parameters for each 100 x 1 instance and concatenating them into an 8 x 1 column vector. (Shen, Xiao, Wang, Song, 2023). The parameters, their formulas and the reasons for choosing them are shown in Table 5.1.

Table 5.1 Datasets Reduction Parameters

Parameter	**Formula**	**Reasoning**
Average value	$\bar{y} = \frac{\sum_{i=1}^{u} y_i}{u}$	Reflects the signal's general stability
Peak value	$y_{pp} = max(y_i) - min(y_i)$	Accurately depicts the signal's strength
Effective value	$y_{rms} = \sqrt{\frac{\sum_{i=1}^{u} y_i^2}{u}}$	Captures the signal's energy properties
Standard deviation	$\sigma_y = \sqrt{\frac{\sum_{i=1}^{u}(y_i - \bar{y}^2)}{u}}$	Reflects the dynamic portion of the signal's energy
Margin coefficient	$C_e = \frac{max(\lvert y_i \rvert)}{\left(\frac{\sum_{i=1}^{u}\sqrt{\lvert y_i \rvert}}{u}\right)^2}$	Reflects how worn out mechanical equipment is
Pulse Factor	$C_f = \frac{max(\lvert y_i \rvert)}{\lvert \bar{y} \rvert}$	Reflects the detecting signal's impulse energy response
Peak factor	$I_p = \frac{max(\lvert y_i \rvert)}{y_{rms}}$	Can reflect the signal's impact energy
Kurtosis coefficient	$c_q = \frac{\frac{\sum_{i=1}^{u}(y_i - \bar{y})^4}{u}}{\left(\frac{\sum_{i=1}^{u}(y_i - \bar{y})^2}{u}\right)^2}$	Can reflect the signal's impact energy

The three models, and the control experiment without any optimization algorithms, were run again using the reduced datasets. The same numbers of iterations and search agents, and parameters' upper and lower bounds were used for the GWO-SVM and PSO-SVM models. For the KOA-SVM model, the number of iterations was increased to 5000 to bring the optimization time closer to the other two models. The number of search agents and the parameters' upper and lower bounds of the model were kept the same. Figures 5.5, 5.6, 5.7, and 5.8 below show the confusion matrices of the three SVM models, and the default model with no optimization, when the reduced datasets were used.

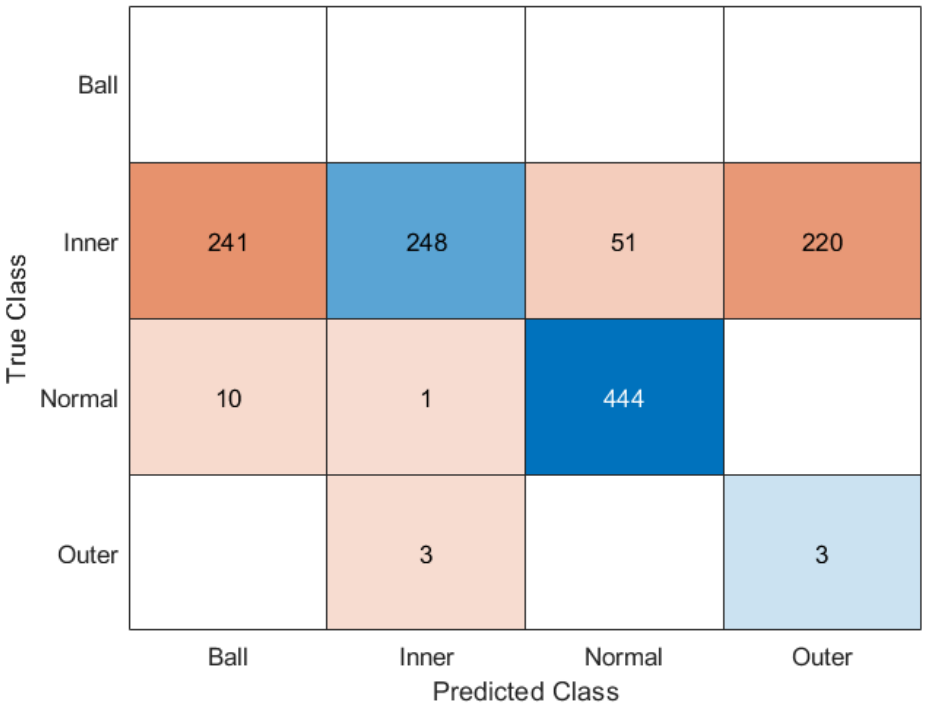


Figure 5.5 Confusion Matrix - SVM Model with Reduced Datasets (Not Optimized)

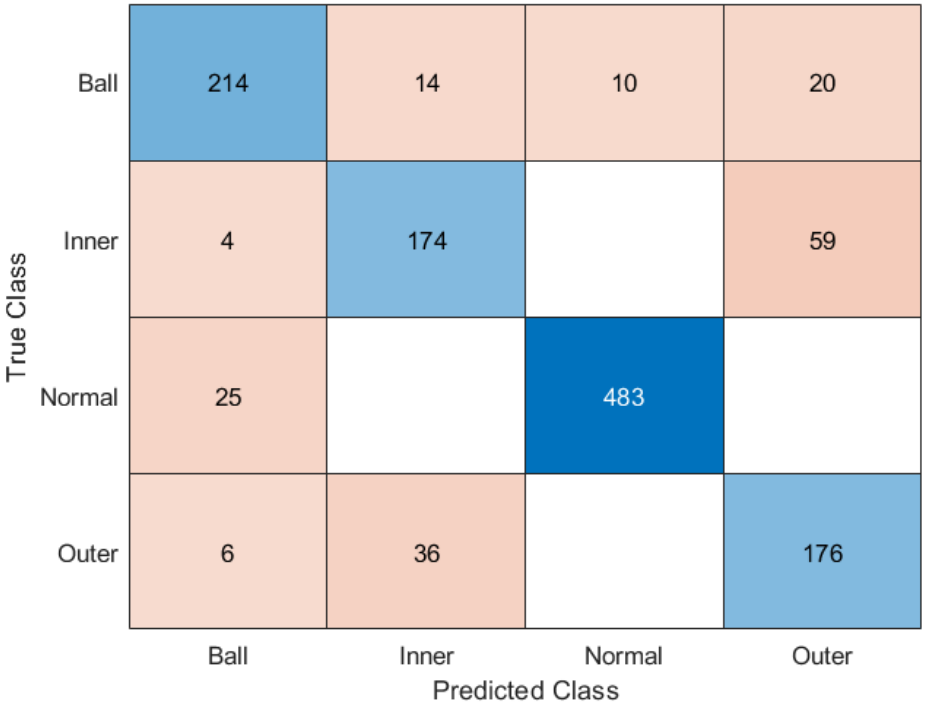


Figure 5.6 Confusion Matrix - GWO-SVM Model with Reduced Datasets

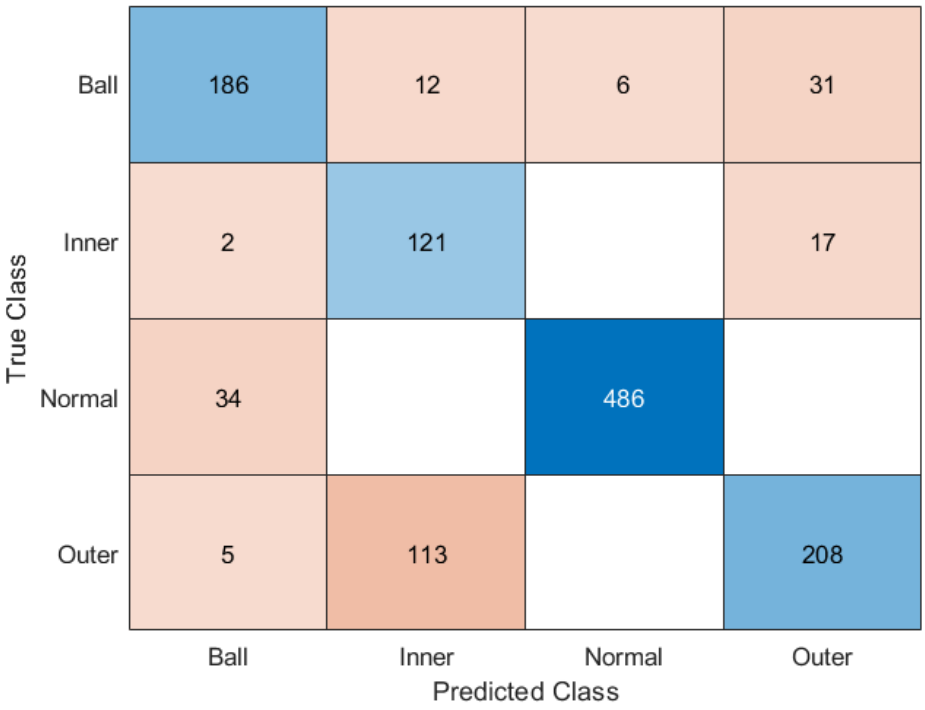


Figure 5.7 Confusion Matrix - PSO-SVM Model with Reduced Datasets

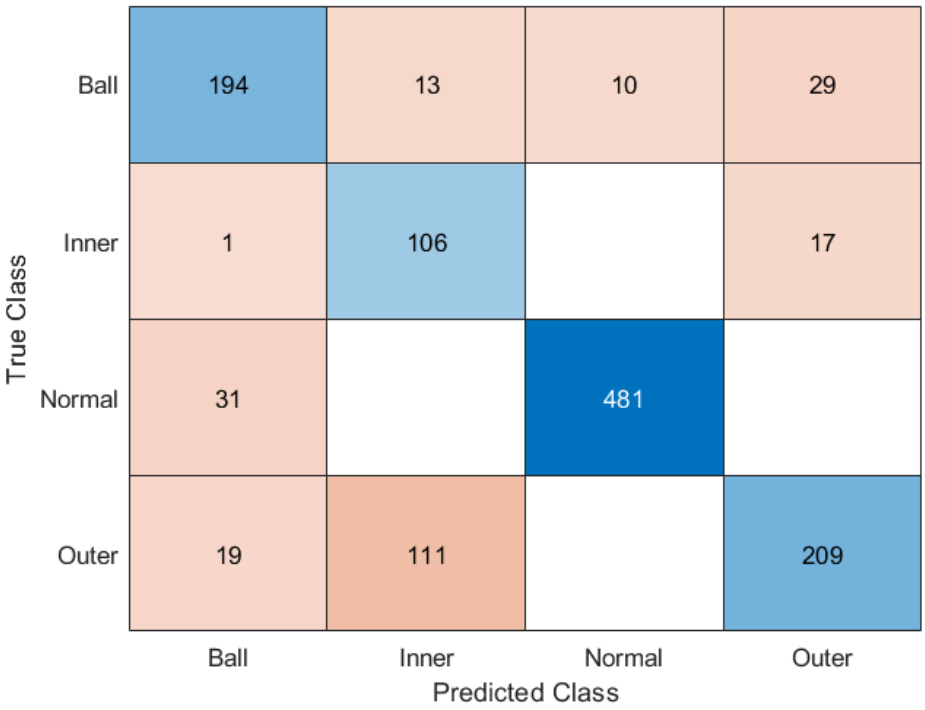


Figure 5.8 Confusion Matrix - KOA-SVM Model with Reduced Datasets

As expected, using the reduced datasets enabled the SVM models to achieve higher accuracies. The accuracies of the GWO-SVM, PSO-SVM, and the KOA-SVM were 84.75%, 81.98%, 81.08%, respectively. The three models achieved significantly higher accuracies than the non-optimized model, which achieved an accuracy of only 56.92%. Despite the increase in accuracies, none of the models could achieve very high accuracies, especially when compared with the deep learning models whose results will be presented in the next section.

5.2 Results of the BiLSTM models - CWRU Dataset

In this subsection, we present the results of the BiLSTM models and the process of finding the best hyper-parameters for the BiLSTM model using the CWRU dataset. The datasets used to train and test the SVM models had the following properties:

- Frequency: 12 kHz
- Fault size: 0.021"
- Motor load: 0 Horsepower
- Position of accelerometer: Drive End
- Classes:
  - Normal
  - Inner race fault
  - Ball fault
  - Outer race fault

The first step was to decide how to split each CWRU dataset into data instances. As previously mentioned, each dataset consists of a single column and thousands of rows of vibration data from the accelerometer. We started by splitting the data into 50 x 1 arrays and treating each array as a data instance, either a training data instance or a test one. The instances were split in the ratio 80:20 between train and test data. After that, we iteratively increased the size of the data instances by 50 rows. The size that resulted in the highest accuracy was chosen as the optimal data instance size. The optimal size was found to be 100 x 1. The rest of the

hyper-parameters were kept, for now, at their default values in the MATLAB code. Table 5.2 displays the effect of changing the data instances' size on the accuracy of the model. Figure 5.9 shows the confusion matrices of the model for different data sizes.

