UNIVERSITY OF BIRMINGHAM

Deep learning for remaining useful life prediction in a remanufacturing system

by

NATHINEE THEINNOI

A thesis submitted to the University of Birmingham for the degree

DOCTOR OF PHILOSOPHY

Department of Mechanical Engineering

School of Engineering

College of Engineering and Physical Sciences

University of Birmingham

2023

UNIVERSITY OF
BIRMINGHAM

University of Birmingham Research Archive

e-theses repository

This unpublished thesis/dissertation is copyright of the author and/or third parties. The intellectual property rights of the author or third parties in respect of this work are as defined by The Copyright Designs and Patents Act 1988 or as modified by any successor legislation.

Any use made of information contained in this thesis/dissertation must be in accordance with that legislation and must be properly acknowledged. Further distribution or reproduction in any format is prohibited without the permission of the copyright holder.

For “the Wind Beneath My Wings”

Abstract

This thesis investigates Remaining Useful Life (RUL) prediction, which is central to Predictive Maintenance (PdM), as part of efforts to address uncertainties related to the return of products to a remanufacturing system. By accurately predicting the remaining useful life of critical components in a product, uncertainties associated with its return for remanufacturing are reduced. Remanufacturing planners can determine the optimal timing for product retrieval and establish well-structured schedules for subsequent processes.

The central focus of the research is utilising Deep Learning (DL) algorithms to predict the remaining useful life of components, a critical factor in determining the ideal timing for sending a product to the remanufacturing process. DL techniques, namely, Long Short-Term Memory (LSTM), Gated Recurrent Units (GRUs), Convolutional Neural Networks (CNNs) combined with LSTM, and CNNs combined with GRUs, have been selected for prediction of the RUL of bearings in rotating machinery. Additionally, a swarm-based optimisation tool, the Bees Algorithm (BA), and one its latest variants, the Two-Parameter Bees Algorithm (BA_2), have been introduced to optimise DL learnable parameters and tune DL hyperparameters, respectively.

The performance of these predictive models was evaluated through comprehensive benchmarking utilising prediction scores. Three studies were conducted. In the first study, the integrated CNN with LSTM model attained a maximum score of 0.49, with the possible values ranging from 0.0 to 1.0, where the optimal value is 1.0. However, in the second study, which offers an automated alternative to manual

hyperparameter tuning using BA_2, the best score achieved by the CNN-GRU model was 0.48.

The results of the final study, which incorporates the Adaptive Moment (ADAM) optimisation algorithm into the BA, demonstrate that the highest score obtained by the three models - LSTM, CNN with LSTM, and CNN with GRU - was 0.42. Thus, this study revealed that integrating BA with ADAM optimisation leads to reduced efficiency and inferior outcomes across all the models compared to previous studies. Nevertheless, the study underscores the superiority of combined DL models compared to individual DL models.

This thesis has shown that DL models can be developed for predicting the remaining useful life of critical components, such as bearings in rotating machinery, and that the best DL for the bearing RUL prediction problem is a combination of CNN and LSTM. The three studies conducted in this research have also demonstrated the possibility of the Bees Algorithm as a tool for optimising DL models.

Acknowledgements

First, I would like to acknowledge Chulachomklao Royal Military Academy through the Royal Thai Army, which supported my full scholarship to start my PhD in 2019. This funding provided me with the opportunity to work under the guidance of Professor Duc Truong Pham and to become a member of the Autonomous Remanufacturing Laboratory within the Mechanical Engineering Department at the University of Birmingham. Professor Pham's mentorship has directed my academic journey to achieve my goal, and I appreciate his patience and support.

Second, I thank Professor Athanasios Tsolakis, who facilitated my connection with Professor Duc Truong Pham and paved the way for me to become his student. I also thank all the Autoreman and Bees Algorithm Group members. Your unwavering support, assistance, and knowledge sharing have enriched my academic experience throughout this long and challenging journey. I would like to express my gratitude to my Thai friends who provided support during my stay in the UK, who offered companionship during both difficult and joyful times and who were motivating each other to overcome the various challenges of PhD life.

I would like to express special appreciation to Lieutenant Colonel Dr. Siwaphong Kusolpuchong, Colonel Asst.Prof. Dr. Arsit Boonyaprapasorn, Dr. Tinnakorn Kumsaen, Yamonporn Thummanusarn, Kitimoon Senee and Lieutenant Colonel Narong Phoomsuk, who was my special support team, helped me navigate through the critical phases of my study.

Finally, I would like to express my heartfelt gratitude to my family, Assoc. Prof. Dr. Vichai, Duangporn, and Assoc. Prof. Dr. Kampanart Theinnoi, whose unwavering support was the foundation of my success in life.

Table of Contents

List of Figures

List of Tables

List of abbreviations

ABC	Artificial Bee Colony Optimisation
ADAM	Adaptive Moment
AE	Autoencoder
AHP	Analytic Hierarchy Process
AI	Artificial Intelligence
ALO	Ant Lion Optimisation
ANN	Artificial Neural Network
ANOVA	Comprehensive Analysis Of Variance
BA	Bees Algorithm
BA-3+	Ternary Bees Algorithm
BA-BO-CNN	Bee Bayesian Convolutional Neural Network
BA_2	Two-Parameter Bees Algorithm
BAS	Beetle Antennae Search Algorithm
BAT	Bat Algorithm
BDA	Big Data Analysis
Bi-GRU	Bidirectional Gated Recurrent Unit
BiLSTM	Bidirectional LSTM
BO	Bayesian Optimisation
BP	Backpropagation
BPFI	Ball-Pass Frequency Inner race
BPFO	Ball-Pass Frequency Outer race
BPTT	Backpropagation Through Time
BSI	The British Standards Institution

CBM	Condition-Based Maintenance
CHI	Component Health Index
CM	Condition Monitoring
CNN	Convolutional Neural Network
CS	Cuckoo Search
DBM	Deep Boltzmann Machine
DBN	Deep Belief Network
DAE	Denoising Autoencoder
DCNN	Deep Convolutional Neural Network
DE	Differential Evolution
DL	Deep Learning
DNN	Deep Neural Network
DTL	Deep Transfer Learning
E-LSTM	LSTM with an Elastic net
ECG	Electrocardiogram
EMD	Empirical Mode Decomposition
EMD-CNN	Ensemble Deep Convolution Neural Network
EOL	End-of-Life
FA	Firefly Algorithm
FC	Fully Connected
FFNN	Feedforward Neural Network
FNN	Fuzzy Neural Network
GIS	Gas Insulated Switchgear
GPUs	Graphics Processing Units

GRU	Gated Recurrent Unit
GWO	Grey Wolf Optimisation
HSSA	Hybrid Sparrow Search Algorithm
IoT	Internet of Things
LSTM	Long Short-Term Memory
LSTM-AON	LSTM neural networks with attention-guided ordered neurons
LSTM-A	LSTM neural networks with weight amplification
MBA	Modified Bees Algorithm
MBA-3+	Modified Ternary Bees Algorithm
MHMS	Machine Health Monitoring System
ML	Machine Learning
MLP	Multilayer Perceptron
MSCNN	Multiscale Convolutional Neural Network
NB-GWO	New Balanced Grey Wolf Optimiser
ODC	Optimal Deep CNN classifier
PdM	Predictive Maintenance
PHM	Prognostics and Health Management
PM	Preventive Maintenance
PPC	Production Planning and Control
PSO	Particle Swarm Optimisation
RBM	Restricted Boltzmann Machine
ReLU	Rectified Linear Units
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network

RNN-HI	RNN-based Health Indicator
RUL	Remaining Useful Life
RVR	Relevance Vector Regression
SAE	Sparse Autoencoder
STAE	Stacked Autoencoder
SDA	Stacked Denoising Autoencoder
SGD	Stochastic Gradient Descent
SRU	Simple Recurrent Unit
SSA	Sparrow Search Algorithm
SSM	State-Space Method
SVD	Singular Value Decomposition
SVM	Support Vector Machine
SVR	Support Vector Regression

List of Symbols

$\%Er_i$	The percentage prediction error
A_i	The experimental accuracy scores
$ActRUL_i$	The RUL prediction from the proposed model
b_c	The biases of the cell candidates of the LSTM cell
b_f	The biases of the forget gate of the LSTM cell
b_h	The biases of the candidate hidden state of the GRU cell
b_i	The biases of the input gate of the LSTM cell
b_o	The biases of the output gate of the LSTM cell
b_r	The biases of the reset gate of the GRU cell
b_z	The biases of the update gate of the GRU cell
$\tilde{c}_t$	A vector of new candidate values of the LSTM cell
$c_{(t-1)}$	The previous cell value of the LSTM cell
c_t	The cell state of the LSTM cell
e	Number of elite sites in the Bees Algorithm
f_t	The forget gate of the LSTM cell
$\tilde{h}_t$	The current candidate state of a GRU cell
$h_{(t-1)}$	The previously hidden unit
h_t	The output of the updated cell state
i_t	The input gate of the LSTM cell
k	The patch's rank
L_{10}	Bearing life reliability laws
m	The height or width of the tensor

m	Number of best sites in the Bees Algorithm
MSD	The mean square deviation
n	The number of scout bees
n	The number of repetitions (y_i).
ngh_k	The unique magnitude of the exploration vicinity
nep	Number of worker bees on the elite patches
nsp	Number of bees recruited for ($m - e$) sites in the Bees Algorithm
o_t	The output gate of the LSTM cell
r	The number of channels of the dimensional tensor
r_t	The reset gate of the GRU cell
RUL_i	The actual RUL data
S/N Ratios	Signal-to-Noise Ratios
T	Triangular distribution of random number generator
T[a,b,c]	Triangular distribution with minimum limit a, likely value b and maximum limit c
tanh	Hyperbolic tangent
w_c	Weights of the cell candidate gate of the LSTM cell
w_f	The weights of the forget gate of the LSTM cell
w_h	The weights of the candidate hidden states of the GRU cells
w_i	The weights of the input gate of the LSTM cell
w_k	The number of worker bees in the k-th patch
w_{max}	Maximum worker bee count
w_{min}	The minimum worker bee count
w_o	The weights of the output gate of the LSTM cell

w_r	The weights of the reset gate of the GRU cell
w_z	The weights of the update gate of the GRU cell
x	Input variable
x_{max}	The maximum search bounds in the Bees Algorithm
x_{min}	The minimum search bounds in the Bees Algorithm
x_t	Current input
y_i	Target value of the result
z_t	The update gate of the GRU cell
σ	A sigmoid function

Chapter 1

Introduction

1.1 Background

In modern manufacturing landscapes, sustainability and resource efficiency are fundamental considerations. Remanufacturing is a crucial element of sustainable manufacturing, encompassing renovating previously utilised products to reinstate them to a state comparable to their original condition. This practice optimises the efficiency of resources and significantly reduces negative environmental consequences [1], [2]. However, an efficient remanufacturing process requires informed decision-making concerning the selection of products for refurbishment, which is contingent on understanding their condition and remaining lifespan.

The efficiency of the remanufacturing process is significantly influenced by the uncertainty surrounding the return of used products or 'cores'. The predictability of the cores returned for remanufacturing can sometimes be uncertain, with factors such as timing, quality, and quantity playing significant roles [1]-[6]. This uncertainty poses significant challenges to inventory management, production planning, and resource allocation within remanufacturing facilities.

To reduce these uncertainties, evaluating the Remaining Useful Life (RUL) of machinery (or the critical components therein) is crucial for remanufacturing, as it helps identifying the products that can be deactivated and the optimal time for remanufacturing [7]-[11]. Therefore, the precision of RUL evaluation holds significant

importance in the remanufacturing industry. Nevertheless, research gaps remain regarding the determination of the RUL of a product at any stage of its lifecycle [12].

RUL prediction based on the Predictive Maintenance (PdM) framework, a subset of the Prognostics and Health Management (PHM) Prognostics system, relies primarily on data analytics called condition monitoring (CM) data to ascertain the health status of machinery and predict its RUL [13], [14]. This technique focuses on monitoring equipment conditions using data-driven approaches that analyse extensive amounts of data, such as sensor data and historical records, to develop predictive models. These models are instrumental in guiding maintenance schedules and resource allocation in remanufacturing processes.

In a data-driven paradigm, Deep Learning (DL) techniques, a form of machine learning, are increasingly adopted in PHM to increase the accuracy of RUL predictions [15]-[18]. The advantages of these methods include the ability to analyse large and complex datasets and extracting unprocessed data without needing professional knowledge or manual processing [19]-[21]. As a result, DL techniques have evolved into potent and widely utilised tools in RUL prediction.

Recent advancements in DL have presented encouraging opportunities for forecasting the RUL of components in a product. This can be a crucial factor in reducing the uncertainty encountered in remanufacturing procedures. By accurately forecasting the RUL, remanufacturers can gain insights into when the product will enter the remanufacturing process, thereby enhancing the precision of planning and resource allocation.

This study investigated two individual and two combined DL techniques to predict the RUL of bearings in a rotating machine, contributing to the overall reliability

of machinery. However, using DL to predicting RUL is challenging, particularly given the complex network configurations [22]-[24], which significantly impacts the achievement of high prediction accuracy. Selecting appropriately hyperparameter values of a DL model is crucial, as any inaccuracies in this choice could lead to suboptimal predictive performance.

In this study, two DL network configuration methodologies, manual and automatic tuning, were implemented to adjust the hyperparameter settings. The former is an iterative trial-and-error approach, and the second is the Taguchi method. The study also entails using the Two-Parameter Bees Algorithm (BA_2), a metaheuristic algorithm founded upon a nature-inspired optimisation technique that mimics the foraging behaviour of honey bees. This automated tuning mechanism identifies the most fitting hyperparameters for DL algorithms. This functionality has the potential to streamline the hyperparameter selection process, leading to time savings and an overall enhancement in the performance of DL models [25].

Due to the many advantages of metaheuristic mechanisms [26], this research presents an innovative approach that integrates the Bees Algorithm (BA) and Adaptive Moment (ADAM) optimisation to refine DL learnable parameters—specifically, weights and biases—to enhance RUL prediction performance. This integrated methodology amalgamates the strengths inherent in each algorithm. The BA mitigates the risk of being trapped in local optima, thereby improving the efficiency of exploring the search space [26]. Concurrently, incorporating adaptive learning rates and momentum-based updates within the ADAM algorithm expedites the convergence process. This convergence is particularly well suited to addressing the requirements of

large-scale optimisation problems, such as training DL models, which frequently exhibit complex loss landscapes and many parameters [27].

1.2 Aim and objectives

This research aims to amalgamate PdM concepts with a remanufacturing system, thereby reducing the uncertainties associated with the return of products. This study focuses on implementing DL algorithms to forecast the RULs of components in a product, facilitating informed decision-making regarding the optimal timing for sending the product for remanufacturing. The main research questions addressed are: (i) what is the best DL model for RUL prediction? (ii) how can a metaheuristic optimisation technique be used to enhance the performance of a DL-based RUL prediction model?

The objectives formulated to attain this aim and answer the research questions are as follows:

1) To develop DL models for integration within a remanufacturing system.
2) To identify the health status of machine components and forecast RULs through DL techniques.
3) To integrate DL techniques with BA to optimise network configurations and enhance RUL prediction accuracy.
4) To assess the proposed DL approaches and compare them against existing RUL prediction methods.

1.3 Thesis Outline

The remainder of the thesis is structured as follows.

Chapter 2: Literature Review. This chapter examines a remanufacturing system and complicated features of production planning and control. The analysis indicates that

the most significant impact on the remanufacturing process is uncertainty in the timing and quantity of returned products. Prognostic methodologies rooted in maintenance concepts are viable solutions for addressing this concern. The RUL is an important piece of information in determining when a product should be withdrawn from service for remanufacturing. The potential of DL techniques to predict RULs for diverse purposes is explained within this context. To increase the precision of RUL prediction, swarm-based optimisation algorithms have been proposed for fine-tuning DL structures; these algorithms can be used to improve the performance of DL models. Finally, this chapter outlines the roadmap for integrating DL techniques into the remanufacturing process to achieve operational outcomes.

Chapter 3: Deep learning techniques for predicting remaining useful life. In this chapter, four DL techniques are employed - two individual algorithms - Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs) - and two combination algorithms - Convolutional Neural Networks (CNNs) with LSTM and CNNs with GRU. These techniques are applied to analyse data, discern patterns, classify bearing vibration signals, and predict RULs.

Chapter 4: Hyperparameter Configuration Using the Two-Parameter Bees Algorithm (BA_2). This chapter introduces the application of automatic configuration tuning through BA_2 to optimise a set of DL hyperparameters.

Chapter 5: Deep Learning Techniques with the Bees Algorithm for Remaining Useful Life Prediction. BA integration with Adaptive Moment (ADAM) optimisation is presented here. The integration aims to optimise the DL learnable parameters. The trajectory of the bees is directed toward specific regions within the search space, generating initial DL learnable parameters. These DL networks undergo training

optimised weights and biases with the ADAM optimiser. Including adaptive learning rates through the ADAM optimiser accelerates convergence toward the optimal solution.

Chapter 6: Conclusion and Future Work. This concluding chapter offers a summary of the study, delineates the contributions of the thesis, demonstrates its limitations, and outlines potential avenues for future research.

Chapter 2

Literature Review

2.1 Remanufacturing system overview

Along with the quality of production, environmental problems are considered by manufacturers. Reuse, recycling, and remanufacturing help to prevent environmental crises caused by the harmful disposal of industrial waste. In contrast to reuse and recycling, remanufacturing is the only method that can restore end-of-life (EOL) health to a good state and retain high value before reselling or returning to the production process [28], [29]. In addition, after reprocessing, remanufactured parts should last longer than recycled or reused components.

Remanufacturing is the process of restoring used products to new conditions and is eco-friendly because it tends to produce fewer greenhouse gases, contributing to reduce global warming [30]. Remanufacturing distinguishes itself from recycling and repairing within the realm of recoverable products based on Figure 2.1[28].

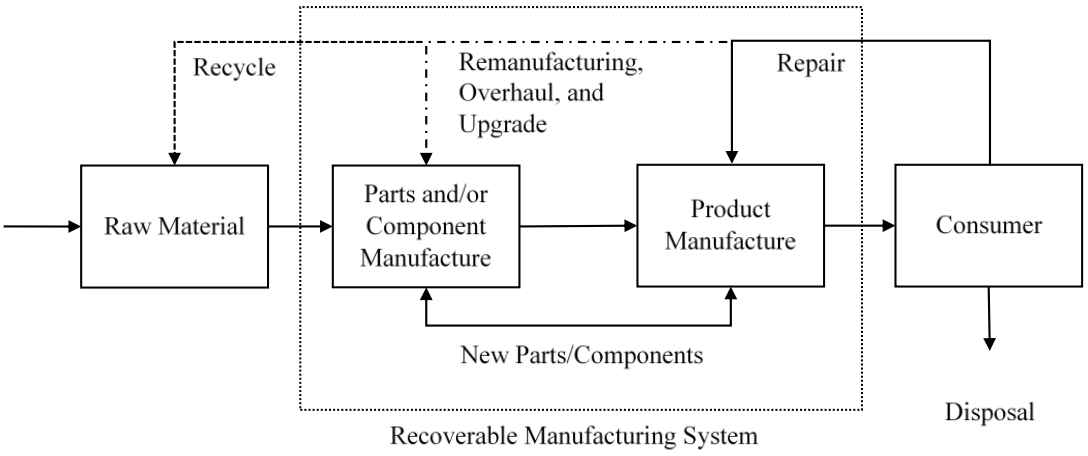


Figure 2.1 The recovery of product environments [28]

Remanufacturing consists of five steps, the disassembly, cleaning, inspection, reconditioning, and reassembly [31]; information flows is illustrated in Figure 2.2 [3].

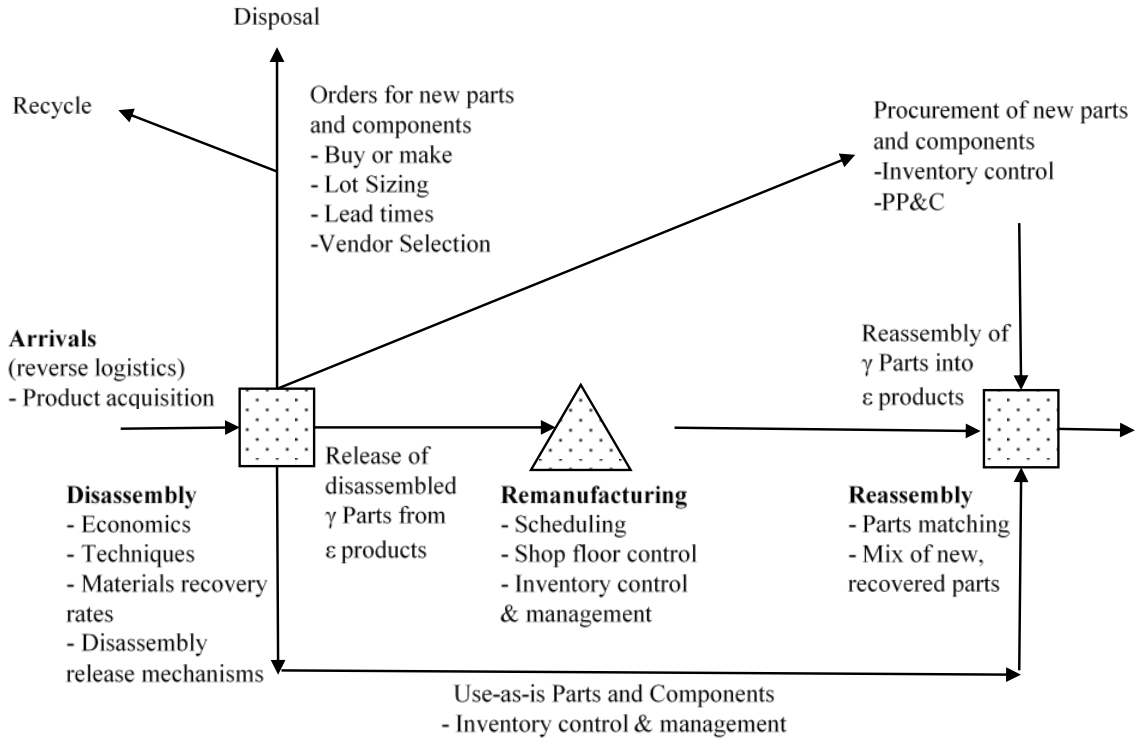


Figure 2.2 Flows of information in remanufacturing [3]

Based on Figure 2.2, researchers noted that these processes and information were incredibly complex and challenging for remanufacturers to manage due to uncertainty. These authors demonstrated seven complicating characteristics of production planning and control conditions, which are summarised in Table 2.1, and described the approaches used to solve these problems. As shown in Table 2.1, two of the seven complicating characteristics were caused by uncertainty: 1) the timing and quantity of the return and 2) the uncertainty of the materials recovered from the returned items. Compared to other characteristics, uncertainty related to the timing and quantity of returns had the greatest impact on all production planning and control activities.

Similarly, Sakao and Sundin [32] observed a high degree of variability, uncertainty, and complexity underlying the processes involved in remanufacturing. The uncertainty of the quantity, quality, and timing of the returned cores remained the main problem and made the disassembly process more complex. Several methodologies have been used to improve remanufacturing processes [3]. However, the optimal method to address these uncertainties, such as the timing or quality of product returns, has not yet been found.

Table 2.1 Literature review on the complicating characteristics of the activities of production planning and control [3]

Complicating characteristic	Production planning and control activity			
	Forecasting	Logistics	Scheduling/shop floor control	Inventory control and management
(1) The uncertain timing and quality of returns	✔	✔	✔	✔
(2) The need to balance returns with demands				✔
(3) The disassembly of returned products			✔	✔
(4) The uncertainty in materials recovered from returned items			✔	✔
(5) The requirement for a reverse logistics network		✔		✔
(6) The complication of materials matching restriction			✔	
(7) The problems of stochastic routings for materials and highly variable processing times			✔	

It is challenging to manage remanufacturing planning due to time and quality uncertainties since returned machine parts are out of a source control system, especially at the product life cycle stage [33], [34]. Bentaha *et al*. [35] observed that multiple factors, including age, conditions and usage patterns of returned parts, influence the disassembly step of remanufacturing. In addition, diverse products commonly exhibit different types of wear or failure. The remanufacturing industry uses procedures such as product condition evaluation, failure analysis, and product life prediction as the main methods of reducing uncertainty [9]. As a result, clarifying the condition and quality of

the returned products is needed to guide decisions in remanufacturing production planning and control [36].

2.2 Prognostics for the remanufacturing process

To achieve the remanufacturing goal, production planning and control (PPC) is needed. Predicting returned product disassembly is a critical data stream that can support production planning activities but has rarely been investigated [33]. The remanufacturing decision depended on three reliability relationships: the current time reliability, the predicted reliability, and the predetermined threshold reliability [8]. When the predetermined threshold reliability was less than the current threshold but greater than the prediction reliability, the optimal decision was to place the selected parts into the remanufacturing phase.

As part of the remanufacturing engineering process, the RUL is important for determining when a product should be taken out of service for remanufacturing [8]. RUL is the length of time from the point of normal condition to the end of the machine's useful life. Based on the Component Health Index (CHI), the RUL of a product was measured from the time it first deteriorated to the time when it reached its EOL [37]. A range of RULs is illustrated in Figure 2.3, and the relationships between the RUL and the prediction time (t_i) and EOL are shown in equation 2.1 [38].

$$RUL_{t_i} = EOL - t_i \, , \, t_i < EOL \qquad (2.1)$$

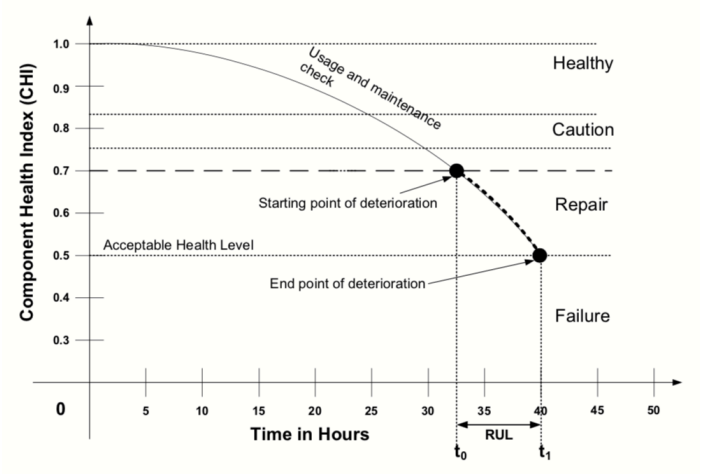


Figure 2.3 Relationships between CHI and time (hours) [37]

Based on the time-dependent degradation, the RUL was an effective indicator of the relationship between the remaining quality and recovery value [34]. The failure performance and life of operation parts are factors influencing remanufacturing planning [9]. As shown in Figure 2.4, there was a relationship between the RUL and remanufacturing and failure. The researchers identified three forms of failure—early, accidental, and exhausted—called the bathtub curve.

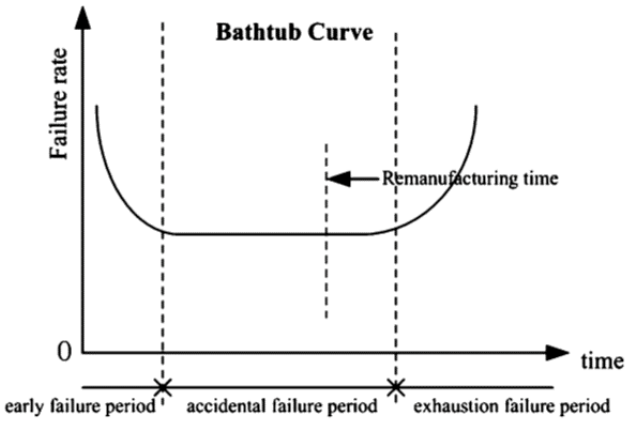


Figure 2.4 Failure period of a product [9]

The remanufacturing time point and RUL were presented in the second stage, called the accidental failure phase. The RUL started from the beginning of the constant

rate until it reached the remanufacturing time position at the end. In this step, defective products were routed for remanufacturing. After this point, the failure rate of the product increases, making remanufacturing difficult due to the EOL status of the components. As previously mentioned, the operation of remanufacturing depends on the RUL estimation. Therefore, knowing the RUL of products can help solve production planning problems due to the uncertain timing and quality problems of returned items. Operators can estimate when products will be returned and plan to allocate resources to support the remanufacturing process planning.

The idea of using RUL prediction for planning the remanufacturing process is followed by maintenance concepts. The key to maintenance techniques is to prevent unexpected failure or end of life in machines. Furthermore, replacing parts with the exact timeline will save additional costs for maintenance. For these reasons, performing maintenance work at the right time is essential to save costs and time due to unnecessary maintenance tasks.

The types of maintenance are classified into different groups. For example, Motaghare and Pillai [39] categorised three maintenance strategies: run-to-failure management, preventive maintenance and predictive maintenance for industry and factories. In the British Standards Institution (BSI) [40], the types of maintenance, as shown in Figure 2.5, were divided according to the causes of problems. To prevent machine breakdowns, degradation should be assessed and mitigated, and the likelihood of equipment failure should be reduced through preventive maintenance (PM). PM systems rely on predetermined maintenance tasks based on machine functionality or product lifespans. Therefore, tasks were scheduled during machine stops and shutdowns to replace components before they failed [41].