Table 5.2 Hyper-parameters Tuning - Array Size

Size	**Epochs**	**Hidden Layers**	**Solver**	**Learning rate**	**Accuracy (%)**	**Time (mins)**
50	200	120	Adam	0.002	98.36	46.98
100	200	120	Adam	0.002	99.51	42.57
150	200	120	Adam	0.002	99.51	56.72
200	200	120	Adam	0.002	97.54	92.08

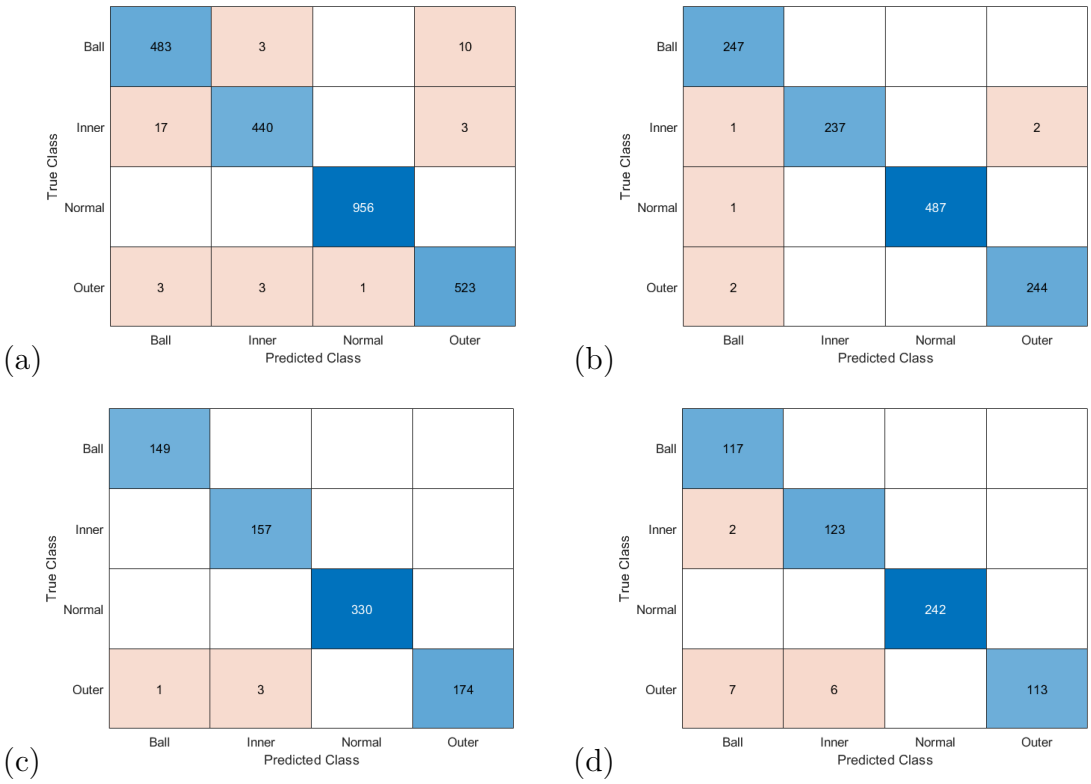


Figure 5.9 Confusion Matrices - Array Size: (a) 50 (b) 100 (c) 150 (d) 200

Next, we tried to find the optimal number of epochs. In general, increasing the number of epochs should increase the accuracy of the model because it is getting trained on the data more. However, overfitting becomes a risk if the model is trained on the training data too much as the model becomes too complex and fits the training data too closely. On the other hand, if the model is not trained enough, the model could end up being too simple and consequently underfit the data, not capturing relationships and patterns accurately. Therefore, it is very important to find the optimal number of epochs to avoid both overfitting and underfitting. The starting number of epochs was chosen as 20 and was iteratively increased by 20. The optimal number of epochs was found to be 280. Table 5.3 shows the effect of increasing the number of epochs on the accuracy of the model. Figure 5.10 shows some of the confusion matrices of the model for different epochs.

Table 5.3 Hyper-parameters Tuning - Number of Epochs

Size	**Epochs**	**Hidden Layers**	**Solver**	**Learning rate**	**Accuracy (%)**	**Time (mins)**
100	20	120	Adam	0.002	93.28	8.63
100	40	120	Adam	0.002	97.54	16.17
100	60	120	Adam	0.002	98.44	12.65
100	80	120	Adam	0.002	97.22	17.88
100	100	120	Adam	0.002	99.02	22.02
100	120	120	Adam	0.002	98.61	25.58
100	140	120	Adam	0.002	99.02	29.65
100	160	120	Adam	0.002	98.28	32.27
100	180	120	Adam	0.002	99.10	83.37
100	200	120	Adam	0.002	99.51	42.57
100	220	120	Adam	0.002	97.55	46.92
100	240	120	Adam	0.002	99.51	42.58
100	260	120	Adam	0.002	99.59	74.80
100	280	120	Adam	0.002	99.59	61.25
100	300	120	Adam	0.002	99.10	85.00
100	320	120	Adam	0.002	99.10	67.55
100	340	120	Adam	0.002	99.34	78.37
100	360	120	Adam	0.002	98.28	81.92

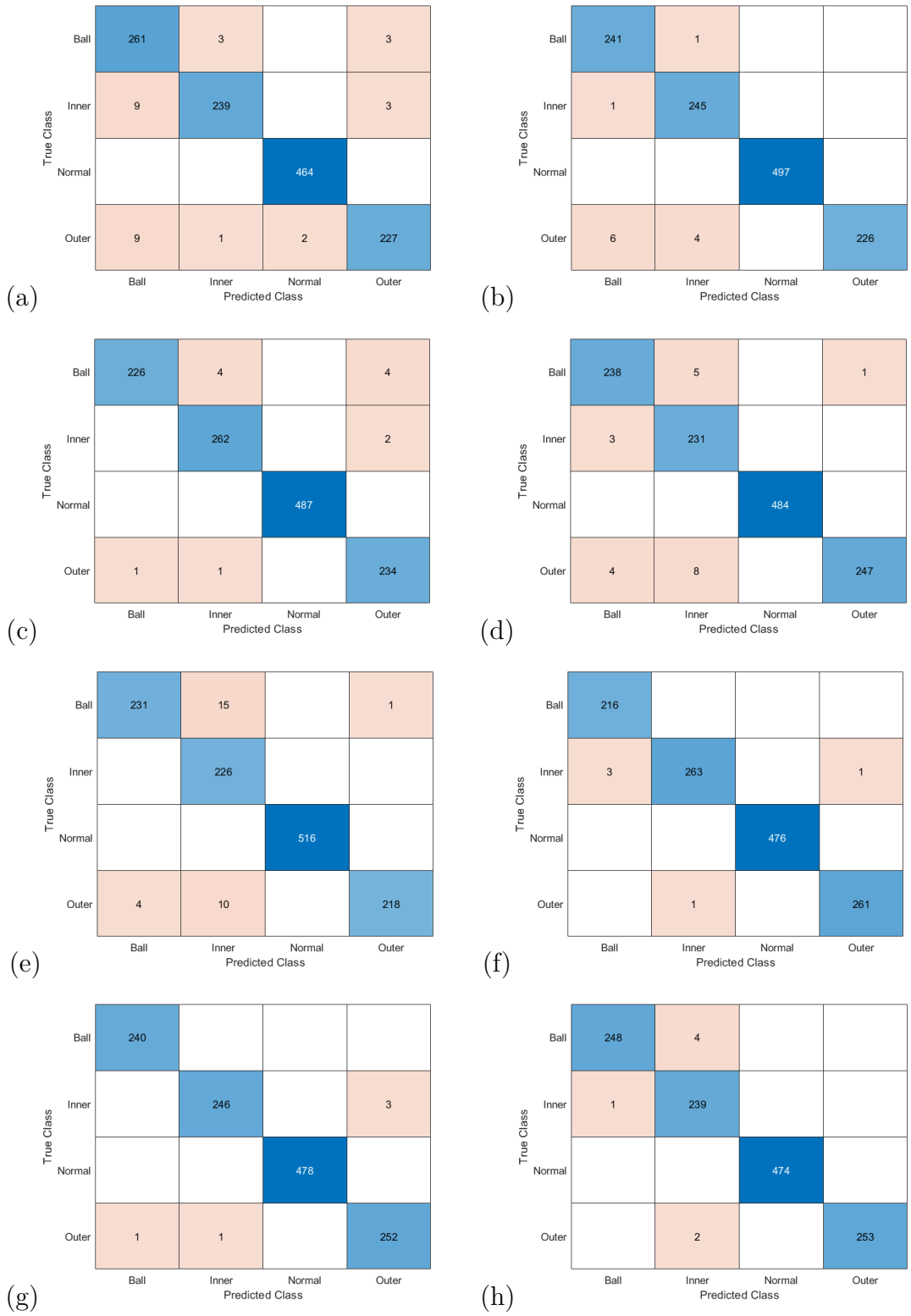


Figure 5.10 Confusion Matrices - Epochs: (a) 40 (b) 100 (c) 140 (d) 160 (e) 220 (f) 260 (g) 280 (h) 340

The next hyper-parameter to optimize was the number of hidden layers. Like epochs, increasing the number of hidden layers increases the complexity of the model, which can lead to over-fitting. Having too little hidden layers can result in the model under-fitting. Moreover, having too many hidden layers increases the model's training time. Hence, optimizing the number of hidden layers is crucial to achieving the highest possible performance from the model. The starting number of hidden layers was set as 10 and was increased in increments of 10. The optimal number of hidden layers was found to be 190. Table 5.4 presents the model's performance at different number of hidden layers. Figure 5.11 shows some of the confusion matrices of the model for different numbers of hidden layers.

Table 5.4 Hyper-parameters Tuning - Hidden Layers

Size	**Epochs**	**Hidden Layers**	**Solver**	**Learning rate**	**Accuracy (%)**	**Time (mins)**
100	280	10	Adam	0.002	96.97	17.42
100	280	20	Adam	0.002	98.12	27.82
100	280	30	Adam	0.002	98.36	37.52
100	280	40	Adam	0.002	98.77	45.63
100	280	50	Adam	0.002	99.51	28.22
100	280	60	Adam	0.002	99.10	31.17
100	280	70	Adam	0.002	99.02	36.18
100	280	80	Adam	0.002	99.10	41.92
100	280	90	Adam	0.002	99.02	46.28
100	280	100	Adam	0.002	99.18	50.70
100	280	110	Adam	0.002	98.28	55.70
100	280	120	Adam	0.002	99.59	61.25
100	280	130	Adam	0.002	99.59	81.65
100	280	140	Adam	0.002	99.34	83.40
100	280	150	Adam	0.002	99.43	74.20
100	280	160	Adam	0.002	98.85	118.18
100	280	170	Adam	0.002	99.43	87.22
100	280	180	Adam	0.002	99.59	95.60
100	280	190	Adam	0.002	99.92	101.33
100	280	200	Adam	0.002	99.51	111.35
100	280	210	Adam	0.002	99.84	116.60
100	280	220	Adam	0.002	99.34	181.65

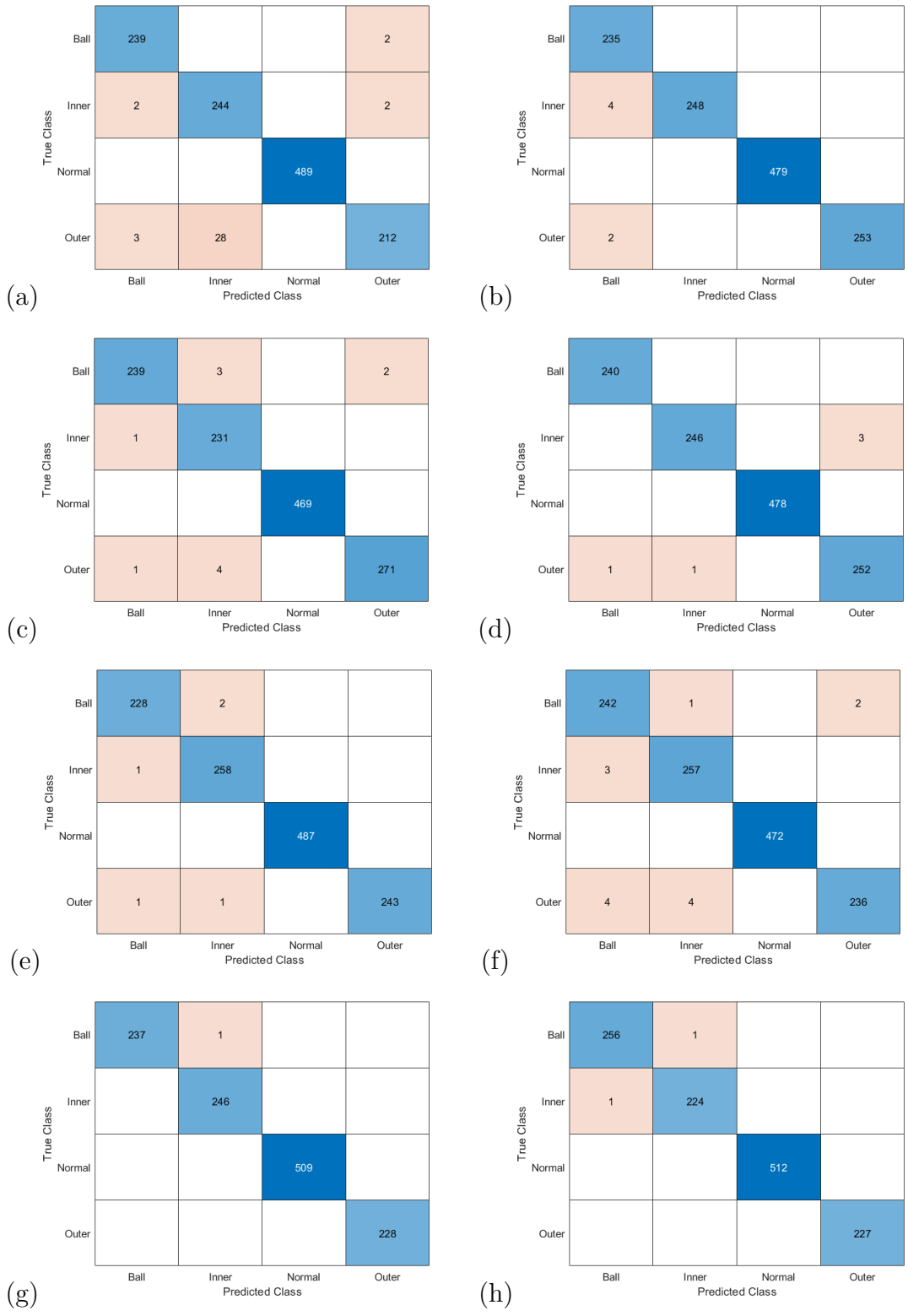


Figure 5.11 Confusion Matrices - Hidden Layers: (a) 10 (b) 50 (c) 80 (d) 120 (e) 130 (f) 160 (g) 190 (h) 210

The optimal solver was found after the number of hidden layers. The MATLAB deep learning toolbox offers the choice of three solvers: Adam, RMSprop and SGDM. While Adam and RMSprop were able to effectively optimize the weights and biases of the BiLSTM model, SGDM could not replicate their success. The optimal solver was found to be Adam. Table 5.5 presents the model's accuracies when each solver is used. Figure 5.12 shows the confusion matrices of the model for different solvers. One possible reason why the SGDM solver performed poorly compared to Adam and RMSprop, despite being somewhat similar in their underlying computations, is because SGDM is typically sensitive to the choice of learning rate. Adam and RMSprop adjust the learning rate for each parameter independently, reducing its dependency on the initial learning rate. On the other hand, SGDM utilizes a fixed learning rate, slightly modified by momentum, which can be challenging if the gradients differ greatly in magnitude across the dimensions of the dataset.