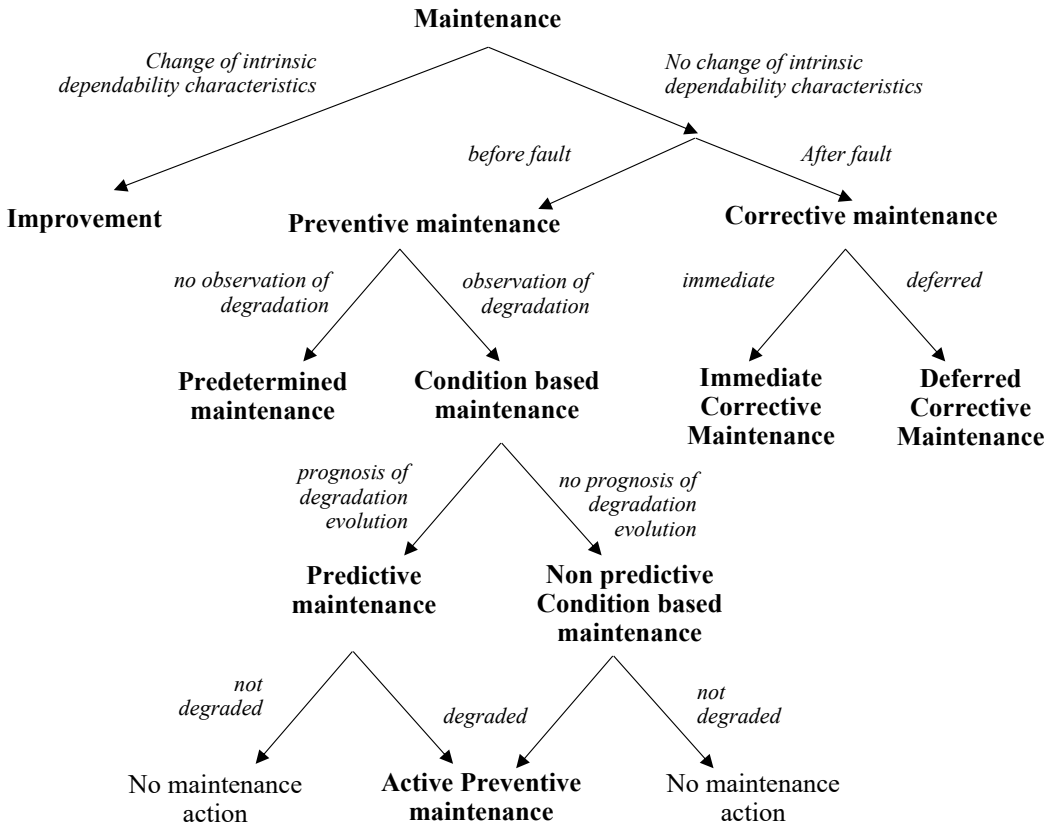


Figure 2.5 BSI standard maintenance types [40]

Preventive maintenance based on condition analysis and possible maintenance actions is known as Condition Based Maintenance (CBM). The CBM approach relies on information about equipment conditions to recommend appropriate maintenance procedures [42]. Various tools and techniques have been used in CBMs to detect deviations from normal operating conditions, diagnose impending failures, and predict future asset conditions. A CBM program generally has three main steps: data acquisition, data processing, and maintenance decision-making [42], [43]. As part of CBM, diagnostics and prognostics are crucial elements of maintenance decision-making related to RUL prediction[43]-[45].

Based on the conditions monitored and the diagnosis obtained, a prognosis has been used to estimate the RUL of a tool, machine, or system [46]. As previously

mentioned, machine prognostics are among the most critical keys to maintenance decision-making and are related to RUL prediction. The prognostic technique can be divided into three categories: physics-based, data-driven, and model-based, as illustrated in Figure 2.6 [47]. In physics-based models, failure modelling was created by integrating machine malfunctioning, fault processing, and CBM data for efficient prognoses. Although this technique used less data than data driven techniques, it was rather complicated for real-life modelling [48]. Additionally, Yan *et al*. [49] noted that complex systems cannot be utilised in physics-based methods due to human limitations in understanding these systems.

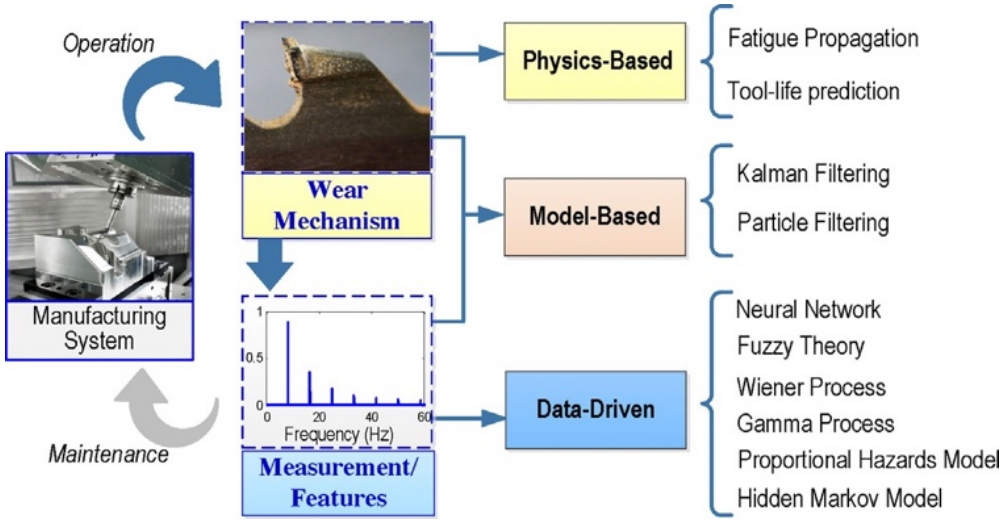


Figure 2.6 Prognosis methods [47]

Conversely, data-driven models do not require physical parameter assumptions but use massive amounts of data to create a prediction system [48]. In addition, these models rely on past information and the relationship between current machine failures and future health issues [47], [49]-[51]. Model-based prognostic strategies utilise a combination of physics-based and data-driven methodologies to enhance the accuracy

of predictions [47]. Furthermore, the new methods of integrating traditional physical methods with modern data-driven technologies are called hybrid methods [50],[52]. Hybrid approaches can capitalise on the accuracy and precision of physics-based models while utilising the flexibility and adaptability of data-driven techniques. However, hybrid approaches suffer from the amalgamation of weaknesses inherent in both methods. For instance, the complexity and computational requirements of physics-based modelling persist, augmenting a hybrid approach's overall complexity. Moreover, incorporating data-driven components may introduce challenges such as overfitting or the need for extensive data, which can potentially impact the performance and reliability of hybrid approaches. Consequently, hybrid approaches increase the complexity of finding solutions [52].

The Internet of Things (IoT) and Industry 4.0 have significantly impacted modern manufacturing. Sensors in manufacturing have provided a large amount of data to provide reliable information and improve production efficiency [53]. In Industry 4.0, a new technology based on data-driven models has recently been introduced for Predictive Maintenance (PdM) and predicting RULs [49]. Prognostics utilise trends and states from monitoring sensors to generate a model that predicts RULs [54]. Thus, it is possible to track the degradation of machine parts before machine failure [2].

An advancement over PM and CBM that involves condition monitoring is PdM. In contrast to PM [41], PdM was performed during machine operation. PdM aims to eliminate faults before instantaneous failure to ensure smooth operation and lower energy consumption. By detecting early signs of fault or failure, a PdM program can initiate maintenance procedures at the appropriate time [55]. Additionally, sensory information is used to monitor the status of a machine or equipment as part of condition

monitoring. Accordingly, PdM can use machine condition data to plan maintenance activities [56].

RUL prediction was implemented as a maintenance decision support tool. Examples demonstrating the relationship between RUL prediction and PdM are presented in [45], [57], [58]. Jimenez-Cortadi *et al.* [45] presented the development of PdM as a decision-making application that allows visual analysis of the exact RUL of the machining tool. Cao *et al.* [57] used RUL prediction results to establish a maintenance decision model. A multicomponent system PdM strategy was optimised by considering the average maintenance cost based on dynamic RUL predictions and a ranking of functional importance [58]. As mentioned above, the success of applying RUL prediction to a maintenance decision plan illustrates its importance.

However, engineers and researchers have continually improved the PdM technique to estimate faults and system deterioration over the lifetime of a system based on historical monitoring data, modelling, simulation, and failure probabilities [29]. Currently, the conditional predictive maintenance technique is one of the most widely used methods [59]. This method can display the status of the system in real time according to the parameters used. A new approach has been proposed using Industry 4.0 standards to facilitate the processes contained in the predictive maintenance technique.

Condition-based predictive maintenance solutions based on advanced data analytics were presented by Sakib and Wuest [29]. This article describes how PdM utilises data such as machinery efficiency, productivity, and RUL to identify and monitor maintenance problems using mathematical models. CBMs detect real-time signal irregularities caused by damaged machine parts via the development technique of PdM. As Wu *et al.*[56], a real-time condition monitoring system was used to

determine the appropriate replacement policy. According to these investigations, condition-based PdM can also prevent machine breakdowns before they occur by analysing data to predict machine anomalies.

As mentioned earlier, diagnostics and prognostics support maintenance decision-making. In general, prognostics determine how long a machine will last before it fails based on its current condition and previous operation history. This approach is typically called RUL [43]. A new framework for implementing a maintenance concept for remanufacturing was presented by Zhang *et al.* [14]. Their approach was to maintain the RULs of products in the high remanufacturability zone through a combination of maintenance concepts coupled with RUL prediction and condition monitoring techniques. In this case, remanufacturability was maintained at a high level, the remanufacturing costs were kept low, and the equipment value remained acceptable.

RUL represents not only deterioration indicators, which indicate when defective parts should be removed and sent for remanufacturing but also a point of correlation between quality and recovery time [34], [9]. Due to uncertainty, managing returned products for the remanufacturing is quite challenging for the industry [7]. RUL estimation is a concept for analysing product life before failure and assessing whether it is suitable for remanufacturing [9]. In addition, a remanufacturing plan was developed based on condition monitoring data, including RUL and reliability assessments. Accurate RUL predictions are important for determining when products or components are withdrawn from the process [10] and returned to remanufacturing.

2.3 RUL prediction methods

RUL prediction is a challenging aspect of engineering, mechanics, and automation [60] due to several causes of uncertainty. An efficient prognostic and health

management (PHM) system includes an RUL as a key functional component [61]. One of the features of a machinery prognostic program [62] or RUL indicates the approximate time until the machine fails [63]. The purpose of the PHM is to estimate the RUL based on condition monitoring data and maintenance decision-making [64].

As mentioned above, the three prognosis methods used were physics-based, data-driven, and model-based. Physics-based models and data-driven models were classified as RUL prediction models [7], [9], [31], [32]. Physics-based approaches, also known as model-based approaches, were preferred over data-driven approaches due to their accuracy and precision [52]. Moreover, Medjaher *et al.* [54] claimed that the prediction performance of data-driven approaches was not better than that of model-based approaches. However, constructing accurate physical models for predicting RUL in real-world scenarios is often difficult, particularly when the failure process is intricate compared to data-driven approaches [10], [34], [54], [66].

Due to the IoT and Industry 4.0, the rapid advancement of sensor and instrumentation technologies has made measuring and gathering condition-monitoring data effortless. Integrating real-time estimation with failure mechanics and dynamics will yield more precise and significant information regarding real-life scenarios. Therefore, a data-driven methodology has emerged as a comprehensive subject for predicting the RUL by training the model with lifecycle data that encompasses failure events and condition monitoring data [10], [34], [54], [66].

Over the past several years, prognostic data-driven methods have increasingly focused on establishing a connection between system data and corresponding RULs [50]. Data-driven models were generated by mapping sensor data and life degradation

against historical data [34], [46], [54], [65]. For these reasons, RUL estimation models could be built more accurately with data-driven approaches [65].

Moreover, data-driven methods created better prediction models than physics-based methods because they did not require complex mathematical models [46], [54] or analytic specialists [9], as demonstrated by their efficiency in estimating RULs in various fields [50], [63], [67]-[70]. In addition, data-driven prognostics have spread faster than physics-based prognostics in RUL prediction due to the rapid implementation and deployment of artificial intelligence (AI) tools [52], [71]. As a result, data-driven methodologies have been increasingly used to manage the amount of prognostic data used for RUL prediction and have attracted more attention than physics-based methods [63].

Lei *et al.* [62] divided RUL prediction approaches into four categories: 1) physics model-based approaches, 2) statistical model-based approaches, 3) Artificial Intelligence (AI) approaches, and 4) hybrid approaches based on basic techniques and methodologies. A pie chart summarises the RUL prediction approaches from the publications between 1997 and 2016 is shown in Figure 2.7.

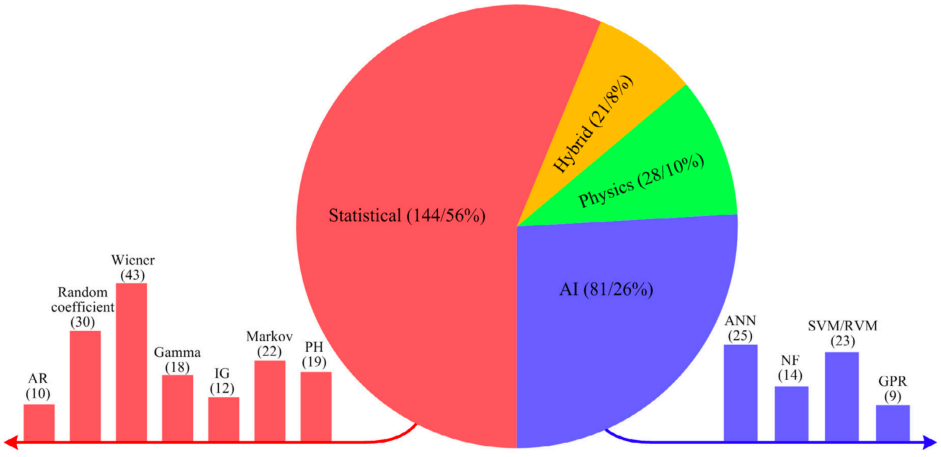


Figure 2.7 RUL prediction approaches based on published research [62]

According to Figure 2.7, a statistical model-based method represented the most significant proportion of publications because the model based on empirical knowledge and probabilities made these methodologies very reliable. Consequently, these approaches explain the uncertainties of deterioration and the effects of RUL estimation.

The second most frequently cited technique in the literature was AI (see figure 2.7), which has been used to solve complex prognostic problems where statistical or physical methods cannot be employed. For example, using AI methods made it possible to keep track of the degradation processes of complex mechanical systems that were difficult to relate to physics or statistical models.

The physical model-based method was rarely applied for RUL prediction, and hybrid methods even more rarely. In physics model-based methods, mathematical models were presented to explain machine failures. When mechanism damage and model parameters were considered, these methodologies were able to predict the RUL with high accuracy. However, this approach has been limited in some complex mechanical systems because of the difficulty in understanding the physics of damage.