Table 5.5 Hyper-parameters Tuning - Solver

Size	**Epochs**	**Hidden Layers**	**Solver**	**Learning rate**	**Accuracy (%)**	**Time (mins)**
100	280	190	Adam	0.002	99.92	101.33
100	280	190	RMSprop	0.002	99.34	107.43
100	280	190	SGDM	0.002	69.21	119.68

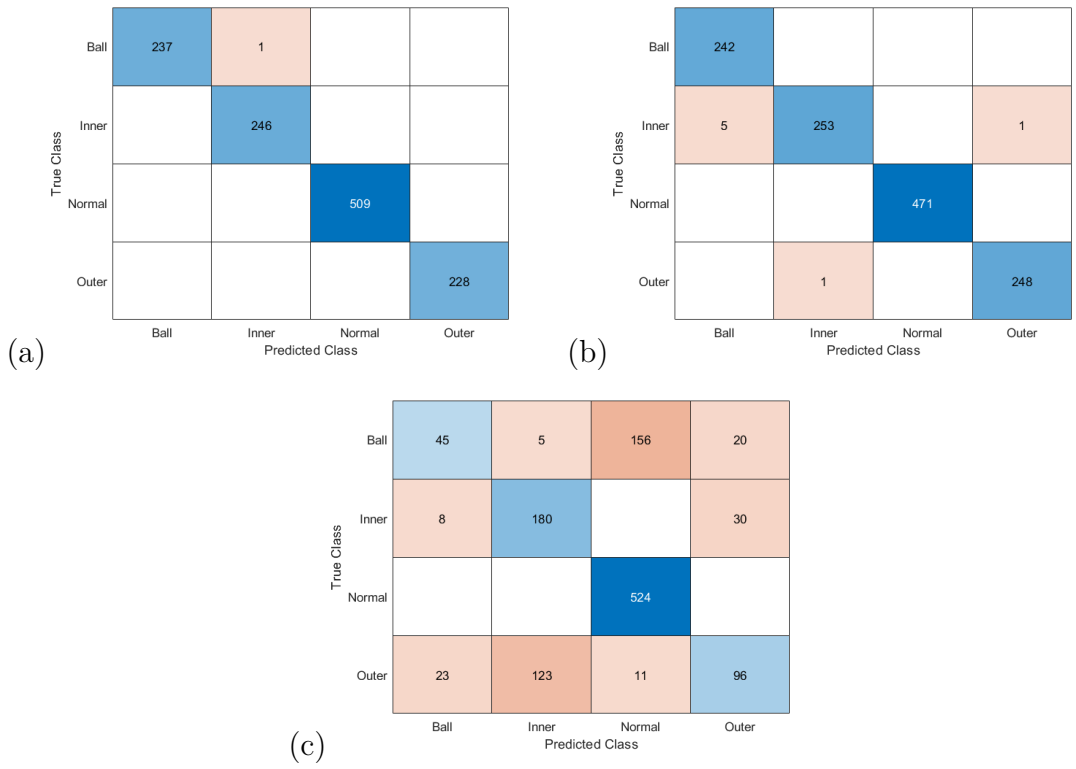


Figure 5.12 Confusion Matrices - Solver: (a) Adam (b) RMSprop (c) SGDM

After choosing the optimal solver, the learning rate was optimized. Picking a suitable learning rate is critical to the model's learning. A small learning rate can lead to a slow learning process, while a large one could prevent the convergence of the loss value, resulting in the failure of the learning process. The initial learning rate was chosen to be 0.001 and was iteratively increased by 0.001. The optimal learning rate was found to be 0.002. Even though the model achieved the same accuracy when the learning rate was set to 0.009, it took longer to train the model. The learning rate that led to a shorter training time was chosen. Table 5.6 shows the effect of increasing the learning rate on the accuracy of the BiLSTM model. Figure 5.13 shows some of the confusion matrices of the model for different learning rates.

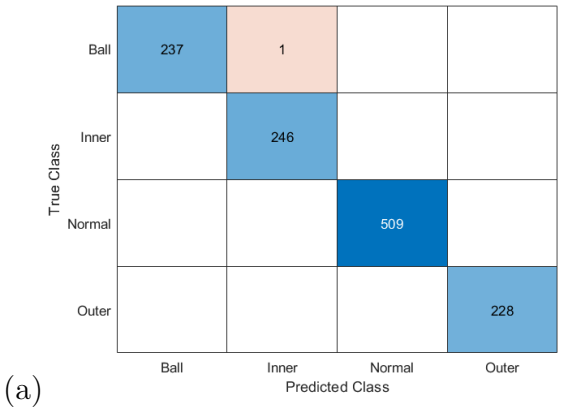
Table 5.6 Hyper-parameters Tuning - Learning Rate

Size	**Epochs**	**Hidden Layers**	**Solver**	**Learning rate**	**Accuracy (%)**	**Time (mins)**
100	280	190	Adam	0.001	98.77	97.33
100	280	190	Adam	0.002	99.92	101.33
100	280	190	Adam	0.002	99.67	102.88
100	280	190	Adam	0.004	99.34	104.25
100	280	190	Adam	0.005	99.51	121.33
100	280	190	Adam	0.006	99.75	114.83
100	280	190	Adam	0.007	99.59	103.87
100	280	190	Adam	0.008	98.12	115.58
100	280	190	Adam	0.009	99.92	134.77
100	280	190	Adam	0.01	99.51	122.13

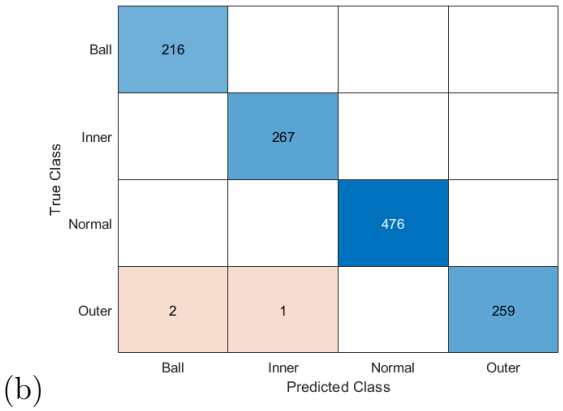
The best hyper-parameters found by this thesis for the BiLSTM model are shown in Table 5.7 below.

Table 5.7 BiLSTM Model Optimal Hyper-parameters

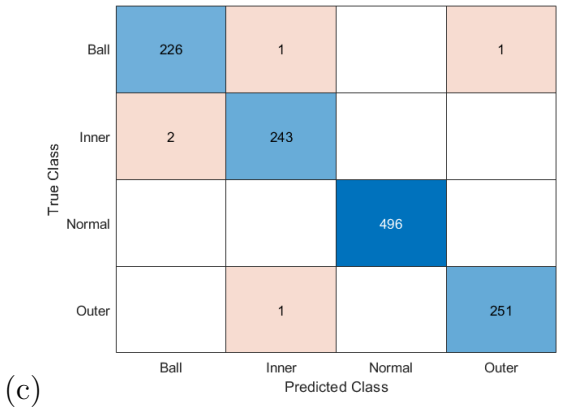
Size	**Epochs**	**Hidden Layers**	**Solver**	**Learning rate**
50	200	120	Adam	0.002



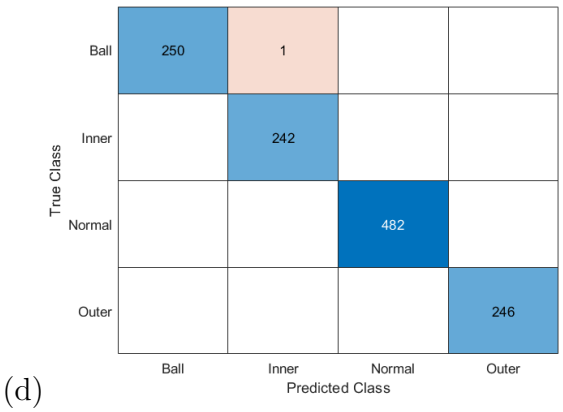
(a)



(b)



(c)



(d)

Figure 5.13 Confusion Matrices - Learning Rate: (a) 0.002 (b) 0.006 (c) 0.007 (d) 0.009

After achieving a high accuracy of 99.92% on the datasets of the fault size 0.021", the model was tested on other fault sizes, 0.007", 0.014", and 0.028", to test the model's competence in detecting and classifying bearing faults of different sizes. The model achieved accuracies of 100%, 99.84% and 99.9% on the datasets of fault sizes 0.007", 0.014", and 0.028" respectively, proving the model's high performance is not limited to one fault size. Figure 5.14 shows the confusion matrices of the model for different fault sizes, including fault size 0.021" for reference.

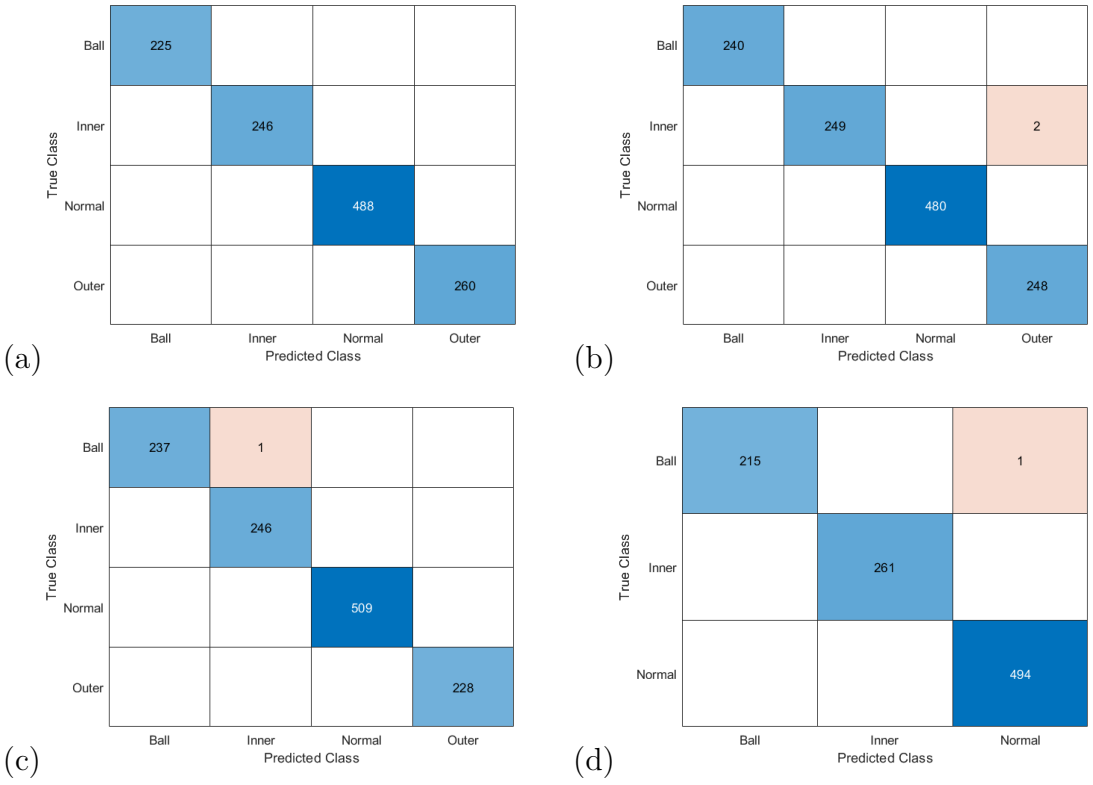


Figure 5.14 Confusion Matrices - Fault Size: (a) 0.007" (b) 0.014" (c) 0.021" (d) 0.028"

The next challenge for the model was to detect and classify the bearing faults when all the fault sizes were combined. We tried to train the model on 12 different classes: 1) Normal, 2) 0.007" inner race fault, 3) 0.014" inner race fault, 4) 0.021" inner race fault, 5) 0.028" inner race fault, 6) 0.007" outer race fault, 7) 0.014" outer race fault, 8) 0.021" outer race fault, 9) 0.007" ball fault, 10) 0.014" ball fault, 11) 0.021" ball fault, 12) 0.028" ball fault. Figure 5.15 shows the confusion matrix of the model for this training trial.

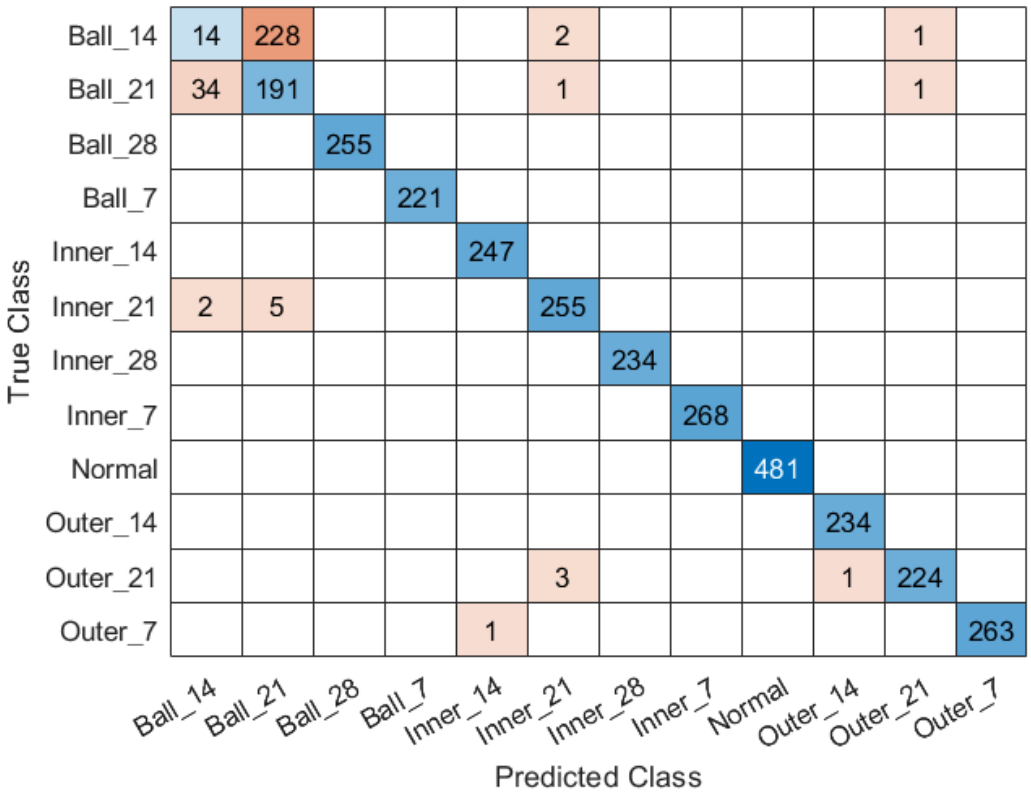


Figure 5.15 Confusion Matrix - Combination of Fault Sizes (12 Classes)

The model had 279 misclassifications out of 3166 predictions, achieving an accuracy of 91.19%. While it might seem that the model was unable to maintain the same performance in detecting and classifying bearing faults when the fault sizes were combined, this was not the case. By taking a closer look at the confusion matrix, only 16 of the 279 misclassifications were wrong fault type classifications while the remaining 263 were wrong fault size classifications. If we do not consider the wrong fault size misclassifications as misclassifications, the model's accuracy in detecting and classifying the type of the bearing fault was 99.49%.

The reason why bearing fault detection is vital is the need to detect bearing faults and to know the types of said faults. The size of the fault is not of much

importance; it should not matter whether the fault size is 0.007" or 0.028", the bearing will have to be replaced either way to maintain the correct functionality of the machine and to avoid drops in the machine's efficiency and the quality of production. For this reason, it is reasonable to only care about the model's performance in classifying the types of faults and not their sizes as well.

To verify the model's ability of classifying the types of faults when the different fault sizes are combined, the previous experiment was repeated but with only 4 classes instead of 12: 1) Normal, 2) inner race fault, 3) outer race fault, 4) ball fault. The 4 inner race fault classes in the previous experiment were all given the same label, "inner race fault". Likewise, the 3 outer race fault classes were all given the label "outer race fault", and the 4 ball fault classes were all given the label "ball race fault". The model was able to achieve an accuracy of 99.62%, with only 12 misclassifications out of 3166 predictions. This shows that the model is highly capable of detecting and classifying the types of bearing faults of any size. Figure 5.16 shows the confusion matrix of the model for this experiment.

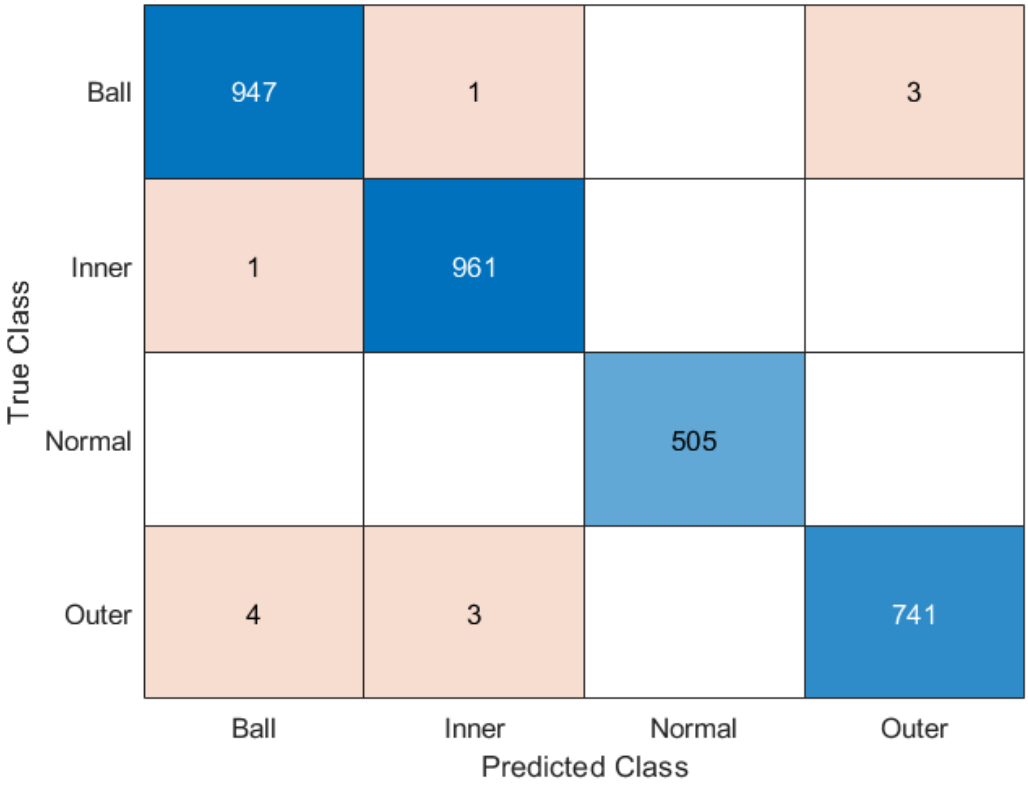


Figure 5.16 Confusion Matrix - Combination of Fault Sizes (4 Classes)

5.3 Results of the optimized BiLSTM model - HUST Dataset

After choosing the set of BiLSTM hyper-parameters that achieved the best results, the model was tested on the HUST Bearing Dataset. The BiLSTM was tested on the vibration data of 4 bearings from the dataset. Table 5.8 shows the model's performance on the HUST Bearing Dataset. The confusion matrices of the BiLSTM model can be seen in Figure 5.17 below. As is evident, the BiLSTM model achieved very high accuracies on the HUST Dataset, further proving its high capability of detecting and classifying bearing faults.

Table 5.8 BiLSTM Model Results - HUST Dataset

Bearing	**Accuracy (%)**
6205	99.27
6206	98.73
6207	99.58
6208	97.09

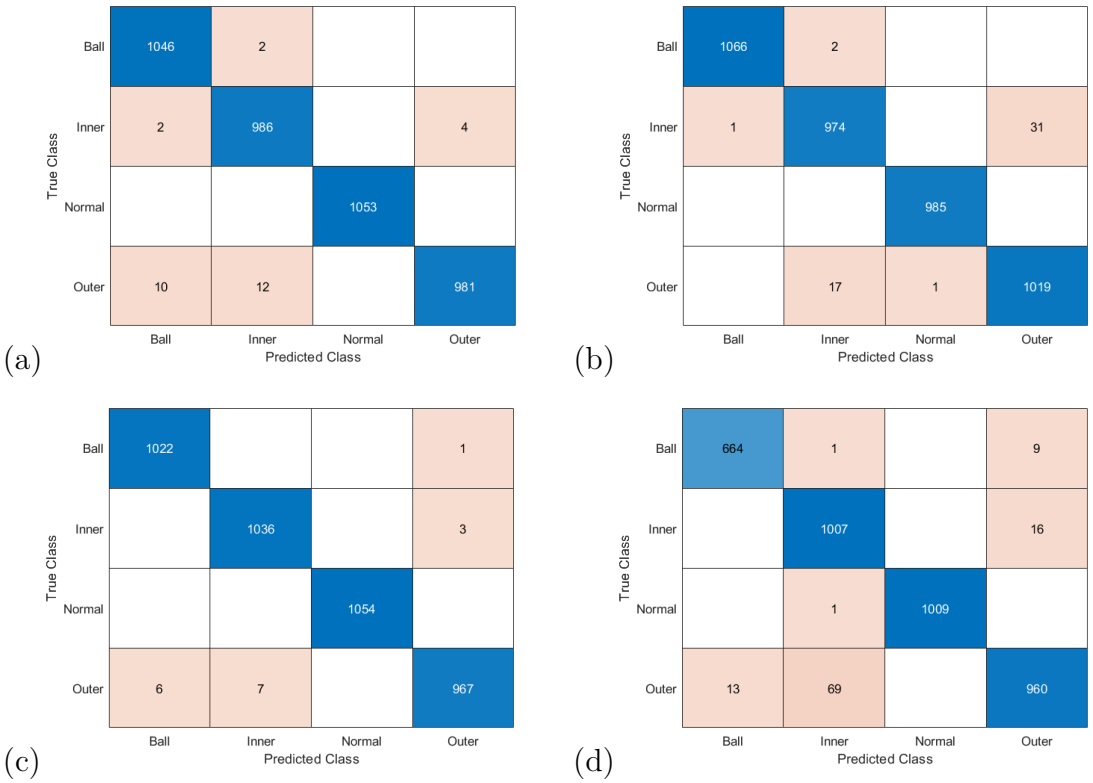


Figure 5.17 Confusion Matrices - HUST Dataset: (a) 6205 (b) 6206 (c) 6207 (d) 6208

5.4 Discussion of the models' performances

The Support Vector Machine (SVM) model using no optimization algorithm achieved the lowest accuracy of all the SVM models on the datasets of fault size 0.021 inches. Integrating the three optimization algorithms, GWO, PSO, and the novel KOA, into the SVM model, one at a time, resulted in a slight increase in accuracy. The GWO proved to yield better results than the novel KOA when coupled with SVMs, while the KOA-SVM model achieved a higher accuracy than the PSO-SVM model. The accuracies, however, were still very low, with the highest being 65.37%. Reducing the datasets by calculating the 8 time-domain parameters further increased the accuracies; the GWO-SVM model's accuracy on the reduced datasets was 85.67%, the highest of the three SVM models. Both the GWO-SVM and the PSO-SVM models performed better than the KOA-SVM model when the reduced datasets were used. This further proves that the novel KOA does not outperform the traditional optimization algorithms in every application.

The default hyper-parameters of the BiLSTM model achieved a bearing fault detection and classification accuracy of 99.51% on the datasets of fault size 0.021 inches. This was already significantly better than the best SVM model. After running through tens of hyper-parameter fine-tuning experiments, it was possible to increase the model's accuracy to 99.92% on the same datasets. Seeing how well the BiLSTM model performed on said datasets, its performance was tested on other datasets. The model showed exceptional accuracies on the datasets with different fault sizes, as well as on a combination of all the fault sizes. Furthermore, the BiLSTM model was tested on the HUST Dataset, and proved to very capable of detecting and classifying bearing faults on this dataset as well.

The deep learning model used in this thesis, with the chosen set of hyper-parameters, proved to be incredibly effective in detecting and classifying faulty bearings with varying fault sizes. When compared with the SVM models, GWO-SVM, PSO-SVM, and the novel KOA-SVM model, it is clear that the Support Vector Machines could not compete with the performance of the BiLSTM model, even when the datasets were reduced to accommodate the limitations of SVMs. The BiLSTM model was able to achieve accuracies very close to 100%, and reached 100% on the smallest fault size, which is a very significant improvement compared to the best SVM model, the GWO-SVM.

6. CONCLUSIONS AND FUTURE WORK

Bearing fault detection is a critical aspect of maintaining the reliability and efficiency of machinery across various industries. By detecting faults early, predictive maintenance strategies can significantly reduce downtime, prevent catastrophic failures, and lower maintenance costs. This thesis explored the integration of popular optimization algorithms, Grey Wolf Optimization (GWO), and Particle Swarm Optimization (PSO), as well as the novel Kepler Optimization Algorithm (KOA) into Support Vector Machine (SVM) models. Additionally, the deep learning model, Bi-directional Long Short-Term Memory model was explored. The models were tested on the Case Western Reserve University (CRWU) Bearing Dataset, and their ability of detecting and classifying bearing faults based on raw accelerometer data was analyzed. The thesis also provided a set of chosen hyper-parameters for the BiLSTM model that proved to achieve exceptional accuracies in bearing fault detection and classification. The BiLSTM model was also tested on the HUST Bearing Dataset to further verify its ability in detecting and classifying bearing faults.

The optimized models proved to significantly outperform the non-optimized SVM model, especially when the datasets' size was reduced to accommodate the SVM's limitation of not being particularly suitable for large datasets. With the reduced datasets, the GWO-SVM model achieved the highest accuracy of the optimized SVM models, with an accuracy of 84.75%, while the PSO-SVM and KOA-SVM models achieved accuracies of 81.98% and 81.08%, respectively. The novel KOA failed to outperform the established GWO and PSO models with the reduced datasets, while it did manage to outperform only the PSO algorithm with the regular non-reduced datasets.

The BiLSTM deep learning model with the set of chosen hyper-parameters achieved exceptional accuracies on the CWRU Bearing Dataset, outperforming the best SVM model greatly. The BiLSTM model managed to detect and classify small (0.007") faults with 100% accuracy, highlighting its outstanding ability in detecting

and classifying bearing faults early. The model's detection and classification accuracies for the bigger fault sizes, 0.014", 0.021", and 0.028" were 99.84%, 99.92%, and 99.9%, respectively. The deep learning model also managed to successfully detect and classify the bearing faults when all the fault sizes were combined, achieving an accuracy 99.62%. The BiLSTM model also achieved very high accuracies on the HUST Bearing Dataset, reaching accuracies as high as 99.58%. The BiLSTM model with the chosen hyper-parameters proved to outperform many state-of-the-art bearing fault detection techniques, with the added advantage that it uses raw unprocessed accelerometer data directly, eliminating the need for data preprocessing and simplifying the process of integrating it into any industrial setup where bearing fault detection is needed.