Among the hybrid methods, the strengths of all the methods mentioned above were combined. Although these approaches were the least used, they may enhance RUL prediction in the future.

During the past few decades, online performance monitoring has become part of the RUL assessment for remanufacturing plans to maximise product life efficiency [8]. The establishment of RUL prediction models from online sensors has been explored through data-driven approaches based on AI [49]. When predicting RULs, data-driven methods are required to support a variety of performance monitoring data. In addition,

AI techniques can handle the prognosis of complex degradation problems, where the deterioration processes are challenging to link using physics or statistical models.

AI-based approaches can be classified into two categories: knowledge-driven approaches, which include expert systems and qualitative reasoning, and data-driven approaches, which include machine learning (ML) and neural networks [72]. Figure 2.8 depicts some AI methodologies employed in system health monitoring applications.

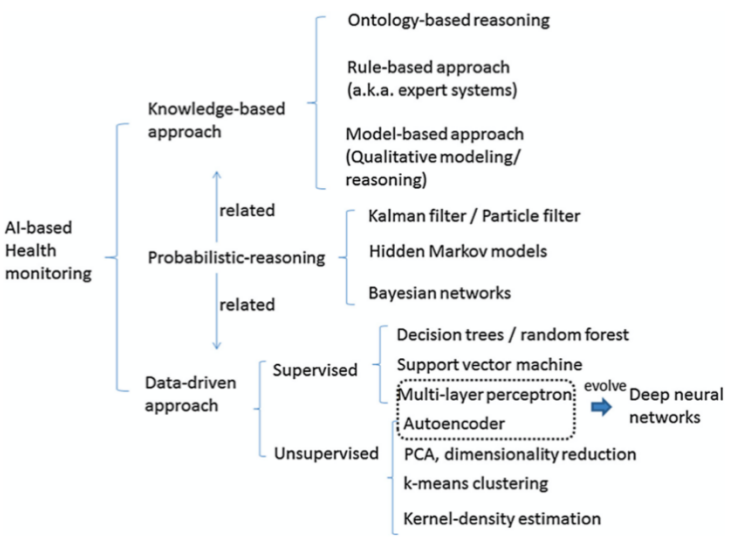


Figure 2.8 AI-based systems for monitoring system health [72]

As indicated in Figure 2.8, there are two types of data-driven methods: supervised and unsupervised [73]. Based on online performance monitoring, the RUL model was estimated using the Support Vector Machine (SVM) model and the State-Space Method (SSM) [8], [9]. Furthermore, RUL prediction was based on statistical and ML methods such as the Weibull model, Artificial Neural Network (ANN), or Fuzzy Neural Network (FNN), which provide good performance accuracy for remanufacturing [36].

The traditional ML cannot solve complicated problems [74]-[76]. However, due to the changing nature of big data and computing power, a new type of ML technique called deep learning (DL) plays an important role in RUL prediction, which was summarised in [72]. The relationships and examples of ML and DL related to AI-based approaches are shown in Figure 2.9 [77].

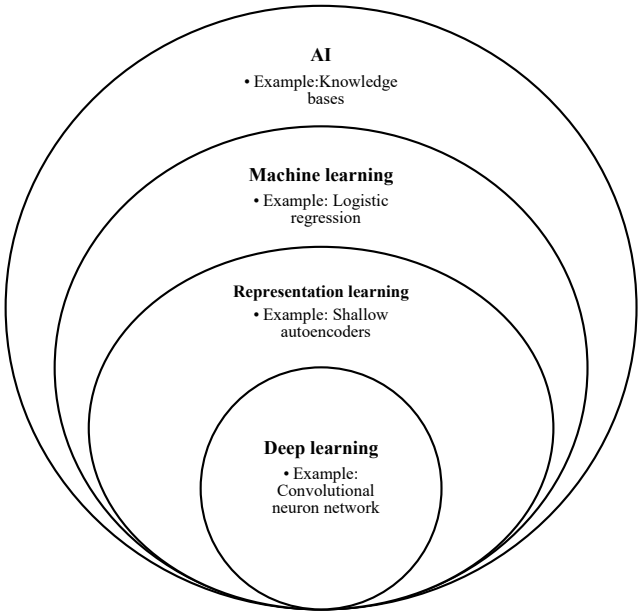


Figure 2.9 ML and DL related to AI-based approaches adapted from [77]

AI methods, particularly DL techniques, have gained popularity in recent years. The ability to analyse complex data has improved dramatically with the development of DL technology, and prediction issues using DL are becoming more common. DL networks probe a deeper level of data and have made considerable progress in various areas. Moreover, DL can perform better when predicting RULs for two main reasons. First, the advancement of big data analysis techniques has partially resolved the overfitting problem in training data. Second, the pre-training procedure prior to unsupervised learning assigns non-random initial values to the network. Consequently,

the training process can lead to a superior local minimum and a more rapid convergence rate [76].

Furthermore, DL can identify degradation patterns from a large amount of data without human associations or knowledge [72], [78]. In addition, DL possesses unique characteristics in feature learning, model design, and model training, setting it apart from traditional shallow machine learning [75]. Although Khan and Yairi [72] have discussed published studies related to DL in system health management and demonstrated that each technique has strengths and weaknesses, it is evident that DL has received more attention than ML in the context of RUL assessment and health monitoring.

2.4 DL techniques for RUL prediction

RUL prediction is divided into two parts. The former is the feature extraction methodology, whereas the latter is a prediction model [79]. First, feature extraction is the process of choosing or deriving relevant information (features) from raw data that can be utilised to construct predictive models. After that, the datasets are divided into two groups for training and testing. Finally, a set of models is selected and trained on training datasets, tested on test datasets, and prepared to predict the RUL.

According to Ren *et al.* [80], DL frameworks involve four phases: 1) data acquisition, 2) feature extraction and computing, 3) model training and 4) RUL prediction. Four types of DL, recurrent neural networks (RNNs), convolutional neural networks (CNNs), restricted Boltzmann machines (RBMs), and autoencoders (AEs), are successful architectures for machine health monitoring [15], [73], [75]. The strengths and weaknesses of each algorithm are summarised in Table 2.2 [75].

Table 2.2 Model comparison of DL algorithms [75]

Model	Principle	Pros.	Cons.
CNN	Abstracted features are learned by stacked convolutional and sampling layers.	Reduced parameter number, invariance of shift, scale and distortion	High computational complexity for high hierarchical model training
RBM	Hidden layer describes variable dependencies and connections between input or output layers as representative features.	Robust to ambiguous input and training label is not required in pre-training stage	Time-consuming for joint parameter optimization
AE	Unsupervised feature learning and data dimensionality reduction are achieved through encoding	Irrelevance in the input is eliminated, and meaningful information is preserved	Error propagation layer-by-layer and sparse representations are not guaranteed
RNN	Temporal pattern stored in the recurrent neuros connection and distributed hidden states for time-series data.	Short-term information is retained and temporal correlations are captured in sequence data.	Difficult to train the model and save the long-term dependence

CNNs are supervised learning algorithms. These structures are very effective at detecting features because they include both feature extractors and feature classifiers [81]. In addition, their architectures were able to extract features from complex data such as videos, images, speeches, and languages [15], [73], [81]. The other supervised learning feature is RNNs. The architectures include data series learning features [75], and the action depends on time [82]. As a subclass of RNNs, long short-term memory (LSTM) networks, can discover and create models from time series [77], [82]. Moreover, vanishing gradient problems or exploding gradient problems in RNNs have been solved with LSTM [72].

Both multiple RBMs and AEs are unsupervised methods. RBMs are stacked on each other to construct deep belief networks (DBNs) [73]. RBMs have the advantage of being able to produce samples that appear representative of the distribution of the input data. Even if data were missing, it is possible to create patterns [72]. AE consists of two components: an encoder and a decoder. The encoder function converts the input information into a new format, while the decoder function converts the new format back into the original representation [75], [77].

DL algorithms for RUL prediction have grown in popularity over the years. The success of DL models has been demonstrated in several studies. DL provides a robust

and effective solution in Machine Health Monitoring Systems (MHMS) [15]. Several types of neural networks can detect rolling bearing failure conditions, such as DBNs [83], deep Boltzmann machines (DBMs) [84], stacked denoising autoencoders (SDAs) [85] and CNNs [78], [86].

According to Shao *et al.* [83], three RBMs were stacked in DBN to analyse vibration signals. The advantages of this model include high performance in complex classification and high accuracy with efficient feature learning. The results confirmed that the model outperforms other intelligent defect identification techniques. The other variant of the RBM, DBM, which has the time, frequency, and time-frequency domains, was also the most effective model for analysing bearing fault conditions [84]. With SDA, it has become possible to achieve the benefits promised by robust feature representations based on deep architectures. SDA performed best at detecting anomalies when the signal contained considerable noise [85]. To confirm that DBM, DBN, and SDA were the best fault extraction methods, Chen *et al.* [87] tested whether each DL technique provided 99% fault classification accuracy. Moreover, in addition to DBM, DBN and SDA, CNNs have the ability to identify defects accurately even with lower signal preprocessing [86] and automatically create dominant features with minimal human input or expert knowledge [78].

A variety of DL models have been used for RUL prediction. RNNs and their variants, such as LSTMs, are well-known techniques for RUL estimation [88]. The prediction of bearing defects with long-term health status could be solved by RNNs, which provide highly accurate predictions [89]. Furthermore, the RNN-based health indicator (RNN-HI) successfully supported RUL prediction and achieved good performance [90]. An RNN variant called LSTM was one of the most widespread

techniques for RUL prediction. Compared to straightforward recurrent architectures, deep learning architectures can learn long-term dependencies more effectively [77]. Many case studies using LSTM for RUL were presented in [65], [70], [91]-[106].

The initial case study exemplifies the successful application of LSTM in predicting the RUL of Aero engines, as demonstrated by authors [70], [91]-[98]. Zhang *et al.* [70] and Zheng *et al.* [91] demonstrated how to track system degradation and predict remaining useful life from sensor data streams. Compared to support vector regression (SVR), deep CNNs and deep RNNs, LSTM networks outperform other commonly used ML techniques. Like [92], LSTM achieved better results than traditional statistical models such as SVR, relevance vector regression (RVR) or multilayer perceptron (MLP), and some DL models such as CNNs or RNNs. Moreover, an attention-based DL framework using LSTM was used to learn features from raw sensory data [93]. This experiment combined automatically learned features from LSTM with handcrafted features to improve the RUL prediction performance, and the method outperformed current approaches. LSTM was found to be suitable for diagnosing and predicting the engine performance of aircraft parts in complex operations, mixed faults, and strong noises [91], [94], [95], and it also produced high prediction accuracy from large-scale engine data [96].

Furthermore, LSTM networks were combined with data fusion techniques to estimate RULs [97]. Data from multiple sensors were analysed to determine short-range (local) and long-range (global) characteristics. The proposed LSTM-Fusion architecture method could accurately predict the RUL in both the raw and compressed datasets. Likewise, [98], a deep long short-term memory (DLSTM) model that combines

multisensor monitoring signals, produced an accurate prediction of RULs that used a DL structure to identify hidden long-term relationships between time series signals.

The second case is the RUL prediction of bearing [99]-[101]. In terms of predictability, LSTM had a lower root mean square error (RMSE) than did support vector regression (SVR) [99]. A novel approach to combining LSTM with an elastic net (E-LSTM) was presented by Liu *et al*. [100]. The elastic net-based regularisation term was added to the LSTM structure to solve the overfitting issue during training. For bearing RUL prediction, E-LSTM provided higher stability and more reliable results. According to the effect of external uncertainty quantification, an LSTM network with a combination of nonparametric kernel density estimation and dropout was used to predict the RUL [101]. The bearing RUL point estimate and probability distribution could be accurately determined with the suggested prediction model.

The potential of LSTM for predicting the RUL is illustrated by Zhou *et al*. [102] and Xiang *et al.* [103]. The RUL of supercapacitors and gear prediction [103], [104] was another example used to confirm the success of this method. In the study by Zhou *et al*. [102], a comparison was made between the experimental results and the error analysis of an RNN and a gated recurrent unit (GRU). This study showed that LSTM RNNs perform well in terms of accuracy and robustness. New types of LSTM neural networks with weight amplification (LSTMP-A) [103] and with attention-guided ordered neurons (LSTM-AON) [104] were suggested for predicting the remaining life of gears. The former technique can predict the health features of gears using historical fusion features. Compared to traditional LSTMs, the proposed method achieved a higher prediction accuracy based on monitoring data from a gear life cycle test. In the latter technique, health characteristic information is divided according to attention-ordered

neurons. Therefore, sequence information from neurons could be used to improve predictive performance. Since LSTM-AON relies on attention coefficient guidance, it is significantly more robust. Moreover, the proposed method achieved superior long-term prediction accuracy compared to existing methods such as GRU, LSTM, LSTMP and DLSTM.

As a robust DL structure for discovering time series variation patterns, a variant of LSTM, including "vanilla LSTM" [95] and Bi-directional LSTM (BiLSTM) [105], has been introduced for RUL prediction. BiLSTM is an advanced LSTM neural network that simultaneously processes input sequences in both forward and backward directions. This enables the network to access all the information before and after each time step in each sequence. Furthermore, BiLSTM networks were deployed to smooth tracking and prediction results to address uncertainties caused by operational and environmental factors [105]. After that, the variance in the health indices was tracked using the BiLSTM. In addition, this technique could take advantage of the bi-directional sensor data sequence [65]. As a result, the BiLSTM networks performed better than did SVR, deep convolutional neural network (DCNN), bidirectional RNNs, MLP, RVR, CNN and LSTM. Additionally, RUL prediction has improved by merging transfer learning with BiLSTM [106]. Small sample sets could benefit from transfer learning because they improve prediction models. Although transfer learning works well, it produces a worse result when switching from multiple types to single operating conditions.

CNNs are another technique for RUL prediction [50], [63], [75], [79], [107]-[110]. Initially, CNNs were first proposed for image processing, and several studies have been conducted on computer vision, speech processing, and other topics [50]. In addition, CNNs have been successfully applied in machine health monitoring systems,

particularly for fault detection [86], [111]-[117], before RUL prediction. Although CNN structures have shown excellent ability to extract features, the application of CNN structures for predicting the remaining useful life of machines has rarely been studied. Using CNNs for RUL prediction was the goal of the study presented by Babu *et al.* [107]. They claimed that the CNN-based regression approach was applied for the first time to estimate the RUL in prognostics, and that the potential was more significant than that of shallow regression. A CNN with a smoothing method was the first proposed method [79]. This method uses smoothing to address discontinuity in the predicted RUL of bearings. Specifically, the technique involves linearly smoothing the predicted RUL values using forward prediction data. By doing so, abrupt fluctuations in the predicted RUL values are mitigated, resulting in a more realistic representation of the bearing's degradation trend over time. Experimental results demonstrate that this approach significantly improves the accuracy of RUL prediction.

Moreover, prognostic results were stabilised by the smoothing technique [63]. Multiscale convolutional neural networks (MSCNNs) have been successfully used to extract multiscale features in the early prediction stages [108]. These features offer a comprehensive representation of the data across multiple scales, enabling information to be captured at various levels of detail and resolution. By incorporating a multiscale layer within the MSCNN, global and local features were preserved simultaneously, and the extracted features were salient and useful for RUL prediction. As described by Zhu *et al.* [108], compared with traditional CNNs, empirical mode decomposition (EMD) and ensemble deep convolution neural networks, which were introduced as EMD-CNN model structures, also maintain synchronisation of global and local information for more accurate predictions [110]. As shown in the literature [108] and [110], CNNs were

effective at performing feature extraction and could make accurate and efficient predictions.

The Restricted Boltzmann Machine (RBM) and Deep Belief Network (DBN) composed the DL model for prediction. Initially, the RBM was used to predict the RUL and its performance was compared with that of a particle filter-based approach. The results showed high errors in prediction [118]. Consequently, Deutsch and He [119] attempted to stack multiple RBMs, namely, DBNs, to integrate with feedforward neural networks (FFNNs). The new model slightly improved prediction accuracy but still performed poorly compared to particle filter-based approaches.

Compared to only DBNs, an integrated DBN with a particle filter-based method gave superior results [68]. As Deutsch *et al.* [68], Zhao *et al.* [120] combined a DBN with a relevance vector machine to achieve more accurate lithium-ion battery prediction results. [121] used multiple DL models, including an RBM and an LSTM network, to extract features and predict the RUL. It is evident that the RBM has the ability to extract useful information from features. Additionally, the RBM works with LSTM to reduce the amount of labelled training data and increase the accuracy of predictions.

AE is a specific neural network structure employed in deep learning to perform unsupervised learning tasks. It enables the discovery of meaningful representations from raw data without needing labelled examples. Moreover, it is particularly useful for dimensionality reduction, feature learning, and data denoising. AE is commonly used in signal extraction. The use of deep AEs reduces prediction parameters by automatically detecting and selecting features for signal extraction to avoid overfitting [80]. SDA with deep neural networks (DNNs) was applied in [122] to classify health stages. In terms of predictions, a variety of methodologies have been considered, including the denoising

autoencoder (DDA) with regression [49], DNN [80] and shallow artificial neural network (ANN) [122]. For transfer features, deep transfer learning (DTL) based on a sparse autoencoder (SAE) has also been developed. With limited failure data, DLTs were found to improve the performance of DL predictions and prove more efficient [123]. Several studies have proven that DL models are effective at predicting patient outcomes. It has been noted that DL methodologies are widely applied to RUL prediction and are able to handle large amounts of information and provide better predictions.

The development of new forecasting techniques is a major focus of many researchers. Currently, a new trend in RUL prediction has emerged to improve prediction accuracy. Two or more DL algorithms, for example, the stacked autoencoder (SAE), which works with LSTM [124]; AE, which is connected with BiLSTM [125]; or CNN, which combines with a variant of RNNs, such as a simple recurrent unit (SRU) network [126], LSTM [38], [127]-[139] or BiLSTM [120]-[121], have been implemented for these purposes. Moreover, three integration techniques, such as CNN with LSTM and DNN [142] or AE with CNN and LSTM [143], also performed better than one or two algorithms. Based on this trend, hybrid methods have proven themselves in all case studies because the models incorporate dominant performances and leverage the strengths of each technique.

2.5 Integrated DL techniques using swarm-based optimisation algorithms

Although DL is successful in RUL estimation, it is challenging to select, design, implement or improve performance of DL architectures due to the use of several hyperparameters that control the behaviour of the DL algorithm [77]. A good starting point or familiarity with hyperparameter values can easily be used to tune DL

hyperparameters. However, finding the best configuration within a short period of time is usually complicated. The optimal hyperparameter values can be determined by implementing hyperparameter optimisation techniques such as automatic optimisation, grid search, random search, or model-based optimisation [77]. Metaheuristic algorithms are commonly used in automatic hyperparameter selection and effective methods for solving optimisation problems. These algorithms have also been popularly used to optimise DL structures and parameters because they can find the optimal global solution through exploration and exploitation in the search space [144], [145]. The four main categories illustrate the different types of metaheuristics summarised by Kaveh and Mesgari [145], as shown in Figure 2.10.

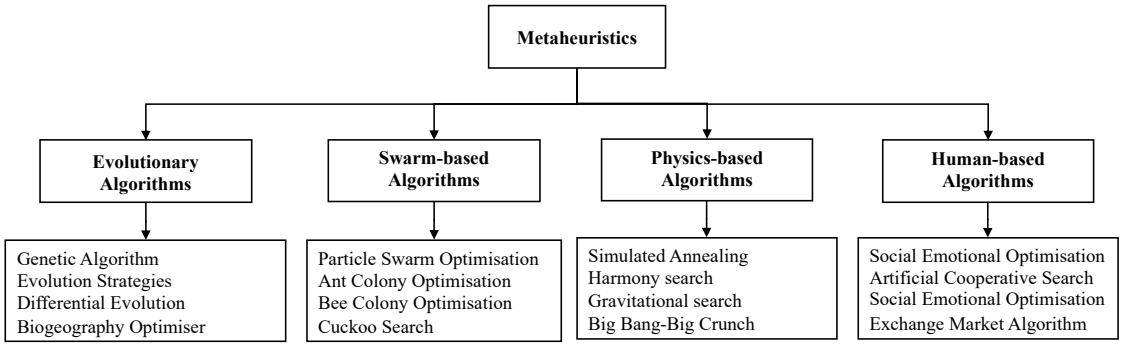


Figure 2.10 Algorithms based on metaheuristics [145]

Swarm-based programming was the most common publication between 2000 and 2013 [146]. In recent years, swarm-based methods have been studied and have attracted a growing number of researchers. These algorithms are the strategies, techniques, and algorithms designed to solve complex problems using bioinspired methodologies [146]-[148]. The method simulates the cooperative behaviour of some insects, birds, and fish, such as foraging ants, foraging bees, and birds flocking, to perform complex tasks such as finding food, organising nests, synchronising movements, or travelling at high speeds [148].

With the increasing complexity of neural networks, it is necessary to set additional hyperparameters. For example, the number of neurons or hidden units, learning rate, momentum, kernel size, and time window size were determined via hyperparameter optimisation using Particle Swarm Optimisation (PSO) [83], [86], [149]-[151]. Compared to genetic algorithms, PSO can reduce computational expenses and achieve efficient global optimisation by eliminating crossovers and mutations during the exploration process [150]. Moreover, it has proven effective in optimising nonlinear continuous functions [86], efficiently using computational resources and automating hyperparameter selection [149]. After obtaining the best hyperparameters, the fault diagnosis, classification and prediction performances are improved. It can be confirmed that PSO was an effective algorithm for automating parameter selection.

In addition to PSO, several algorithms have been adapted for hyperparameter optimisation, Ant Lion Optimisation (ALO), Bat Algorithm (BA), Grey Wolf Optimisation (GWO) and Sparrow Search Algorithm (SSA). For example, GWO identifies the optimal configuration of hyperparameters to create LSTM models that perform well when used for language modelling [144]. Additionally, the improved version of GWO, the New Balance Grey Wolf Optimiser (NB-GWO), was developed [152]—the latest version balances exploration and exploitation better than the original GWO. Based on the results of this study, it is clear that the NB-GWO has proven to be a very reliable and successful optimisation method for optimising DNN hyperparameters.

Lin *et al.* [153] utilised DL with SSA to predict dam deformation. In this study, SSA was considered a mechanism for selecting the optimal parameter set for the CNN-GRU model, thus thoroughly exploring the nonlinear mapping capabilities of both

CNNs and GRUs. Like Li *et al.* [154], SSA works with the bidirectional gated recurrent unit (Bi-GRU) to increase the accuracy of oil rate forecasts. The Bi-GRU model hyperparameters were tuned using the SSA to improve the model performance. In actual application, the hybrid model outperformed the current model in terms of accuracy, stability, and resilience.

The Hybrid Sparrow Search Algorithm (HSSA) combines the strengths of the SSA and PSO for an effective global search capability in complicated situations to determine hyperparameter optimisation within an acceptable time and avoid local optima [155]. However, large datasets, complex cases, and computational performance limitations remain for the HSSA. In addition, four metaheuristic algorithms, namely, the ALO algorithm, BA, ABC algorithm and PSO algorithm, were studied to optimise CNN hyperparameters [156]. This research showed that using metaheuristic algorithms for hyperparameter optimisation was an excellent alternative. Although ALO had the best result in this study, other metaheuristic architectures were interesting for improving hyperparameter optimisation.

A novel Bees Algorithm (BA), called the ternary BA (BA-3+), has been used as a training algorithm to find the best parameters for a sequential deep RNN language model to classify sentiment [157]. Unlike other iterative, metaheuristic algorithms, the BA-3+ algorithm uses only three candidate solutions during each training step, enabling rapid convergence through simultaneous local search, global search, and stochastic gradient descent (SGD) learning with singular value decomposition (SVD). Compared to the differential evolution (DE) and PSO algorithms, BA-3+ converged to the global minimum faster and outperformed SGD, DE, and PSO. In addition, Zeybek [158] developed a version of BA-3+, namely, the Modified Ternary Bees Algorithm (MBA-

3+), to improve learning capabilities and build an accurate prediction model. First, the learnable parameters are updated via gradient-based backpropagation for faster convergence in the local search operator. Next, the mutation operator and global search were employed to explore potential LSTM models with optimal hyperparameters. As a result, this approach minimised aircraft engine downtime and maximised equipment lifespan and achieved 98% accuracy under safe and unsafe conditions.

A combination of BA and Bayesian Optimisation (BO) has been proposed to improve CNN performance [159]. This hybrid algorithm is known as BA-BO-CNN. The authors aimed to fill the research gap by presenting a novel hybrid algorithm that took advantage of BA to train CNNs. BO was used to determine the best hyperparameters for the network, while a weight learning rate factor was optimised using BA to adjust the global learning rate generated by the BO. This study proved that the validation accuracy increased to 1.5%. Due to the success of this study [159], Alamri *et al.* [160] also designed BA to optimise BiLSTM parameters by adjusting the learning rate factors of LSTM cells. The weighted network was optimally updated to drive the optimal learning rate. This research examined three cases using the hybrid BA-CNN-BiLSTM and BA-LSTM algorithms. BA-CNN-BiLSTM was applied to estimate the porosity percentage of sequential layers of artificial porosity images and classify electrocardiogram (ECG) images, while BA-LSTM was used to predict the RUL of aircraft engines. The results demonstrated an increase in accuracy percentage from 93.17% to 95.17% for predicting porosity and from 92.50% to 95% for classifying ECG images using BA-CNN-BiLSTM. Moreover, the accuracy of RUL prediction for aircraft engines improved from 74% to 77% using BA-LSTM.

From all relevant publications, swarm-based optimisation algorithms have been widely implemented in DL to improve model performance, especially for optimal DL parameters or hyperparameters. However, applying these algorithms to train deep neural networks to boost DL performance remains challenging. Therefore, combining swarm-based algorithms and the most popular method of neural network optimisation, such as adaptive moment (ADAM), to train DL is an idea for filling a research gap in which a DL model is developed to avoid local minima and achieve faster convergence to the global minimum.

2.6 Integrated approach between DL techniques and the remanufacturing process

DL, or advanced machine learning, is a method for resolving various problems. It can learn the connections between the vast amounts of data from layers to layers, interpret information and evaluate feasibility solutions. These models tend to be very general, making them an extremely flexible technique [161]. Various DL methods have been used in several fields, including machine health prognostics, machine translation, audio recognition, automatic speech recognition, and language processing. However, despite their widespread application in other fields, DL techniques have rarely been used in RUL assessment for remanufacturing [162]-[164]. Consequently, limited studies on remanufacturing still exist for investigating DL techniques.

The integration of the DL technique and remanufacturing process examples are presented by Lin *et al.*[165], Zheng *et al.*[166], Schlüter *et al.* [167], Schlüter *et al.* [168], Nwankpa *et al.* [169], Nwankpa *et al.* [170], Wang *et al.* [171], Wang *et al.* [172], Pan and Miao [173]. Remanufacturing begins with damage detection, a necessary step in the decision-making process. Currently, damage detection is performed manually,

which is very time-consuming. To detect remanufacturing quality volatility, the feasibility and effectiveness of the BP multilayer perceptron neural network for detecting vibration signals quickly and accurately from a gearbox have been demonstrated [165]. Zheng *et al.* [166] developed a DL-based damage recognition and localisation method called the Mask-RCNN model. ML was integrated into the core selection to improve identification, inspection, and sorting to eliminate manual transcription errors [167]. In the study by Schlüter *et al.* [168], a CNN was developed for the sorting of replacement parts using vision-based identification. With CNNs, images and weights of used parts can be identified and categorised before being remanufactured, recycled, or disposed of. DCNNs have been applied to create automated sorting [169] and inspection [170] systems for remanufacturing. According to the results, the DCNN was significantly better than the existing sorting and inspection system design used in the remanufacturing industry in terms of accuracy, reliability, cost-effectiveness, and speed.

In addition to damage detection, other examples demonstrating the use of DL in remanufacturing are presented by Wang *et al.* [171], Wang *et al.* [172], Pan and Miao [173]. Wang *et al.* [171] investigated the relationship between the remanufacturing time and equipment cost. The DBN algorithm was developed to enhance the prediction accuracy of the optimal remanufacturing time. This research made it possible to use DL models to predict remanufacturing times, which is a fundamental factor in making decisions about remanufacturing. Additionally, due to the efficiency of DL techniques, modern remanufacturing systems will be supported by DL architectures. As presented by Wang *et al.* [172], DL models were introduced into remanufacturing big data analysis (BDA) procedures, specifically into the data mining and learning prediction model

stage, to address uncertainty problems in remanufacturing. In addition, a BP neural network technique was created to assess the risk of a remanufacturing supply chain. The accuracy of this work was greater than that of the analytic hierarchy process (AHP) [173].

As mentioned above, applying DL to remanufactured systems management has become one of the most challenging areas. According to a study by Moon *et al.* [174], determining replaceable parts and performing an economic valuation were the two most critical aspects of remanufacturing. Regression models based on machine learning algorithms were constructed to estimate the RUL for gas-insulated switchgear (GIS) remanufacturing decision-making. The process identified the most cost-effective and critical parts that needed to be replaced based on increasing the RUL value per cost.

From Moon *et al.* [174], it is clear that no research has been conducted on applying DL to upstream remanufacturing processes. EOL products will be analysed, and products in suitable conditions will be selected using DL techniques for continued remanufacturing. The procedure for disassembly is planned, and the return time is estimated. Therefore, remanufacturing planners can determine when the returned products will be ready for retrieval, and the returned product uncertainties are reduced based on the RUL prediction concept.