One possible area of work for future researchers is to test to BiLSTM model with the chosen set of hyper-parameters on more bearing datasets. Other possible areas for future research include:

- Trying to achieve better results with the SVM models by optimizing different parameters
- Optimizing the BiLSTM model hyper-parameters using optimization algorithms
- Integrating different optimization algorithms into the SVM models

BIBLIOGRAPHY

Abdel-Basset, M., Mohamed, R., Azeem, S. A. A., Jameel, M., & Abouhawwash, M. (2023). Kepler optimization algorithm: A new metaheuristic algorithm inspired by kepler's laws of planetary motion. *Knowledge-based systems*, *268*, 110454.

Aherwar, A. (2012). An investigation on gearbox fault detection using vibration analysis techniques: A review. *Australian Journal of Mechanical Engineering*, *10*(2), 169–183.

Azeez, A. A., Alkhedher, M., & Gadala, M. S. (2020). Thermal imaging fault detection for rolling element bearings. In *2020 Advances in Science and Engineering Technology International Conferences (ASET)*, (pp. 1–5). IEEE.

Chen, X., Zhang, B., & Gao, D. (2021). Bearing fault diagnosis base on multi-scale cnn and lstm model. *Journal of Intelligent Manufacturing*, *32*(4), 971–987.

Fei, S.-w. (2017). Fault diagnosis of bearing based on wavelet packet transform-phase space reconstruction-singular value decomposition and svm classifier. *Arabian Journal for Science and Engineering*, *42*(5), 1967–1975.

Gholizadeh, S., Leman, Z., Baharudin, B., et al. (2015). A review of the application of acoustic emission technique in engineering. *Struct. Eng. Mech*, *54*(6), 1075–1095.

Graves, A. & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm networks. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 4, (pp. 2047–2052). IEEE.

Hadden, T. & Hadi, M. U. (2023). Efficient roller element bearing fault detection using wavelet analysis and k-means clustering for variable-speed industrial machinery. *Available at SSRN 4626692*.

Hakim, M., Omran, A. A. B., Ahmed, A. N., Al-Waily, M., & Abdellatif, A. (2023). A systematic review of rolling bearing fault diagnoses based on deep learning and transfer learning: Taxonomy, overview, application, open challenges, weaknesses and recommendations. *Ain Shams Engineering Journal*, *14*(4), 101945.

Hoang, D.-T. & Kang, H.-J. (2019). Rolling element bearing fault diagnosis using convolutional neural network and vibration image. *Cognitive Systems Research*, *53*, 42–50.

Holm-Hansen, B. T. & Gao, R. X. (2000). Vibration analysis of a sensor-integrated ball bearing. *J. Vib. Acoust.*, *122*(4), 384–392.

Hui, K. H., Ooi, C. S., Lim, M. H., & Leong, M. S. (2016). A hybrid artificial neural network with dempster-shafer theory for automated bearing fault diagnosis. *Journal of Vibroengineering*, *18*(7), 4409–4418.

Kanai, R., Desavale, R., & Chavan, S. (2016). Experimental-based fault diagnosis of rolling bearings using artificial neural network. *Journal of Tribology*, *138*(3), 031103.

Kennedy, J. & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, (pp. 1942–1948). ieee.

Li, B., Chow, M.-Y., Tipsuwan, Y., & Hung, J. C. (2000). Neural-network-based

motor rolling bearing fault diagnosis. *IEEE transactions on industrial electronics*, *47*(5), 1060–1069.

Li, K., Su, L., Wu, J., Wang, H., & Chen, P. (2017). A rolling bearing fault diagnosis method based on variational mode decomposition and an improved kernel extreme learning machine. *Applied Sciences*, *7*(10), 1004.

Lin, Z., Ji, Y., & Sun, X. (2023). Landslide displacement prediction based on ceemdan method and cnn–bilstm model. *Sustainability*, *15*(13), 10071.

Liu, H., Zhou, J., Zheng, Y., Jiang, W., & Zhang, Y. (2018). Fault diagnosis of rolling bearings with recurrent neural network-based autoencoders. *ISA transactions*, *77*, 167–178.

Lu, C., Wang, Z., & Zhou, B. (2017). Intelligent fault diagnosis of rolling bearing using hierarchical convolutional network based health state classification. *Advanced Engineering Informatics*, *32*, 139–151.

Mirjalili, S., Mirjalili, S. M., & Lewis, A. (2014). Grey wolf optimizer. *Advances in engineering software*, *69*, 46–61.

Ran, Y., Zhou, X., Lin, P., Wen, Y., & Deng, R. (2019). A survey of predictive maintenance: Systems, purposes and approaches. *arXiv preprint arXiv:1912.07383*.

Salomon, C. P., Ferreira, C., Sant'Ana, W. C., Lambert-Torres, G., Borges da Silva, L. E., Bonaldi, E. L., de Oliveira, L. E. d. L., & Torres, B. S. (2019). A study of fault diagnosis based on electrical signature analysis for synchronous generators predictive maintenance in bulk electric systems. *Energies*, *12*(8), 1506.

Schoen, R. R., Lin, B. K., Habetler, T. G., Schlag, J. H., & Farag, S. (1995). An unsupervised, on-line system for induction motor fault detection using stator current monitoring. *IEEE Transactions on Industry Applications*, *31*(6), 1280–1286.

Selcuk, S. (2017). Predictive maintenance, its implementation and latest trends. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, *231*(9), 1670–1679.

Sharifani, K. & Amini, M. (2023). Machine learning and deep learning: A review of methods and applications. *World Information Technology and Engineering Journal*, *10*(07), 3897–3904.

Shen, W., Xiao, M., Wang, Z., & Song, X. (2023). Rolling bearing fault diagnosis based on support vector machine optimized by improved grey wolf algorithm. *Sensors*, *23*(14), 6645.

Shenfield, A. & Howarth, M. (2020). A novel deep learning model for the detection and identification of rolling element-bearing faults. *Sensors*, *20*(18), 5112.

Thuan, N. D. & Hong, H. S. (2023). Hust bearing dataset. data retrieved from BMC Research Notes, `https://doi.org/10.1186/s13104-023-06400-4`.

Wen, L., Li, X., Gao, L., & Zhang, Y. (2017). A new convolutional neural network-based data-driven fault diagnosis method. *IEEE Transactions on Industrial Electronics*, *65*(7), 5990–5998.

Wu, M. (2024). Rolling bearing fault diagnosis data set.

Xiao, L., Yang, X., & Yang, X. (2023). A graph neural network-based bearing fault detection method. *Scientific Reports*, *13*(1), 5286.

Yang, L., Hu, Q., & Zhang, S. (2020). Fault diagnosis of motor rolling bearing based on imf sample entropy and particle swarm optimization svm. In *IOP Conference Series: Earth and Environmental Science*, volume 461, (pp. 012037).

IOP Publishing.

Youcef Khodja, A., Guersi, N., Saadi, M. N., & Boutasseta, N. (2020). Rolling element bearing fault diagnosis for rotating machinery using vibration spectrum imaging and convolutional neural networks. *The International Journal of Advanced Manufacturing Technology*, *106*, 1737–1751.

Zhang, J., Yi, S., Liang, G., Hongli, G., Xin, H., & Hongliang, S. (2020). A new bearing fault diagnosis method based on modified convolutional neural networks. *Chinese Journal of Aeronautics*, *33*(2), 439–447.

Zhang, S., Zhang, S., Wang, B., & Habetler, T. G. (2020). Deep learning algorithms for bearing fault diagnostics—a comprehensive review. *IEEE Access*, *8*, 29857–29881.

Zonta, T., Da Costa, C. A., da Rosa Righi, R., de Lima, M. J., da Trindade, E. S., & Li, G. P. (2020). Predictive maintenance in the industry 4.0: A systematic literature review. *Computers & Industrial Engineering*, *150*, 106889.

APPENDIX A

Code for preparing Datasets for SVM

```
clear
clc

load('Normal_Dataset.mat')
load('Inner_21.mat')
load('Outer_21.mat')
load('Ball_21.mat')

Normal = X097_DE_time;
Inner = X209_DE_time;
Outer = X234_DE_time;
Ball = X222_DE_time;

Number_Of_Data_Normal = floor(length(Normal)/100);
Number_Of_Data_Inner = floor(length(Inner)/100);
Number_Of_Data_Outer = floor(length(Outer)/100);
Number_Of_Data_Ball = floor(length(Ball)/100);

Number_Of_Data = Number_Of_Data_Normal + Number_Of_Data_Inner
+ Number_Of_Data_Outer + Number_Of_Data_Ball;

All_Data = cell(Number_Of_Data,1);
Labels = strings(Number_Of_Data,1);

temp_normal = [];
temp_inner = [];
temp_outer = [];
temp_ball = [];

for i = 1:length(Normal)

    temp_normal = [temp_normal;Normal(i)];

    if rem(i,100) == 0
```

```
        All_Data{i/100,1} = temp_normal;
        Labels(i/100) = "Normal";
        temp_normal = [];
    end

end

for i = 1:length(Inner)

    temp_inner = [temp_inner;Inner(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal+i/100,1} = temp_inner;
        Labels(Number_Of_Data_Normal+i/100) = "Inner";
        temp_inner = [];
    end

end

for i = 1:length(Outer)

    temp_outer = [temp_outer;Outer(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner+i/100,1} = temp_outer;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner+i/100) = "Outer";
        temp_outer = [];
    end

end

for i = 1:length(Ball)

    temp_ball = [temp_ball;Ball(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
```

```
        +Number_Of_Data_Inner
        +Number_Of_Data_Outer+i/100,1} = temp_ball;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner
        +Number_Of_Data_Outer+i/100) = "Ball";
        temp_ball = [];
    end

end

Labels_Category = categorical(Labels);
classNames = categories(Labels_Category);

numChannels = size(All_Data{1},2);
numObservations = numel(All_Data);
[idxTrain,idxTest] = trainingPartitions(numObservations,[0.8 0.2]);
XTrain = All_Data(idxTrain);
TTrain = Labels_Category(idxTrain);

XTest = All_Data(idxTest);
TTest = Labels_Category(idxTest);

A = cell2mat(XTrain);
Train_Size = size(A,1)/100;
temp_Train = zeros(100,1);
Train_Data = zeros(Train_Size,100);
t1 = 0;

for i = 1:size(A,1)
    temp_Train(i-t1*100) = A(i);

    if rem(i,100) == 0
        Train_Data(i/100,:) = temp_Train';
        t1 = t1 + 1;
        temp_Train = zeros(100,1);
    end
end

B = cell2mat(XTest);
```

```
Test_Size = size(B,1)/100;
temp_Test = zeros(100,1);
Test_Data = zeros(Test_Size,100);
t2 = 0;

for i = 1:size(B,1)
    temp_Test(i-t2*100) = B(i);

    if rem(i,100) == 0
        Test_Data(i/100,:) = temp_Test';
        t2 = t2 +1;
        temp_Test = zeros(100,1);
    end
end

Train_Labels = string(TTrain);
Test_Labels = string(TTest);
```

Code for preparing reduced Datasets for SVM

```
clear
clc

load('Normal_Dataset.mat')
load('Inner_21.mat')
load('Outer_21.mat')
load('Ball_21.mat')

Normal = X097_DE_time;
Inner = X209_DE_time;
Outer = X234_DE_time;
Ball = X222_DE_time;

Number_Of_Data_Normal = floor(length(Normal)/100);
Number_Of_Data_Inner = floor(length(Inner)/100);
Number_Of_Data_Outer = floor(length(Outer)/100);
Number_Of_Data_Ball = floor(length(Ball)/100);
```

```
Number_Of_Data = Number_Of_Data_Normal
+Number_Of_Data_Inner
+Number_Of_Data_Outer + Number_Of_Data_Ball;

All_Data = cell(Number_Of_Data,1);
Labels = strings(Number_Of_Data,1);

temp_normal = [];
temp_inner = [];
temp_outer = [];
temp_ball = [];

for i = 1:length(Normal)

    temp_normal = [temp_normal;Normal(i)];

    if rem(i,100) == 0
        All_Data{i/100,1} = temp_normal;
        Labels(i/100) = "Normal";
        temp_normal = [];
    end

end

for i = 1:length(Inner)

    temp_inner = [temp_inner;Inner(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal+i/100,1} = temp_inner;
        Labels(Number_Of_Data_Normal+i/100) = "Inner";
        temp_inner = [];
    end

end

for i = 1:length(Outer)
```

```
    temp_outer = [temp_outer;Outer(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner+i/100,1} = temp_outer;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner+i/100) = "Outer";
        temp_outer = [];
    end

end

for i = 1:length(Ball)

    temp_ball = [temp_ball;Ball(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner
        +Number_Of_Data_Outer+i/100,1} = temp_ball;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner
        +Number_Of_Data_Outer+i/100) = "Ball";
        temp_ball = [];
    end

end

Labels_Category = categorical(Labels);
classNames = categories(Labels_Category);

numChannels = size(All_Data{1},2);
numObservations = numel(All_Data);
[idxTrain,idxTest] = trainingPartitions(numObservations,[0.8 0.2]);
XTrain = All_Data(idxTrain);
TTrain = Labels_Category(idxTrain);

XTest = All_Data(idxTest);
TTest = Labels_Category(idxTest);
```

```
A = cell2mat(XTrain);
Train_Size = size(A,1)/100;
temp_Train = zeros(100,1);
Train_Data = zeros(Train_Size,8);
t1 = 0;

for i = 1:size(A,1)
    temp_Train(i-t1*100) = A(i);

    if rem(i,100) == 0

        temp_1 = zeros(1,8);
        temp_1(1) = mean(temp_Train);
        temp_1(2) = max(temp_Train) - min(temp_Train);
        temp_1(3) = rms(temp_Train);
        temp_1(4) = std(temp_Train);

        temp_Train_X = 0;
        for j = 1:100
            temp_Train_X = temp_Train_X
            +sqrt(abs(temp_Train(j)));
        end

        temp_1(5) = max(abs(temp_Train))/((temp_Train_X/100)^2);
        temp_1(6) = max(abs(temp_Train))/abs(mean(temp_Train));
        temp_1(7) = max(abs(temp_Train))/rms(temp_Train);

        temp_Train_Y = 0;
        temp_Train_Z = 0;

        for k = 1:100
            temp_Train_Y = temp_Train_Y + (temp_Train(k)
            -mean(temp_Train))^4;
            temp_Train_Z = temp_Train_Z + (temp_Train(k)
            -mean(temp_Train))^2;
        end

        temp_1(8) = (temp_Train_Y/100)/((temp_Train_Z/100)^2);
```

```
        Train_Data(i/100,:) = temp_1';
        t1 = t1 + 1;
        temp_Train = zeros(100,1);
        temp_1 = zeros(1,8);
    end
end

B = cell2mat(XTest);
Test_Size = size(B,1)/100;
temp_Test = zeros(100,1);
Test_Data = zeros(Test_Size,8);
t2 = 0;

for i = 1:size(B,1)
    temp_Test(i-t2*100) = B(i);

    if rem(i,100) == 0

        temp_2 = zeros(1,8);
        temp_2(1) = mean(temp_Test);
        temp_2(2) = max(temp_Test) - min(temp_Test);
        temp_2(3) = rms(temp_Test);
        temp_2(4) = std(temp_Test);

        temp_Train_X = 0;
        for j = 1:100
            temp_Train_X = temp_Train_X
            +sqrt(abs(temp_Test(j)));
        end

        temp_2(5) = max(abs(temp_Test))/((temp_Train_X/100)^2);
        temp_2(6) = max(abs(temp_Test))/abs(mean(temp_Test));
        temp_2(7) = max(abs(temp_Test))/rms(temp_Test);

        temp_Train_Y = 0;
        temp_Train_Z = 0;

        for k = 1:100
```

```
            temp_Train_Y = temp_Train_Y + (temp_Test(k)
            -mean(temp_Test))^4;
            temp_Train_Z = temp_Train_Z + (temp_Test(k)
            -mean(temp_Test))^2;
        end

        temp_2(8) = (temp_Train_Y/100)/((temp_Train_Z/100)^2);

        Test_Data(i/100,:) = temp_2';
        t2 = t2 +1;
        temp_Test = zeros(100,1);
        temp_2 = zeros(1,8);
    end
end

Train_Labels = string(TTrain);
Test_Labels = string(TTest);
```