2.7 Summary

This chapter has reviewed and analysed the complexities of production planning and control in a remanufacturing system. The analysis has revealed that the most significant challenge in remanufacturing lies in the uncertainty surrounding the timing and quantity of returned products. To address this concern, prognostic methodologies rooted in maintenance concepts have been proposed. One crucial functional component

is the product's remaining useful life (RUL), which determines when a product should be withdrawn from service for remanufacturing. Within this context, the potential of deep learning (DL) techniques for predicting RULs has been discussed. Additionally, swarm-based optimisation algorithms have been introduced to fine-tune DL structures, enhancing overall prediction performance. Finally, a plan has been outlined for integrating DL techniques into the remanufacturing process, aiming to achieve operational outcomes.

Chapter 3

Use of deep learning techniques for predicting remaining useful life

3.1 Preliminaries

This research uses DL techniques to manage processes upstream of a remanufacturing system to address the issue of uncertainty in product returns. This chapter introduces DL techniques for predicting the RULs of critical components in a product before planning for subsequent processes. The prognosis will show when components start having problems and when they are likely to fail. At this point, an operation can be stopped, and a decision made as to whether parts or components should be repaired or replaced, or the product sent for remanufacturing before the end of life.

3.2 Case study

This section presents details of the selected case study, which is derived from the IEEE PHM 2012 Prognostic Challenge. The details of which are presented in the subsequent subsection.

3.2.1 IEEE PHM 2012 Prognostic Challenge

The training dataset for this study was obtained from the laboratory of the FEMTO-ST Institute, where an experimental platform called PRONOSTIA (Figure 3.1) was developed for the IEEE PHM 2012 Data Challenge [175], [176]. In the competition, the objective was to estimate the RUL under different loading conditions for bearings. PRONOSTIA represents a platform for testing and validating the detection, diagnosis, and prognosis of bearing failures. The main goal of PRONOSTIA is to obtain accurate information about the accelerated wear of bearings under online-controlled, constant

and/or variable operating conditions.

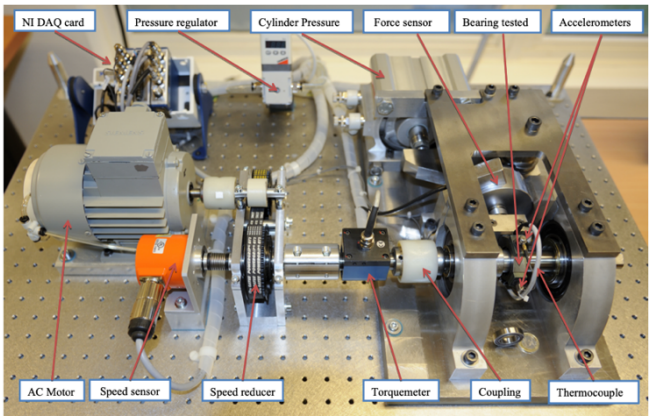


Figure 3.1 PRONOSTIA experimental platform [176]

The three main parts of the PRONOSTIA testbed are a rotating part, a deterioration generation part, and a degradation measurement part. Instantaneous measurements of the radial force applied to the bearing, the speed of rotation of the shaft handling the bearing, and the torque applied to the bearing are used to calculate the operating conditions. In addition, vibration and temperature, two data types from the sensors shown in Figure 3.2, are used to characterise the bearing's degradation.

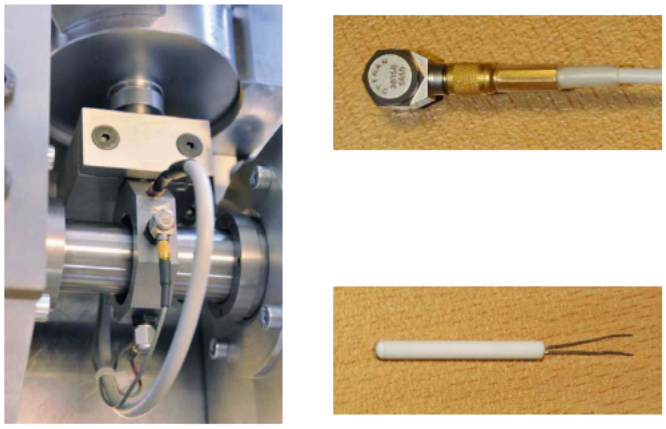


Figure 3.2 Two accelerometers and one temperature sensor [176]

To detect vibrations, two accelerometers are positioned radially on the outer race of the bearing and are arranged 90 degrees to each other; the first is on the vertical axis,

while the second is on the horizontal axis. The thermocouple measures the bearing temperature, and the probe is inserted into a hole near the ring of the outer bearing. The sampling frequency rates of this experiment for accelerations and temperatures are 25.6 kHz and 10 Hz, respectively [175], [176].

Since these bearings are the most likely components to fail in rotating machines, they have a massive impact on the availability, cost effectiveness, and safety of these industries. Therefore, bearings are considered critical because bearing failure can significantly disrupt machine availability and security. For these reasons, this dataset is an attractive case study for this PhD research. The prediction model is developed using the FEMTO bearing dataset to decide whether to send a machine for remanufacturing.

3.2.2 Dataset description and preparation

In this subsection, the datasets utilised in this study are described in detail, followed by the data preprocessing procedures. Additionally, the bearing health assessment process is examined before feeding into the prediction models. All details are illustrated in topics 3.2.2.1-3.2.2.3.

3.2.2.1 FEMTO bearing dataset

According to the IEEE PHM 2012 Prognostic Challenge [175], [176], 6 run-to-failure learning datasets and 11 truncated test bearings under three different load conditions, as shown in Tables 3.1 and 3.2, were prepared to construct prognostic models and evaluate the RUL. The data collection rate of vibration signals was 2560 data points per ten seconds, and all the data were recorded in files named “acc_xxxxx.csv”. For the rate of temperature signals, 600 data points per minute were collected and stored in files named “temp_xxxxx.csv”. The details of each data arrangement are illustrated in Table 3.3. Based on the experimental conditions, a run-

to-failure experiment was suspended once the vibration signal amplitude exceeded 20 g to avoid damaging the entire testbed, and the RUL was defined.

Table 3.1 IEEE 2012 PHM Prognostic dataset [175], [176]

Datasets	**Operating Conditions**		
	Conditions 1	Conditions 2	Conditions 3
Learning set	Bearing1_1 Bearing1_2	Bearing2_1 Bearing2_2	Bearing3_1 Bearing3_2
Test set	Bearing1_3 Bearing1_4 Bearing1_5 Bearing1_6 Bearing1_7	Bearing2_3 Bearing2_4 Bearing2_5 Bearing2_6 Bearing2_7	Bearing3_3

Table 3.2 Conditions of the bearing load [175], [176]

Operating Condition	Load	
	(rpm)	(N)
1	1800	4000
2	1650	4200
3	1500	5000

Table 3.3 Recorded file descriptions [175], [176]

Signal	Column					
	1	2	3	4	5	6
Vibration	Hour	Minute	Second	μ-second	Horizontal data	Vertical data
Temperature	Hour	Minute	Second	0.x second	Rtd sensor	

The first limitation of the FEMTO bearing dataset produced by the FEMTO-ST Institute is the small number of learning sets. Only two sets of learning data were used for each condition (Table 3.1). The second is the life range of all bearings. The PRONOSTIA experiments demonstrated very different behaviors of bearing degradation, resulting in very different durations of experimentation from the beginning of the investigation until the fault occurred. Thus, bearing life spans vary from one hour

to seven hours and are extensive. The experimental characteristics of the learning and test datasets are shown in Tables 3.4 and 3.5.

Table 3.4 Learning dataset characteristics [175]

Bearing1_1	- Experiment date: 2010-12-01 - Recording duration: 7h47m00s	- Nb. of files / channels: 3269 / 3 - Signals: vibration and temperature
Bearing1_2	- Experiment date: 2011-04-06 - Recording duration: 2h25m00s	- Nb. of files / channels: 1015 / 3 - Signals: vibration and temperature
Bearing2_1	- Experiment date: 2011-05-06 - Recording duration: 2h31m40s	- Nb. of files / channels: 1062 / 3 - Signals: vibration and temperature
Bearing2_2	- Experiment date: 2011-06-17 - Recording duration: 2h12m40s	- Nb. of files / channels: 797 / 2 - Signals: vibration
Bearing3_1	- Experiment date: 2011-04-07 - Recording duration: 1h25m40s	- Nb. of files / channels: 604 / 3 - Signals: vibration and temperature
Bearing3_2	- Experiment date: 2011-06-28 - Recording duration: 4h32m40s	- Nb. of files / channels: 1637 / 2 - Signals: vibration

Table 3.5 Test dataset characteristics [175]

Bearing1_3	- Experiment date: 2010-11-17 - Recording duration: 5h00m10s	- Nb. of files / channels: 1802 / 2 - Signals: vibration
Bearing1_4	- Experiment date: 2010-12-07 - Recording duration: 3h09m40s	- Nb. of files / channels: 1327 / 3 - Signals: vibration and temperature
Bearing1_5	- Experiment date: 2011-04-13 - Recording duration: 6h23m30s	- Nb. of files / channels: 2685 / 3 - Signals: vibration and temperature
Bearing1_6	- Experiment date: 2011-04-14 - Recording duration: 6h23m29s	- Nb. of files / channels: 2685 / 3 - Signals: vibration and temperature
Bearing1_7	- Experiment date: 2011-04-15 - Recording duration: 4h10m11s	- Nb. of files / channels: 1752 / 3 - Signals: vibration and temperature
Bearing2_3	- Experiment date: 2011-05-19 - Recording duration: 3h20m10s	- Nb. of files / channels: 1202 / 2 - Signals: vibration
Bearing2_4	- Experiment date: 2011-05-26 - Recording duration: 1h41m50s	- Nb. of files / channels: 713 / 3 - Signals: vibration and temperature
Bearing2_5	- Experiment date: 2011-05-27 - Recording duration: 5h33m30s	- Nb. of files / channels: 2337 / 3 - Signals: vibration and temperature
Bearing2_6	- Experiment date: 2011-06-07 - Recording duration: 1h35m10s	- Nb. of files / channels: 572 / 2 - Signals: vibration
Bearing2_7	- Experiment date: 2011-06-08 - Recording duration: 0h28m30s	- Nb. of files / channels: 200 / 2 - Signals: vibration
Bearing3_3	- Experiment date: 2011-04-08 - Recording duration: 0h58m30s	- Nb. of files / channels: 410 / 3 - Signals: vibration and temperature

Finally, there is a lack of information on bearing degradation. No further details on the components were available apart from the rotating system, and the load

and speed were constant, so only the sensors around the bearings provided data. In addition, no information was available regarding the nature of bearing degradations, such as balls, rings and cages, and no assumptions were made about what type of failure will occur. Furthermore, PRONOSTIA data did not support theoretical models for detecting bearing faults (inner or outer races, cage faults, etc.) based on frequency signatures, and bearing life reliability laws (L_{10})[a] did not reflect the results reported in the experiments. Therefore, theoretical frameworks such as the L_{10} law, ball pass frequency inner race (BPFI), and ball pass frequency outer race (BPFO) cannot be used to determine bearing life.

Owing to the limitations of the datasets, this research has great challenges in processing the data to create accurate estimates. In this study, only bearing vibration datasets were chosen for the RUL prediction experiments.

[a]L_{10} in the context of bearing life refers to the calculated fatigue life of a bearing, typically expressed in terms of the number of hours or revolutions under specified operating conditions before 10% of a group of identical bearings fail due to fatigue.

3.2.2.2 Data preprocessing

As mentioned in the earlier section, the insufficiency of training dataset samples for each condition necessitates the acquisition of additional training samples to train the model effectively. A sampling technique is used to generate additional training samples. The original vibration signals are uniformly sampled over time until the sequence is exhausted. Newly constructed sequences or samples maintain identical training sequence lengths within each condition, while the duration of data recording in the initial experiment remains constant. Conversely, the data preparation process for the test set yields subtle differences compared to that for the training data. As delineated in

MathWorks [177], a single sequence is required for model testing. Therefore, in generating data for model testing, the average of data points is utilised to depict the information within each time interval. This ensures that the sequence lengths of the dataset under each condition are approximate while maintaining consistent recording durations from the original experiments.

Regarding the dimensions of the training and test sets, the lengthy sequences require division into subsamples due to the optimal sequence length for LSTMs falling between 200-to-400-time steps [178], [179]. For this reason, the augmented dimensions of the training and test datasets, corresponding to each condition, are outlined in Tables 3.6 and 3.7.

Table 3.6 The augmented dimensions of the training dataset

Operating Conditions	Datasets	New training sizes
Condition 1	Bearing1_1	25600 samples x 281 timesteps
	Bearing1_2	7680 samples x 291 timesteps
Condition 2	Bearing2_1	5120 samples x 456 timesteps
	Bearing2_2	5120 samples x 399 timesteps
Condition 3	Bearing3_1	5120 samples x 258 timesteps
	Bearing3_2	15360 samples x 273 timesteps

Table 3.7 Dimensions of the test dataset

Operating Conditions	Datasets	New Test sizes
Condition 1	Bearing1_3	1 sample x 361 timesteps
	Bearing1_4	1 sample x 285 timesteps
	Bearing1_5	1 sample x 288 timesteps
	Bearing1_6	1 sample x 288 timesteps
	Bearing1_7	1 sample x 502 timesteps
Condition 2	Bearing2_3	1 sample x 1202 timesteps
	Bearing2_4	1 sample x 612 timesteps
	Bearing2_5	1 sample x 501 timesteps
	Bearing2_6	1 sample x 572 timesteps
	Bearing2_7	1 sample x 688 timesteps
Condition 3	Bearing3_3	1 sample x 352 timesteps

3.2.2.3 Bearing health assessment

As indicated in section 3.2.2.1, theoretical principles tailored toward identifying bearing defects do not apply to PRONOSTIA data, given the lack of labelled bearing fault information. To address this issue, this study utilised graphical plotting to investigate the health status of bearings. Figure 3.3 presents an illustration of this methodology for assessing bearing health.

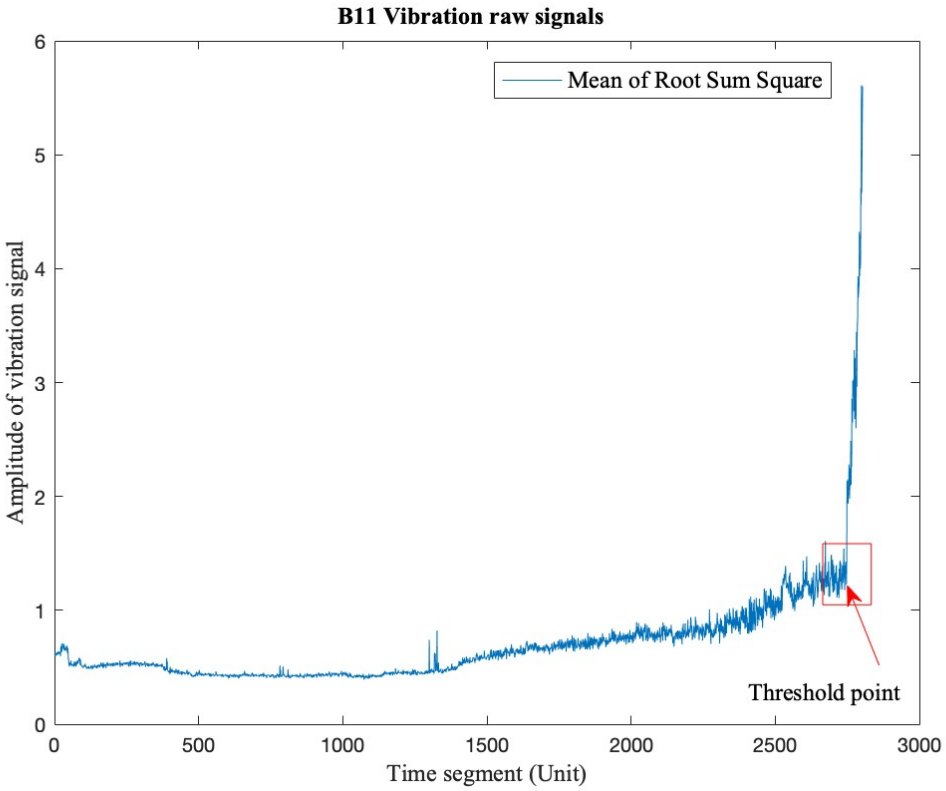


Figure 3.3 Bearing health assessment

The graph delineates the mean value of the root sum square of both raw vibration signals, as captured by horizontal and vertical accelerometers. The X-axis represents the variation in the number of data segmentation points recorded by two accelerometers, while the Y-axis represents the amplitude of the vibration signal. A sequence-to-sequence classification approach was used to distinguish between normal and abnormal vibration signals at sudden signal alterations or threshold points. At this point, the health status was appropriately labelled. Despite these challenges, this approach offers a feasible means of evaluating the health status of bearings without comprehensive data.

In contrast to the training data, the bearing health assessment for test data employs the actual RULs, as estimated in the IEEE PHM 2012 competition [175], to

analyse the health threshold. As discussed in the data preprocessing section, the dimensions of the dataset underwent alterations. These alterations exerted an impact on the threshold points. The reformed threshold was determined by applying a proportional relationship, termed the 'rule of three'. The relationship for the training data was derived from the labelled graphical plots, while the test data were based on the actual RULs, as documented in Table 3.8. The revised threshold points of the training and testing datasets for this study are presented in Tables 3.9 and 3.10, respectively.

Table 3.8 The actual RULs of the IEEE PHM 2012 competition [175]

Test set	Actual RUL (Sec.)
Bearing1_3	5730
Bearing1_4	339
Bearing1_5	1610
Bearing1_6	1460
Bearing1_7	7570
Bearing2_3	7530
Bearing2_4	1390
Bearing2_5	3090
Bearing2_6	1290
Bearing2_7	580
Bearing3_3	820

Table 3.9 Threshold points of the training dataset

Operating Conditions	Datasets	Threshold points (Time steps)
Condition 1	Bearing1_1	276
	Bearing1_2	271
Condition 2	Bearing2_1	434
	Bearing2_2	372
Condition 3	Bearing3_1	246
	Bearing3_2	226

Table 3.10 Threshold points of the test dataset

Operating Conditions	Datasets	Threshold points (Time steps)
Condition 1	Bearing1_3	246
	Bearing1_4	276
	Bearing1_5	268
	Bearing1_6	270
	Bearing1_7	248
Condition 2	Bearing2_3	449
	Bearing2_4	473
	Bearing2_5	423
	Bearing2_6	443
	Bearing2_7	456
Condition 3	Bearing3_3	270

3.3 Deep learning (DL) techniques

3.3.1 Convolutional Neural Networks (CNNs)

Convolutional neural networks (CNNs) are a type of neural network that process data based on a grid-like topology [180]. Typically, CNNs are employed for recognising or classifying two-dimensional (2D) grid data, such as images or videos; however, they can also extract time-series data represented as a one-dimensional (1D) grid. The CNN

architecture comprises three types of layers: convolutional, pooling and fully connected layers. By stacking these layers, the CNN architecture is formed, as depicted in Figure 3.4.

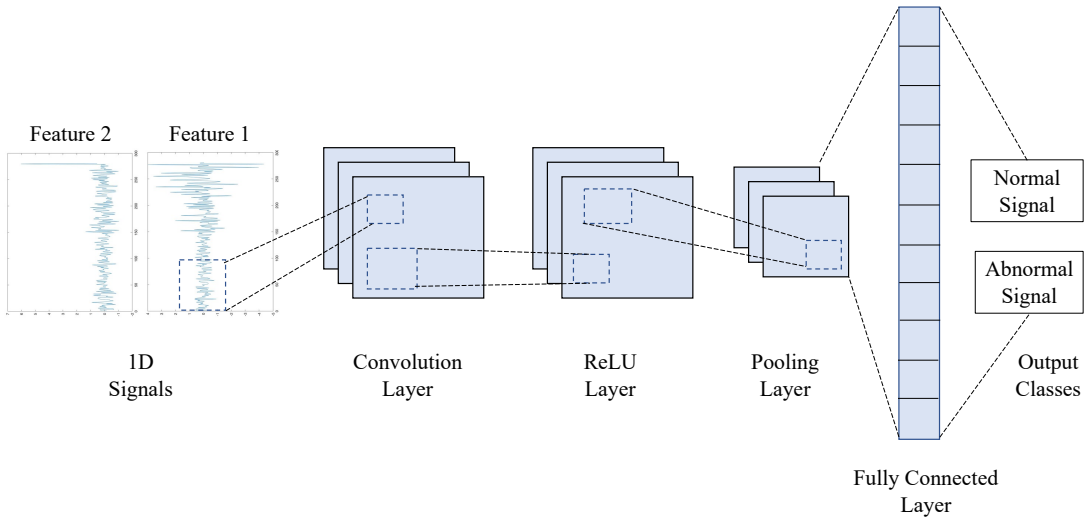


Figure 3.4 Example of a CNN architecture

In each layer of the CNNs, the input variable x is organised into a three-dimensional tensor comprising height, width, and depth. The tensor is defined as m × m × r, where m denotes the height and width, which are equal, and r is the number of channels, which is equivalent to the depth [19]. Feature extraction is accomplished by a convolution layer that typically integrates linear and nonlinear operations, such as the convolution operation and activation function [21]. The convolutional layer computes the outputs of neurons, each associated with local receptive fields in the input data. These connections represent sets of weights, and the output of each neuron is produced by calculating the dot of its weights with the local region of the input data.

In addition, convolutional layers can reduce model complexity by optimising the output through tuning three hyperparameters: the depth, the stride, and the zero-padding setting [181]. Typically, a convolution layer consists of five fundamental functions:

1) Convolution: A type of linear operation used for feature extraction. This operation involves transforming the input into a tensor by applying a small array of numbers known as a kernel or convolutional filter. In a kernel, discrete numbers or values are arranged in a grid where each value represents a kernel weight. The kernel weights are initially assigned random values during the CNN training and are subsequently adjusted during each training epoch. As a result, the kernel learns to identify meaningful features from the input data. To create a feature map, elementwise products are calculated between an input tensor and each kernel element. The output values, which correspond to the positions of the output tensor, are then summed. A convolution operation is illustrated in Figure 3.5.

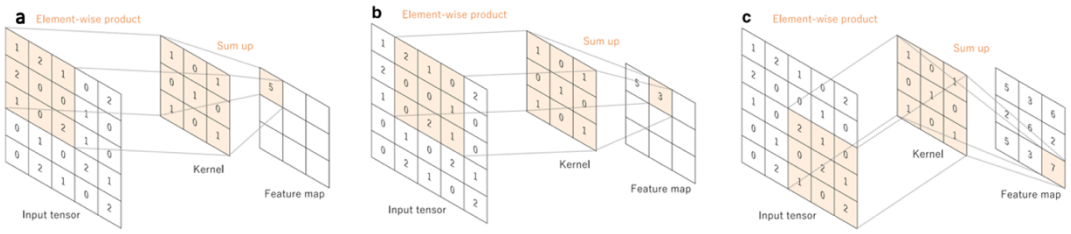


Figure 3.5 A convolution operation example [21]

2) Nonlinear activation function: This layer follows the convolution layer, which transforms the outputs of linear operations. The primary purpose of all activation functions in neural networks is to map inputs to outputs. Neurons become active through activation based on the input strength, determining whether the neuron will become active or fire. Outputs are then generated and transmitted to other neurons in the network. The activation functions most frequently used in CNNs and other deep neural networks are presented in Figure 3.6 and include the following:

- Logistic sigmoid: A sigmoid function is characterised by an S-shaped curve approaching 0 as x approaches negative infinity and 1 as x approaches positive infinity.

Therefore, the input values are converted into values between 0 and 1, and the output values are always within the range (0,1). The logistic sigmoid function is defined by Equation 3.1.

$$f(x) = \frac{1}{1+e^{-x}} \tag{3.1}$$

- Hyperbolic tangent (tanh): There is similarity between the tanh and sigmoid functions. However, the tanh function outperforms the logistic sigmoid, ranging between -1 and 1 instead of 0 and 1. The mathematical equation that describes the tanh function is presented in Equation 3.2.

$$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{3.2}$$

- Rectified linear units (ReLU): The most commonly used activation function in CNNs. The function takes an input value and returns either the input value or 0, depending on whether the input is greater than 0. This is mathematically represented by Equation 3.3.

$$f(x) = Max(0, x) \tag{3.3}$$

In recent years, the ReLU activation function has gained popularity over the sigmoid and tanh functions owing to several advantages. The most significant advantage of ReLU over its competitors is its lower computational load [19]. Additionally, the definitions of ReLU and its gradient are more straightforward, and it has a constant gradient for positive inputs. Although ReLU is not differentiable at zero, this issue can still be ignored during implementation [182]. However, the primary limitation of ReLU is that it cannot learn from cases where the activation is zero using gradient-based methods [180].

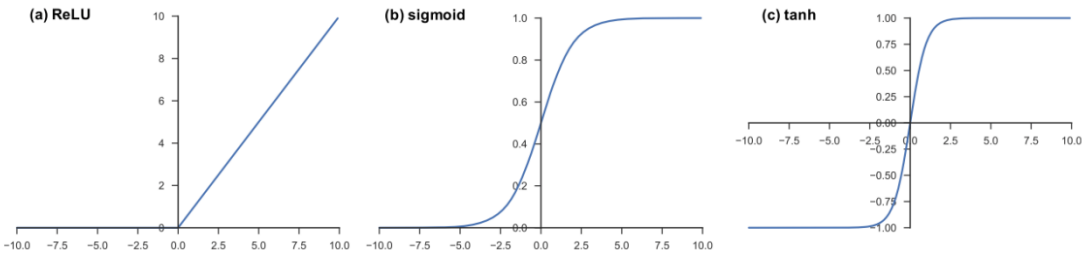


Figure 3.6 Nonlinear activation function types [21]

3) Stride: The separation between two consecutive kernel positions. Generally, one stride is selected. However, when the stride is set to a small number, large feature maps are generated due to extensive overlap in the receptive field. To address this issue, increasing the stride size can reduce the overlap and spatial dimensions in the feature map.

4) Padding: This is important for determining border sizes in the input images and addresses issues arising from the convolution step caused by information loss around the image border. A commonly used padding method is zero-padding, which involves the addition of rows and columns of zeros along the edge of each input tensor. This approach preserves the same in-plane dimension during convolution of the input tensor, ensuring that the kernel canter aligns with the outermost element. Consequently, the output volume dimensions are effectively controlled.

5) Weight sharing: Sharing weights enforce a constraint on using the same set of weights to extract features from different locations in the input image. This constraint results in the same weight values being assigned to each pixel of the input matrix, with no allocation of weights between neurons of neighbouring layers. By allowing the learning of a single set of weights for the entire input image, weight sharing can significantly reduce both training time and costs.

A subsequent layer to the convolutional layer is the pooling layer. The primary function of the pooling layer is to subsample the feature maps by creating a smaller feature map, which reduces the number of parameters and computational complexity of the model. The dominant information or features are maintained during the pooling process, while no learnable parameters, such as filter size, stride, or padding, exist in any pooling layer. Different pooling methods can be used in different pooling layers, including average pooling, maximum pooling, global average pooling, and global maximum pooling. Max pooling with a 2 × 2 filter size and a stride of 2 is the most commonly used pooling operation. This technique selects patches from the input feature maps, returns the highest value in each patch, and discards all the other values. An example of max pooling with a 2 × 2 filter size and a stride of 2 is depicted in Figure 3.7.

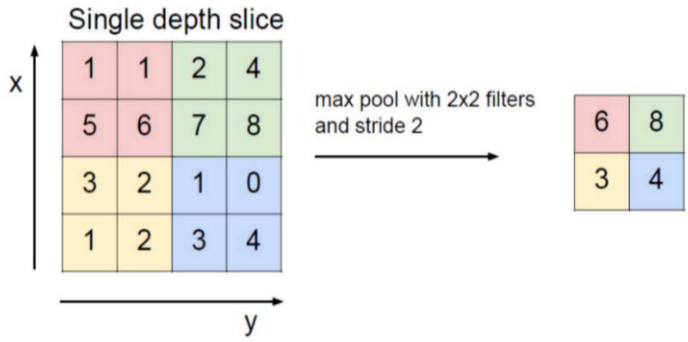


Figure 3.7 The example of max pooling [182]

The final layer of a CNN architecture is the fully connected layer or dense layer. The fully connected (FC) approach involves connecting every neuron of this layer to all neurons within the previous and subsequent layers. Upon completion of the last convolutional or pooling layer, the output feature maps are typically flattened, converted to vectors, and connected to the fully connected layer. Learnable weights are employed

to connect all the inputs and outputs of this layer. Subsequently, a subset of fully connected layers maps the input features to the final output of the CNN, which may consist of the classification probabilities for each class. Typically, the number of output nodes in the final fully connected layer corresponds to the number of classified classes.

3.3.2 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) was first proposed by Hochreiter and Schmidhuber in 1997 [183] as an advanced variant of the recurrent neural network (RNN) architecture. LSTM utilises a gradient-based learning algorithm to handle the issue of exploding or vanishing gradients that arise from imposing a constant error flow through the internal states of units. The algorithm can bridge time intervals exceeding 1000 steps, even in noisy and incompressible input sequences, without compromising its ability to identify short time lags.