GWO-SVM Main Program

```
SearchAgents_no=10; % Number of search agents
Function_name='F1'; % Test function
Max_iteration=300; % Maximum number of iterations
Training = Train_Data;
Classes = Train_Labels;

% Load details of the selected benchmark function
[lb,ub,dim,fobj] = GWO_Get_Functions_details(Function_name);
[Best_score,Best_pos,GWO_cg_curve] =
GWO(SearchAgents_no,Max_iteration,lb,ub,dim,fobj,Training,Classes);

figure('Position',[500 500 660 290])
semilogy(GWO_cg_curve,'Color','r')
title('Objective space')
xlabel('Iteration');
ylabel('Best score obtained so far');
axis tight
grid on
box on
legend('GWO')

display(['The best solution obtained by GWO is : ',
num2str(Best_pos)]);

display(['The best optimal value of the objective funciton
found by GWO is : ', num2str(Best_score)]);

time_in_seconds = toc;
time_in_hours = time_in_seconds/3600;

t = templateSVM("BoxConstraint",Best_pos(1),"KernelScale",
Best_pos(2));
Mdl = fitcecoc(Training,Classes,"Learners",t);
TestPredictedLabelsCell = predict(Mdl,Test_Data);
TestPredictedLabels = string(TestPredictedLabelsCell);
acc = mean(TestPredictedLabels == Test_Labels);
```

```
figure
confusionchart(TestPredictedLabels,Test_Labels)
```

GWO-SVM Initialization

```
% This function initializes the first population of search agents
function Positions = GWO_Initialization(SearchAgents_no,dim,ub,lb)

Boundary_no= size(ub,2); % number of boundaries

% If the boundaries of all variables are equal and user
entered a single number for both upper bound and lower bound
if Boundary_no==1
    Positions=rand(SearchAgents_no,dim).*(ub-lb)+lb;
end

% If each variable has a different upper bound and lower bound
if Boundary_no>1
    for i=1:dim
        ub_i=ub(i);
        lb_i=lb(i);
        Positions(:,i)=rand(SearchAgents_no,1).*(ub_i-lb_i)+lb_i;
    end
end
```

GWO-SVM Objective Function Details

```
% This function containts full information and implementations of
the objective function

% lb is the lower bound: lb=[lb_1,lb_2,...,lb_d]
% up is the uppper bound: ub=[ub_1,ub_2,...,ub_d]
% dim is the number of variables (dimension of the problem)

function [lb,ub,dim,fobj] = GWO_Get_Functions_details(F)
```

```
switch F
    case 'F1'
        fobj = @F1;
        lb = [0.0001,0.0001];
        ub = [100,100];
        dim = 2;
end

end


function o = F1(parameters,Training,Classes)

t = templateSVM("BoxConstraint",parameters(1),"KernelScale",
parameters(2));
Mdl = fitcecoc(Training,Classes,"Learners",t);

o = resubLoss(Mdl);

end
```

GWO-SVM Main Code

```
% Grey Wolf Optimizer
function [Alpha_score,Alpha_pos,Convergence_curve] =
GWO(SearchAgents_no,Max_iter,lb,ub,dim,fobj,Training,Classes)

% initialize alpha, beta, and delta wolves' positions
Alpha_pos=zeros(1,dim);
Alpha_score=inf; %change this to -inf for maximization problems

Beta_pos=zeros(1,dim);
Beta_score=inf; %change this to -inf for maximization problems

Delta_pos=zeros(1,dim);
Delta_score=inf; %change this to -inf for maximization problems
```

```
%Initialize the positions of search agents
Positions = GWO_Initialization(SearchAgents_no,dim,ub,lb);

Convergence_curve=zeros(1,Max_iter);

l=0;% Loop counter

% Main loop
while l<Max_iter
    for i=1:size(Positions,1)

       % Return back the search agents that go beyond the boundaries
       of the search space
        Flag4ub=Positions(i,:)>ub;
        Flag4lb=Positions(i,:)<lb;
        Positions(i,:)=(Positions(i,:).*(~(Flag4ub+Flag4lb)))
        +ub.*Flag4ub+lb.*Flag4lb;

        % Calculate objective function for each search agent
        fitness=fobj(Positions(i,:),Training,Classes);

        % Update Alpha, Beta, and Delta
        if fitness<Alpha_score
            Alpha_score=fitness; % Update alpha
            Alpha_pos=Positions(i,:);
        end

        if fitness>Alpha_score && fitness<Beta_score
            Beta_score=fitness; % Update beta
            Beta_pos=Positions(i,:);
        end

        if fitness>Alpha_score && fitness>Beta_score &&
        fitness<Delta_score
            Delta_score=fitness; % Update delta
            Delta_pos=Positions(i,:);
        end
    end
```

```
    a=2-l*((2)/Max_iter); % a decreases linearly fron 2 to 0
    % Update the Position of search agents including omegas
    for i=1:size(Positions,1)
        for j=1:size(Positions,2)
            r1=rand(); % r1 is a random number in [0,1]
            r2=rand(); % r2 is a random number in [0,1]
            A1=2*a*r1-a;
            C1=2*r2;
            D_alpha=abs(C1*Alpha_pos(j)-Positions(i,j));
            X1=Alpha_pos(j)-A1*D_alpha;
            r1=rand();
            r2=rand();
            A2=2*a*r1-a;
            C2=2*r2;
            D_beta=abs(C2*Beta_pos(j)-Positions(i,j));
            X2=Beta_pos(j)-A2*D_beta;
            r1=rand();
            r2=rand();
            A3=2*a*r1-a;
            C3=2*r2;
            D_delta=abs(C3*Delta_pos(j)-Positions(i,j));
            X3=Delta_pos(j)-A3*D_delta;
            Positions(i,j)=(X1+X2+X3)/3;
        end
    end
    l=l+1;
    Convergence_curve(l)=Alpha_score;
end
end
```

PSO-SVM Main Program + Initialization + Code

```
%% Problem Definition
CostFunction=@PSO_my_objfunc;        % Cost Function
nVar=2;             % Number of Decision Variables
VarSize=[1 nVar];   % Size of Decision Variables Matrix
VarMin = 0.0001;          % Lower Bound of Variables
VarMax = 1000;          % Upper Bound of Variables
Training = Train_Data;
Classes = Train_Labels;
%% PSO Parameters
MaxIt=300;      % Maximum Number of Iterations
nPop=10;        % Population Size (Swarm Size)
% PSO Parameters
w=1;            % Inertia Weight
wdamp=0.99;     % Inertia Weight Damping Ratio
c1=1.5;         % Personal Learning Coefficient
c2=2.0;         % Global Learning Coefficient
% Velocity Limits
VelMax=0.1*(VarMax-VarMin);
VelMin=-VelMax;
%% Initialization
empty_particle.Position=[];
empty_particle.Cost=[];
empty_particle.Velocity=[];
empty_particle.Best.Position=[];
empty_particle.Best.Cost=[];
particle=repmat(empty_particle,nPop,1);
GlobalBest.Cost=inf;
for i=1:nPop

    % Initialize Position
    particle(i).Position=unifrnd(VarMin,VarMax,VarSize);

    % Initialize Velocity
    particle(i).Velocity=zeros(VarSize);

    % Evaluation
```

```
    particle(i).Cost=CostFunction(particle(i).Position,
    Training,Classes);

    % Update Personal Best
    particle(i).Best.Position=particle(i).Position;
    particle(i).Best.Cost=particle(i).Cost;

    % Update Global Best
    if particle(i).Best.Cost<GlobalBest.Cost

        GlobalBest=particle(i).Best;

    end
end

BestCost=zeros(MaxIt,1);
%% PSO Main Loop
for it=1:MaxIt

    for i=1:nPop

        % Update Velocity
        particle(i).Velocity = w*particle(i).Velocity ...
            +c1*rand(VarSize).*(particle(i).Best.Position
            -particle(i).Position) ...
            +c2*rand(VarSize).*(GlobalBest.Position
            -particle(i).Position);

        % Apply Velocity Limits
        particle(i).Velocity = max(particle(i).Velocity,VelMin);
        particle(i).Velocity = min(particle(i).Velocity,VelMax);

        % Update Position
        particle(i).Position = particle(i).Position
        +particle(i).Velocity;

        % Velocity Mirror Effect
        IsOutside=(particle(i).Position<VarMin |
        particle(i).Position>VarMax);
```

```
        particle(i).Velocity(IsOutside)=
        -particle(i).Velocity(IsOutside);

        % Apply Position Limits
        particle(i).Position = max(particle(i).Position,VarMin);
        particle(i).Position = min(particle(i).Position,VarMax);

        % Evaluation
        particle(i).Cost = CostFunction(particle(i).Position,
        Training,Classes);

        % Update Personal Best
        if particle(i).Cost<particle(i).Best.Cost

            particle(i).Best.Position=particle(i).Position;
            particle(i).Best.Cost=particle(i).Cost;

            % Update Global Best
            if particle(i).Best.Cost<GlobalBest.Cost

                GlobalBest=particle(i).Best;

            end

        end

    end

    BestCost(it)=GlobalBest.Cost;

    disp(['Iteration ' num2str(it) ': Best Cost =
    ' num2str(BestCost(it))]);
    w=w*wdamp;
end

BestSol = GlobalBest;
%% Results
figure;
%plot(BestCost,'LineWidth',2);
```

```
semilogy(BestCost,'LineWidth',2);
xlabel('Iteration');
ylabel('Best Cost');
grid on;
time_in_seconds = toc;
time_in_hours = time_in_seconds/3600;

t = templateSVM("BoxConstraint",BestSol.Position(1),
"KernelScale",BestSol.Position(2));
Mdl = fitcecoc(Train_Data,Train_Labels,"Learners",t);
TestPredictedLabelsCell = predict(Mdl,Test_Data);
TestPredictedLabels = string(TestPredictedLabelsCell);
acc = mean(TestPredictedLabels == Test_Labels);
figure
confusionchart(TestPredictedLabels,Test_Labels)
```

PSO-SVM Objective Function

```
function result = PSO_my_objfunc(Positions, Training, Classes)
    t = templateSVM("BoxConstraint",Positions(1),"KernelScale",
    Positions(2));
    Mdl = fitcecoc(Training,Classes,"Learners",t);
    result = resubLoss(Mdl);
end
```

KOA-SVM Main Program

```
N=30; % Number of search agents (Planets)
Tmax=1500; % Maximum number of Function evaluations

fhd = @KOA_my_objfunc;
%fobj = 1;
Training = Train_Data;
Classes = Train_Labels;
[lb,ub,dim] = KOA_Get_Functions_detailsCEC(1);
[Best_score,Best_pos,Convergence_curve]=
KOA(N,Tmax,ub,lb,dim,fhd,Training,Classes);
fitness = Best_score;
% Print the best score
fprintf(['Best Score:\t', num2str(fitness), '\n']);
%Best_pos
% Plotting the convergence curve
figure(1)
h=semilogy(Convergence_curve,'.-','MarkerSize',20,
'Color','red','LineWidth',1.5);
h.MarkerIndices = 1000:4000:Tmax;
xlabel('Iteration');
ylabel('Best Fitness obtained so-far');
axis tight
grid off
box on
legend({'KOA'});
time_in_seconds = toc;
time_in_hours = time_in_seconds/3600;

t = templateSVM("BoxConstraint",Best_pos(1),
"KernelScale",Best_pos(2));
Mdl = fitcecoc(Train_Data,Train_Labels,"Learners",t);
TestPredictedLabelsCell = predict(Mdl,Test_Data);
TestPredictedLabels = string(TestPredictedLabelsCell);
acc = mean(TestPredictedLabels == Test_Labels)
figure
confusionchart(TestPredictedLabels,Test_Labels)
```

KOA-SVM Initialization

```
% This function initialize the first population of
search agents
function Positions =
KOA_Initialization(SearchAgents_no,dim,ub,lb)

Boundary_no= length(ub); % numnber of boundaries

% If the boundaries of all variables are equal and user
entered a signle number for both ub and lb
if Boundary_no==1
     Positions=rand(SearchAgents_no,dim).*(ub-lb)+lb;
end

% If each variable has a different lb and ub
if Boundary_no>1
    for i=1:dim
        ub_i=ub(i);
        lb_i=lb(i);
         Positions(:,i)=rand(SearchAgents_no,1).*(ub_i-lb_i)+lb_i;
    end
end
```

KOA-SVM Function Details

```
% This function containts full information and implementations
of the benchmark function

% lb is the lower bound: lb=[lb_1,lb_2,...,lb_d]
% up is the uppper bound: ub=[ub_1,ub_2,...,ub_d]
% dim is the number of variables (dimension of the problem)

function [lb,ub,dim] = KOA_Get_Functions_detailsCEC(F)

switch F
```

```
   case 1
       dim=2;
       lb=0.0001*ones(1,dim);
       ub=100*ones(1,dim);
 end
end
```

KOA-SVM Objective Function

```
    function result = KOA_my_objfunc(Positions,
    Training, Classes)
    t = templateSVM("BoxConstraint",Positions(1),
    "KernelScale",Positions(2));
    Mdl = fitcecoc(Training,Classes,"Learners",t);
    result = resubLoss(Mdl);
end
```

KOA-SVM Main Code

```
% The Kepler Optimization Algorithm
function [Sun_Score,Sun_Pos,Convergence_curve]=
KOA(SearchAgents_no,Tmax,ub,lb,dim,fhd,Training,Classes)

%%%%-------------------Definitions--------------------%%
%%
Sun_Pos=zeros(1,dim); % A vector to include the best-so-far
Solution, representing the Sun
Sun_Score=inf; % A Scalar variable to include the best-so-far score
Convergence_curve=zeros(1,Tmax);

%%-------------------Controlling parameters--------------%%
%%
Tc=3;
M0=0.1;
lambda=15;
```

```
%% Step 1: Initialization process
%%---------------Initialization----------------------%%
 % Orbital Eccentricity (e)
orbital=rand(1,SearchAgents_no); %% Eq.(4)
 %% Orbital Period (T)
T=abs(randn(1,SearchAgents_no)); %% Eq.(5)
Positions = KOA_Initialization(SearchAgents_no,dim,ub,lb);
% Initialize the positions of planets
t=0; %% Function evaluation counter
%%
%%--------------------Evaluation---------------%%
for i=1:SearchAgents_no
    %% Test suites of CEC-2014, CEC-2017, CEC-2020, and CEC-2022
    PL_Fit(i)=feval(fhd, Positions(i,:), Training, Classes);
    % Update the best-so-far solution
    if PL_Fit(i)<Sun_Score % Change this to > for maximization
    problems
       Sun_Score=PL_Fit(i); % Update the best-so-far score
       Sun_Pos=Positions(i,:); % Update te best-so-far solution
    end
end
while t<Tmax %% Termination condition
 [Order] = sort(PL_Fit);  % Sorting the fitness values of the
 solutions in current population
 %% The worst Fitness value at function evaluation t
 worstFitness = Order(SearchAgents_no);
 M=M0*(exp(-lambda*(t/Tmax)));
 %% Computer R that represents the Euclidian distance between
 the best-so-far solution and the ith solution
 for i=1:SearchAgents_no
    R(i)=0;
    for j=1:dim
       R(i)=R(i)+(Sun_Pos(j)-Positions(i,j))^2;
    end
    R(i)=sqrt(R(i));
 end
%% The mass of the Sun and object i at time t is computed
as follows:
 for i=1:SearchAgents_no
```

```
    sum=0;
    for k=1:SearchAgents_no
        sum=sum+(PL_Fit(k)-worstFitness);
    end
    MS(i)=rand*(Sun_Score-worstFitness)/(sum);
    m(i)=(PL_Fit(i)-worstFitness)/(sum);
end
%% Step 2: Defining the gravitational force (F)
% Computing the attraction force of the Sun and the ith planet
according to the universal law of gravitation:
for i=1:SearchAgents_no
    Rnorm(i)=(R(i)-min(R))/(max(R)-min(R));
    MSnorm(i)=(MS(i)-min(MS))/(max(MS)-min(MS));
    %% The normalized MS
    Mnorm(i)=(m(i)-min(m))/(max(m)-min(m));
    %% The normalized m
    Fg(i)=orbital(i)*M*((MSnorm(i)*Mnorm(i))
    /(Rnorm(i)*Rnorm(i)+eps))+(rand);
end
%% a1 represents the semimajor axis of the elliptical
orbit of object i at time t,
for i=1:SearchAgents_no
    a1(i)=rand*(T(i)^2*(M*(MS(i)+m(i))/(4*pi*pi)))^(1/3);
end

for i=1:SearchAgents_no
    %% a2 is a cyclic controlling parameter that is decreasing
    gradually from -1 to ?2
    a2=-1+-1*(rem(t,Tmax/Tc)/(Tmax/Tc));
    %% a2 is a linearly decreasing factor from 1 to 2
    n=(a2-1)*rand+1;
    a=randi(SearchAgents_no);  %% An index of a solution
    selected at random
    b=randi(SearchAgents_no); %% An index of a solution
    selected at random
    rd=rand(1,dim); %% A vector generated according to
    the normal distribution
    r=rand; %% r1 is a random number in [0,1]
    %% A randomly-assigned binary vector
```

```
U1=rd<r;
O_P=Positions(i,:); %% Storing the current position
of the ith solution
%% Step 6: Updating distance with the Sun
if rand<rand
     %% h is an adaptive factor for controlling the
     distance between the Sun and the current planet at time t
     h=(1/(exp(n.*randn))); %% Eq.(27)
     %% An verage vector based on three solutions:
     the Current solution, best-so-far solution,
     and randomly-selected solution
     Xm=(Positions(b,:)+Sun_Pos+Positions(i,:))/3.0;
     Positions(i,:)=Positions(i,:).*U1+(Xm+h
     .*(Xm-Positions(a,:))).*(1-U1);
else
    %% Step 3: Calculating an object' velocity
    % A flag to opposite or leave the search direction
    of the current planet
     if rand<0.5 %% Eq.(18)
       f=1;
     else
       f=-1;
     end
     L=(M*(MS(i)+m(i))*abs((2/(R(i)+eps))-(1/(a1(i)
     +eps))))^(0.5); %% Eq.(15)
     U=rd>rand(1,dim); %% A binary vector
     if Rnorm(i)<0.5 %% Eq.(13)
        M=(rand.*(1-r)+r); %% Eq.(16)
        l=L*M*U; %% Eq.(14)
        Mv=(rand*(1-rd)+rd); %% Eq.(20)
        l1=L.*Mv.*(1-U);%% Eq.(19)
        V(i,:)=l.*(2*rand*Positions(i,:)
        -Positions(a,:))+l1.*(Positions(b,:)
        -Positions(a,:))+(1-Rnorm(i))*f*U1.*rand(1,dim)
        .*(ub-lb); %% Eq.(13a)
     else
        U2=rand>rand; %% Eq. (22)
        V(i,:)=rand.*L.*(Positions(a,:)-Positions(i,:))
        +(1-Rnorm(i))*f*U2*rand(1,dim).*(rand*ub-lb);
```

```
      end %% End IF

      %% Step 4: Escaping from the local optimum
      % Update the flag f to opposite or leave the search
      direction of the current planet
      if rand<0.5
         f=1;
      else
         f=-1;
      end
      %% Step 5
      Positions(i,:)=((Positions(i,:)+V(i,:).*f)
      +(Fg(i)+abs(randn))*U.*(Sun_Pos-Positions(i,:)));
   end %% End If
   %% Return the search agents that exceed the search
   space's bounds
   if rand<rand
      for j=1:size(Positions,2)
         if  Positions(i,j)>ub(j)
             Positions(i,j)=lb(j)+rand*(ub(j)-lb(j));
         elseif  Positions(i,j)<lb(j)
             Positions(i,j)=lb(j)+rand*(ub(j)-lb(j));
         end %% End If
      end   %% End For
   else
      Positions(i,:) = min(max(Positions(i,:),lb),ub);
   end %% End If
   %% Test suites of CEC-2014, CEC-2017, CEC-2020,
   and CEC-2022
   % Calculate objective function for each search agent
   PL_Fit1=feval(fhd, Positions(i,:)',Training,Classes);
%  Step 7: Elitism, Eq.(30)
   if PL_Fit1<PL_Fit(i) % Change this to > for maximization
   problems
      PL_Fit(i)=PL_Fit1; %
      % Update the best-so-far solution
      if PL_Fit(i)<Sun_Score % Change this to > for maximization
      problems
          Sun_Score=PL_Fit(i); % Update the best-so-far
```

```
            score
            Sun_Pos=Positions(i,:); % Update te best-so-far
            solution
        end
    else
        Positions(i,:)=O_P;
    end %% End IF
    t=t+1; %% Increment the current function evaluation
    if t>Tmax %% Checking the termination condition
        break;
    end %% End IF
    Convergence_curve(t)=Sun_Score; %% Set the best-so-far
    fitness value at function evaluation t in the convergence curve
 end %% End for i
end %% End while
Convergence_curve(t-1)=Sun_Score;
end%% End Function
```

Data Preparation for BiLSTM - One Fault Size (4 Classes)

```
clear
clc

load('Normal_Dataset.mat')
load('Inner_Dataset.mat')
load('Outer_Dataset.mat')
load('Ball_Dataset.mat')

Normal = X097_DE_time;
Inner = X169_DE_time;
Outer = X197_DE_time;
Ball = X185_DE_time;

Number_Of_Data_Normal = floor(length(Normal)/100);
Number_Of_Data_Inner = floor(length(Inner)/100);
Number_Of_Data_Outer = floor(length(Outer)/100);
Number_Of_Data_Ball = floor(length(Ball)/100);

Number_Of_Data = Number_Of_Data_Normal
+ Number_Of_Data_Inner
+ Number_Of_Data_Outer
+ Number_Of_Data_Ball;

All_Data = cell(Number_Of_Data,1);
Labels = strings(Number_Of_Data,1);

temp_normal = [];
temp_inner = [];
temp_outer = [];
temp_ball = [];

for i = 1:length(Normal)

    temp_normal = [temp_normal;Normal(i)];

    if rem(i,100) == 0
```

```
        All_Data{i/100,1} = temp_normal;
        Labels(i/100) = "Normal";
        temp_normal = [];
    end

end

for i = 1:length(Inner)

    temp_inner = [temp_inner;Inner(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +i/100,1} = temp_inner;
        Labels(Number_Of_Data_Normal
        +i/100) = "Inner";
        temp_inner = [];
    end

end

for i = 1:length(Outer)

    temp_outer = [temp_outer;Outer(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner+i/100,1} = temp_outer;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner+i/100) = "Outer";
        temp_outer = [];
    end

end

for i = 1:length(Ball)

    temp_ball = [temp_ball;Ball(i)];
```

```
    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner
        +Number_Of_Data_Outer+i/100,1} = temp_ball;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner
        +Number_Of_Data_Outer+i/100) = "Ball";
        temp_ball = [];
    end

end
```

Data Preparation for BiLSTM - All Fault Sizes (12 Classes)

```
clear
clc

load('Normal_Dataset.mat')
load('Inner_7.mat')
load('Outer_7.mat')
load('Ball_7.mat')
load('Inner_14.mat')
load('Outer_14.mat')
load('Ball_14.mat')
load('Inner_21.mat')
load('Outer_21.mat')
load('Ball_21.mat')
load('Inner_28.mat')
load('Ball_28.mat')

Normal = X097_DE_time;
Inner_7 = X105_DE_time;
Outer_7 = X130_DE_time;
Ball_7 = X118_DE_time;
Inner_14 = X169_DE_time;
Outer_14 = X197_DE_time;
Ball_14 = X185_DE_time;
```

```
Inner_21 = X209_DE_time;
Outer_21 = X234_DE_time;
Ball_21 = X185_DE_time;
Inner_28 = X056_DE_time;
Ball_28 = X048_DE_time;

Number_Of_Data_Normal = floor(length(Normal)/100);
Number_Of_Data_Inner_7 = floor(length(Inner_7)/100);
Number_Of_Data_Outer_7 = floor(length(Outer_7)/100);
Number_Of_Data_Ball_7 = floor(length(Ball_7)/100);
Number_Of_Data_Inner_14 = floor(length(Inner_14)/100);
Number_Of_Data_Outer_14 = floor(length(Outer_14)/100);
Number_Of_Data_Ball_14 = floor(length(Ball_14)/100);
Number_Of_Data_Inner_21 = floor(length(Inner_21)/100);
Number_Of_Data_Outer_21 = floor(length(Outer_21)/100);
Number_Of_Data_Ball_21 = floor(length(Ball_21)/100);
Number_Of_Data_Inner_28 = floor(length(Inner_28)/100);
Number_Of_Data_Ball_28 = floor(length(Ball_28)/100);

Number_Of_Data = Number_Of_Data_Normal
+ Number_Of_Data_Inner_7 + Number_Of_Data_Outer_7
+ Number_Of_Data_Ball_7 + Number_Of_Data_Inner_14
+ Number_Of_Data_Outer_14 + Number_Of_Data_Ball_14
+ Number_Of_Data_Inner_21 + Number_Of_Data_Outer_21
+ Number_Of_Data_Ball_21 + Number_Of_Data_Inner_28
+ Number_Of_Data_Ball_28;

All_Data = cell(Number_Of_Data,1);
Labels = strings(Number_Of_Data,1);

temp_normal = [];
temp_inner_7 = [];
temp_outer_7 = [];
temp_ball_7 = [];
temp_inner_14 = [];
temp_outer_14 = [];
temp_ball_14 = [];
temp_inner_21 = [];
temp_outer_21 = [];
```

```
temp_ball_21 = [];
temp_inner_28 = [];
temp_ball_28 = [];

for i = 1:length(Normal)

    temp_normal = [temp_normal;Normal(i)];

    if rem(i,100) == 0
        All_Data{i/100,1} = temp_normal;
        Labels(i/100) = "Normal";
        temp_normal = [];
    end

end

for i = 1:length(Inner_7)

    temp_inner_7 = [temp_inner_7;Inner_7(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +i/100,1} = temp_inner_7;
        Labels(Number_Of_Data_Normal
        +i/100) = "Inner_7";
        temp_inner_7 = [];
    end

end

for i = 1:length(Outer_7)

    temp_outer_7 = [temp_outer_7;Outer_7(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+i/100,1} = temp_outer_7;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+i/100) = "Outer_7";
```

```
        temp_outer_7 = [];
    end

end

for i = 1:length(Ball_7)

    temp_ball_7 = [temp_ball_7;Ball_7(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +i/100,1} = temp_ball_7;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +i/100) = "Ball_7";
        temp_ball_7 = [];
    end

end

for i = 1:length(Inner_14)

    temp_inner_14 = [temp_inner_14;Inner_14(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+i/100,1} = temp_inner_14;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+i/100) = "Inner_14";
        temp_inner_14 = [];
    end

end

for i = 1:length(Outer_14)
```

```
    temp_outer_14 = [temp_outer_14;Outer_14(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +i/100,1} = temp_outer_14;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +i/100) = "Outer_14";
        temp_outer_14 = [];
    end

end

for i = 1:length(Ball_14)

    temp_ball_14 = [temp_ball_14;Ball_14(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+i/100,1} = temp_ball_14;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+i/100) = "Ball_14";
        temp_ball_14 = [];
    end

end

for i = 1:length(Inner_21)

    temp_inner_21 = [temp_inner_21;Inner_21(i)];

    if rem(i,100) == 0
```

```
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+Number_Of_Data_Ball_14
        +i/100,1} = temp_inner_21;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+Number_Of_Data_Ball_14
        +i/100) = "Inner_21";
        temp_inner_21 = [];
    end

end

for i = 1:length(Outer_21)

    temp_outer_21 = [temp_outer_21;Outer_21(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+Number_Of_Data_Ball_14
        +Number_Of_Data_Inner_21+i/100,1} = temp_outer_21;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+Number_Of_Data_Ball_14
        +Number_Of_Data_Inner_21+i/100) = "Outer_21";
        temp_outer_21 = [];
    end

end

for i = 1:length(Ball_21)

    temp_ball_21 = [temp_ball_21;Ball_21(i)];
```

```
    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+Number_Of_Data_Ball_14
        +Number_Of_Data_Inner_21+Number_Of_Data_Outer_21
        +i/100,1} = temp_ball_21;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+Number_Of_Data_Ball_14
        +Number_Of_Data_Inner_21+Number_Of_Data_Outer_21
        +i/100) = "Ball_21";
        temp_ball_21 = [];
    end

end

for i = 1:length(Inner_28)

    temp_inner_28 = [temp_inner_28;Inner_28(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+Number_Of_Data_Ball_14
        +Number_Of_Data_Inner_21+Number_Of_Data_Outer_21
        +Number_Of_Data_Ball_21+i/100,1} = temp_inner_28;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+Number_Of_Data_Ball_14
        +Number_Of_Data_Inner_21+Number_Of_Data_Outer_21
        +Number_Of_Data_Ball_21+i/100) = "Inner_28";
        temp_inner_28 = [];
    end

end
```

```
for i = 1:length(Ball_28)

    temp_ball_28 = [temp_ball_28;Ball_28(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+Number_Of_Data_Ball_14
        +Number_Of_Data_Inner_21+Number_Of_Data_Outer_21
        +Number_Of_Data_Ball_21+Number_Of_Data_Inner_28
        +i/100,1} = temp_ball_28;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+Number_Of_Data_Ball_14
        +Number_Of_Data_Inner_21+Number_Of_Data_Outer_21
        +Number_Of_Data_Ball_21+Number_Of_Data_Inner_28
        +i/100) = "Ball_28";
        temp_ball_28 = [];
    end

end
```

Data Preparation for BiLSTM - All Fault Sizes (4 Classes)

```
clear
clc

load('Normal_Dataset.mat')
load('Inner_7.mat')
load('Outer_7.mat')
load('Ball_7.mat')
load('Inner_14.mat')
load('Outer_14.mat')
load('Ball_14.mat')
```

```
load('Inner_21.mat')
load('Outer_21.mat')
load('Ball_21.mat')
load('Inner_28.mat')
load('Ball_28.mat')

Normal = X097_DE_time;
Inner_7 = X105_DE_time;
Outer_7 = X130_DE_time;
Ball_7 = X118_DE_time;
Inner_14 = X169_DE_time;
Outer_14 = X197_DE_time;
Ball_14 = X185_DE_time;
Inner_21 = X209_DE_time;
Outer_21 = X234_DE_time;
Ball_21 = X185_DE_time;
Inner_28 = X056_DE_time;
Ball_28 = X048_DE_time;

Number_Of_Data_Normal = floor(length(Normal)/100);
Number_Of_Data_Inner_7 = floor(length(Inner_7)/100);
Number_Of_Data_Outer_7 = floor(length(Outer_7)/100);
Number_Of_Data_Ball_7 = floor(length(Ball_7)/100);
Number_Of_Data_Inner_14 = floor(length(Inner_14)/100);
Number_Of_Data_Outer_14 = floor(length(Outer_14)/100);
Number_Of_Data_Ball_14 = floor(length(Ball_14)/100);
Number_Of_Data_Inner_21 = floor(length(Inner_21)/100);
Number_Of_Data_Outer_21 = floor(length(Outer_21)/100);
Number_Of_Data_Ball_21 = floor(length(Ball_21)/100);
Number_Of_Data_Inner_28 = floor(length(Inner_28)/100);
Number_Of_Data_Ball_28 = floor(length(Ball_28)/100);

Number_Of_Data = Number_Of_Data_Normal
+ Number_Of_Data_Inner_7 + Number_Of_Data_Outer_7
+ Number_Of_Data_Ball_7 + Number_Of_Data_Inner_14
+ Number_Of_Data_Outer_14 + Number_Of_Data_Ball_14
+ Number_Of_Data_Inner_21 + Number_Of_Data_Outer_21
+ Number_Of_Data_Ball_21 + Number_Of_Data_Inner_28
+ Number_Of_Data_Ball_28;
```

```
All_Data = cell(Number_Of_Data,1);
Labels = strings(Number_Of_Data,1);

temp_normal = [];
temp_inner_7 = [];
temp_outer_7 = [];
temp_ball_7 = [];
temp_inner_14 = [];
temp_outer_14 = [];
temp_ball_14 = [];
temp_inner_21 = [];
temp_outer_21 = [];
temp_ball_21 = [];
temp_inner_28 = [];
temp_ball_28 = [];

for i = 1:length(Normal)

    temp_normal = [temp_normal;Normal(i)];

    if rem(i,100) == 0
        All_Data{i/100,1} = temp_normal;
        Labels(i/100) = "Normal";
        temp_normal = [];
    end

end

for i = 1:length(Inner_7)

    temp_inner_7 = [temp_inner_7;Inner_7(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +i/100,1} = temp_inner_7;
        Labels(Number_Of_Data_Normal
        +i/100) = "Inner";
        temp_inner_7 = [];
```

```
    end

end

for i = 1:length(Outer_7)

    temp_outer_7 = [temp_outer_7;Outer_7(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+i/100,1} = temp_outer_7;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+i/100) = "Outer";
        temp_outer_7 = [];
    end

end

for i = 1:length(Ball_7)

    temp_ball_7 = [temp_ball_7;Ball_7(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +i/100,1} = temp_ball_7;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +i/100) = "Ball";
        temp_ball_7 = [];
    end

end

for i = 1:length(Inner_14)

    temp_inner_14 = [temp_inner_14;Inner_14(i)];

    if rem(i,100) == 0
```

```
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+i/100,1} = temp_inner_14;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+i/100) = "Inner";
        temp_inner_14 = [];
    end

end

for i = 1:length(Outer_14)

    temp_outer_14 = [temp_outer_14;Outer_14(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +i/100,1} = temp_outer_14;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +i/100) = "Outer";
        temp_outer_14 = [];
    end

end

for i = 1:length(Ball_14)

    temp_ball_14 = [temp_ball_14;Ball_14(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+i/100,1} = temp_ball_14;
        Labels(Number_Of_Data_Normal
```

```
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+i/100) = "Ball";
        temp_ball_14 = [];
    end

end

for i = 1:length(Inner_21)

    temp_inner_21 = [temp_inner_21;Inner_21(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+Number_Of_Data_Ball_14
        +i/100,1} = temp_inner_21;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+Number_Of_Data_Ball_14
        +i/100) = "Inner";
        temp_inner_21 = [];
    end

end

for i = 1:length(Outer_21)

    temp_outer_21 = [temp_outer_21;Outer_21(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+Number_Of_Data_Ball_14
        +Number_Of_Data_Inner_21+i/100,1} = temp_outer_21;
        Labels(Number_Of_Data_Normal
```

```
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+Number_Of_Data_Ball_14
        +Number_Of_Data_Inner_21+i/100) = "Outer";
        temp_outer_21 = [];
    end

end

for i = 1:length(Ball_21)

    temp_ball_21 = [temp_ball_21;Ball_21(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+Number_Of_Data_Ball_14
        +Number_Of_Data_Inner_21+Number_Of_Data_Outer_21
        +i/100,1} = temp_ball_21;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+Number_Of_Data_Ball_14
        +Number_Of_Data_Inner_21+Number_Of_Data_Outer_21
        +i/100) = "Ball";
        temp_ball_21 = [];
    end

end

for i = 1:length(Inner_28)

    temp_inner_28 = [temp_inner_28;Inner_28(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
```

```
        +Number_Of_Data_Outer_14+Number_Of_Data_Ball_14
        +Number_Of_Data_Inner_21+Number_Of_Data_Outer_21
        +Number_Of_Data_Ball_21+i/100,1} = temp_inner_28;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+Number_Of_Data_Ball_14
        +Number_Of_Data_Inner_21+Number_Of_Data_Outer_21
        +Number_Of_Data_Ball_21+i/100) = "Inner";
        temp_inner_28 = [];
    end

end

for i = 1:length(Ball_28)

    temp_ball_28 = [temp_ball_28;Ball_28(i)];

    if rem(i,100) == 0
        All_Data{Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+Number_Of_Data_Ball_14
        +Number_Of_Data_Inner_21+Number_Of_Data_Outer_21
        +Number_Of_Data_Ball_21+Number_Of_Data_Inner_28
        +i/100,1} = temp_ball_28;
        Labels(Number_Of_Data_Normal
        +Number_Of_Data_Inner_7+Number_Of_Data_Outer_7
        +Number_Of_Data_Ball_7+Number_Of_Data_Inner_14
        +Number_Of_Data_Outer_14+Number_Of_Data_Ball_14
        +Number_Of_Data_Inner_21+Number_Of_Data_Outer_21
        +Number_Of_Data_Ball_21+Number_Of_Data_Inner_28
        +i/100) = "Ball";
        temp_ball_28 = [];
    end

end
```

Code for Partitioning Data into Train and Test Data

```
function varargout
= trainingPartitions(numObservations,splits)

arguments
    numObservations (1,1) {mustBePositive}
    splits {mustBeVector,mustBeInRange(splits,0,1,"exclusive")
    ,mustSumToOne}
end

numPartitions = numel(splits);
varargout = cell(1,numPartitions);

idx = randperm(numObservations);

idxEnd = 0;

for i = 1:numPartitions-1
    idxStart = idxEnd + 1;
    idxEnd = idxStart + floor(splits(i)*numObservations) - 1;

    varargout{i} = idx(idxStart:idxEnd);
end

% Last partition.
varargout{end} = idx(idxEnd+1:end);

end

function mustSumToOne(v)
% Validate that value sums to one.

if sum(v,"all") ~= 1
    error("Value must sum to one.")
end

end
```

BiLSTM Main Code

```
Labels_Category = categorical(Labels);
classNames = categories(Labels_Category);

numChannels = size(All_Data{1},2);
numObservations = numel(All_Data);
[idxTrain,idxTest]
= trainingPartitions(numObservations,[0.8 0.2]);
XTrain = All_Data(idxTrain);
TTrain = Labels_Category(idxTrain);


XTest = All_Data(idxTest);
TTest = Labels_Category(idxTest);

numHiddenUnits = 190;
numClasses = 4;


layers = [
    sequenceInputLayer(numChannels)
    bilstmLayer(numHiddenUnits,OutputMode="last")
    fullyConnectedLayer(numClasses)
    softmaxLayer];

options = trainingOptions("adam", ...
    MaxEpochs=280, ...
    InitialLearnRate=0.002, ...
    GradientThreshold=1, ...
    Plots="training-progress", ...
    Shuffle="never", ...
    Metrics="accuracy", ...
    Verbose=false);

net
= trainnet(XTrain,TTrain,layers,"crossentropy",options);
```

```
numObservationsTest = numel(XTest);
for i=1:numObservationsTest
    sequence = XTest{i};
    sequenceLengthsTest(i) = size(sequence,1);
end

[sequenceLengthsTest,idx] = sort(sequenceLengthsTest);
XTest = XTest(idx);
TTest = TTest(idx);

scores = minibatchpredict(net,XTest);
YTest = scores2label(scores,classNames);

acc = mean(YTest == TTest)

figure
confusionchart(TTest,YTest)
```