The LSTM architecture comprises a sequence of recurrently interconnected subnetworks, commonly called memory blocks, memory cells, or simple cells. Fundamentally, the memory cell is designed to preserve its state temporally and to modulate information flow through nonlinear gating units, specifically the forget gate (f_t), the input gate (i_t), and the output gate (o_t). Each gating mechanism comprises a weighted matrix followed by an activation function layer that generates the output. Figure 3.8 provides a visual representation of a cell's interior components, demonstrating the connection between the input, gates, and output.

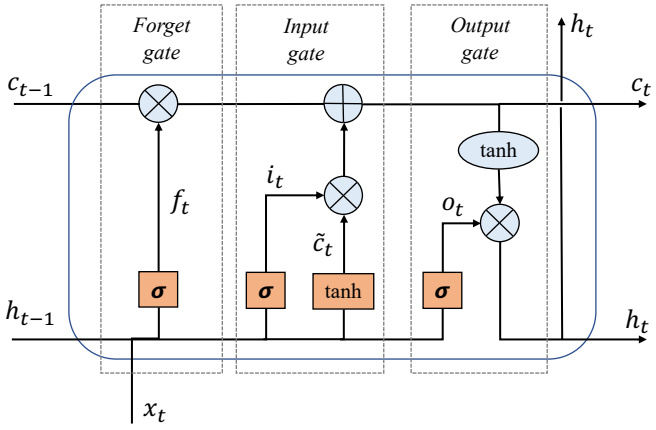


Figure 3.8 LSTM cell structure

The cell state (c_t), as depicted by the line traversing the top of the figure, functions as a conduit for transmitting information from a previously hidden unit to a subsequent unit. This transfer occurs with minimal linear interactions, allowing the information to remain largely unaltered [184]. The current input (x_t) is concatenated with the output from the previous hidden unit $\left(h_{(t-1)}\right)$ and passes through the forget gate. This gate modulates the degree of information retention or removal from the cell state by adjusting the self-loop weight [180]. This weight is determined through a sigmoid function (σ), yielding values ranging from 0 to 1. A value of 0 signifies "complete discard," whereas 1 indicates "full retention." The underlying mathematical relationship can be expressed as Equation 3.4.

$$f_t = \sigma\left(w_f[h_{t-1}, x_t] + b_f\right) \tag{3.4}$$

In the equation above, w_f and b_f represent the weights and biases of the forget gate, respectively.

Subsequently, the process involves a combination of two functions: the input gate and a hyperbolic tangent (tanh) layer [184]. Initially, the input gate, characterised by a sigmoid function, determines the values to be updated. Concurrently, a tanh

function generates a vector of new candidate values ($\tilde{c}_t$). Finally, utilising these two components, a new candidate value is selected, culminating in a state update. This can be expressed mathematically using Equations 3.5 and 3.6:

$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i) \tag{3.5}$$

$$\tilde{c}_t = \tanh(w_c[h_{t-1}, x_t] + b_c) \tag{3.6}$$

Here, w_i and w_c denote the weights, while b_i and b_c represent the biases of the input gate and the cell candidate, respectively.

The process of updating the previous cell value $\left(c_{(t-1)}\right)$ from the last iteration entails multiplying it by the forget gate value (f_t) and subsequently adding the output information from the input gate ($i_t * \tilde{c}_t$). Equation 3.7 represents the updated cell state (c_t) as follows:

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t \tag{3.7}$$

The output (h_t) is derived based on the updated cell state from the last iteration. The current input (x_t) and the output of the preceding hidden unit $\left(h_{(t-1)}\right)$ are processed through a sigmoid function to determine the output (o_t), as illustrated in Equation 3.8.

$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_0) \tag{3.8}$$

Note that w_o and b_o represent the weights and biases of the output gate, respectively.

Subsequently, a tanh function constrains the cell state value between -1 and 1, multiplied by the output of the output gate, as depicted in Equation 3.9.

$$h_t = o_t * \tanh(c_t) \tag{3.9}$$

Ultimately, only the selected segments are presented as the output.

3.3.3 Gated Recurrent Units (GRUs)

Gated Recurrent Units (GRUs), first introduced by Cho *et al*. in 2014 [185], represent an alternative variant of RNNs specifically developed to address the challenge of exploding and vanishing gradients, akin to LSTMs. Particularly well suited for applications in time series data and natural language processing, GRUs diverge from LSTMs in terms of their gating mechanism.

By featuring only two types of gates, update and reset gates, GRUs effectively control the flow of information within the network. Hence, GRUs require fewer computational resources and exhibit faster training times than LSTMs because of the reduced number of parameters. This enhanced efficiency renders GRUs suitable for scenarios in which data are scarce or insufficient [186]. The structure of the GRU cell is shown in Figure 3.9.

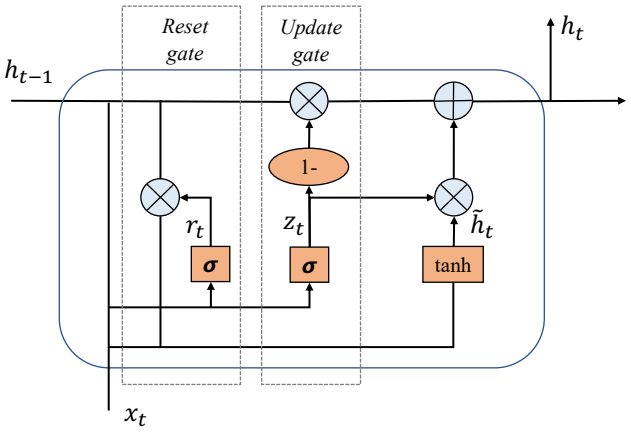


Figure 3.9 GRU cell structure

Like the forget gate and input gate of an LSTM, the update gate (z_t) in a GRU performs a comparable function. This gate modulates the degree to which the previous hidden state $\left(h_{(t-1)}\right)$ is carried over into the current hidden state (h_t). The update gate

enables the network to decide whether to retain or discard earlier information in favour of new input. The update gate is determined according to Equation 3.10.

$$z_t = \sigma(w_z[h_{t-1}, x_t] + b_z) \quad (3.10)$$

The reset gate (r_t) governs the extent to which the preceding hidden state influences the current candidate state $(\tilde{h}_t)$. The network employs the reset gate to discern whether to integrate the previous hidden state with the current input or disregard it entirely. The updated reset gate and candidate new value can be calculated using Equations 3.11 and 3.12.

$$r_t = \sigma(w_r[h_{t-1}, x_t] + b_r) \quad (3.11)$$

$$\tilde{h}_t = \tanh(w_h[r_t \odot h_{t-1}, x_t] + b_h) \quad (3.12)$$

Subsequently, the final output is updated by combining the update gate and the new candidate state value, as shown in Equation 3.13.

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t \quad (3.13)$$

In the equations, w_z, w_r, w_h and b_z, b_r, and b_h denote the weights and biases associated with the update gate, reset gate and candidate hidden state, respectively.

3.3.4 A combination of deep learning techniques

3.3.4.1 CNN with LSTM

As in the previous chapter, deep learning techniques possess distinct characteristics, each with unique abilities and strengths. Based on the advantages of each technique, combining various architectures allows for the utilisation of collective strengths, resulting in a more efficient model. The CNN-LSTM hybrid algorithm combines the robustness of CNNs in automatically learning and extracting dominant features from raw time series or sequence data with the strength of LSTM in interpreting

these features over time [179]. Furthermore, this hybrid approach can handle both spatial and temporal data, as CNNs extract local features from the input data, while LSTMs model the temporal dependencies among the extracted features. Consequently, a hybrid model can effectively manage data characterised by spatial and sequential patterns. For this study, a sequential 1D CNN-LSTM architecture was proposed, the structure of which is depicted in Figure 3.10.

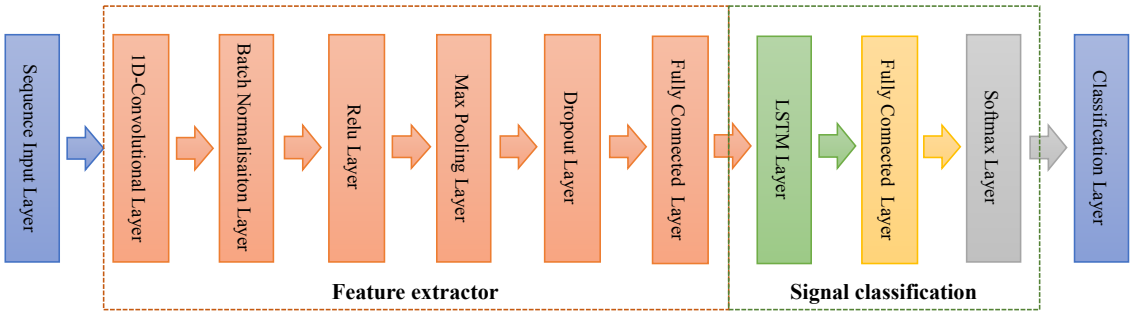


Figure 3.10 A structure of a CNN in combination with LSTM

In the section on feature extraction, the sequential model incorporates CNN layers along with two additional layers: batch normalisation and dropout. The batch normalisation layer functions by independently normalising small input batches, enhancing the training speed and diminishing the influence of initialisation parameters on the training process. Conversely, the dropout layer aids in preventing overfitting by reducing the number of neurons selected during the training phase.

After the extraction and pooling of spatial characteristics, the output is directed to the LSTM layer, which is utilised to model temporal dependencies in the classification section. The learning features of the CNN-LSTM model are combined and mapped to the desired output within the last fully connected layer, which is facilitated by an activation function. In addition, the softmax function is employed for multiclass

classification, and the hybrid models undergo training through backpropagation and a suitable loss function, such as categorical cross-entropy loss.

3.3.4.2 CNN with GRU

In reference to section 3.3.3. The GRU and LSTM networks serve as distinct variations of RNNs and were devised to solve the inherent challenges of RNNs. A key advantage of GRUs over long-term memory (LSTM) resides in their more simplistic gating mechanisms. Hence, compared with LSTMs, GRUs demonstrate faster training and inference times because of fewer parameters and fewer convoluted model architectures. This enables the integration of CNNs and GRUs by adhering to the hybrid structure in section 3.3.4.1. A sequential 1D CNN-GRU architecture is presented in Figure 3.11.

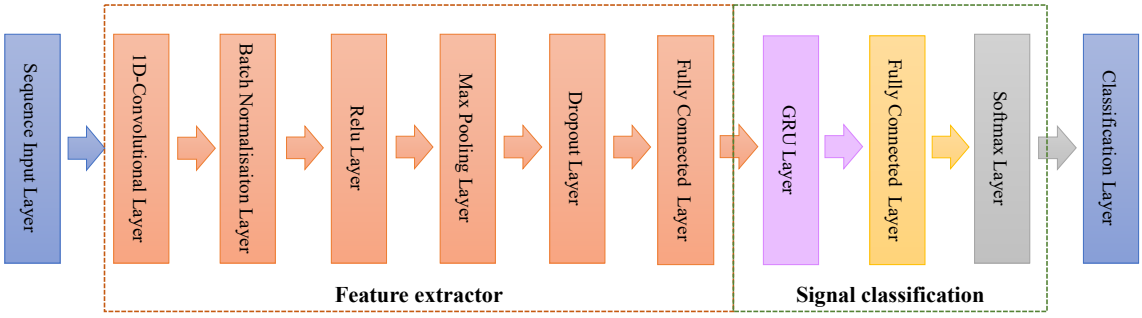


Figure 3.11 A structure of a CNN in combination with a GRU

3.4 Experimental design

3.4.1 RUL prediction framework

This section introduces the structural framework of the research experiment, as represented in Figure 3.12. The experimental design was structured to utilise vibration datasets exclusively for RUL prediction. The models were subjected to individual training for each dataset corresponding to various conditions.

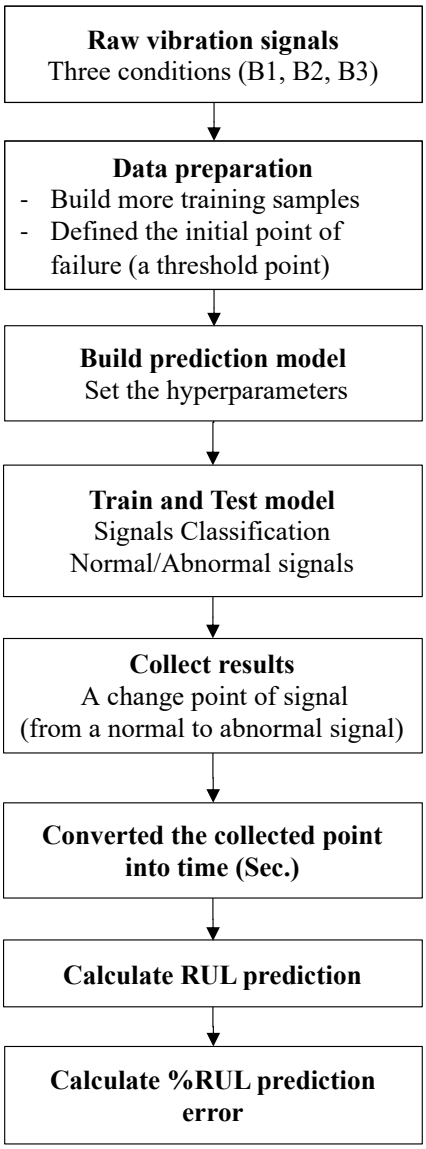


Figure 3.12 The structural framework

Before incorporating these datasets into the models, all datasets underwent thorough data preprocessing and were characterised by a health assessment, as mentioned in sections 3.2.2.2 and 3.2.2.3, respectively.

This study employed four prognostic models for RUL prediction: an LSTM, a GRU, a CNN with LSTM and a CNN with a GRU. The most crucial determinant of performance within these models is the configuration of hyperparameters [25], [187]-[190]. The method employed to determine the optimal set of hyperparameters is outlined

and explained in section 3.4.2. Following the establishment of the model, each DL model was trained using the deep learning toolbox provided by MATLAB.

In the last step of the experimental process, the model segments the vibration signal into two categories, normal and abnormal signals, based on the sequence-to-sequence classification concept [177]. The initial data point that transitioned from a normal to an abnormal signal was captured and designated the initiation point of the RUL period. The period from this point until the EOL was expressed on a time-step scale, as opposed to a clock time scale, necessitating a conversion process before calculating the RUL and percentage prediction error.

The percentage prediction error on test set *i* can be calculated by implementing Equation 3.14. In this equation, $ActRUL_i$ represents the RUL prediction from the proposed model, while RUL_i denotes the actual RUL data collected from the IEEE PHM 2012 challenge [175], as shown in Table 3.8.

$$\%Er_i = 100 \times \frac{ActRUL_i - RUL_i}{ActRUL_i} \tag{3.14}$$

Moreover, the prediction error percentages were weighted differently. The predictions yielded accurate estimates of the RULs of components at an early stage and resulted in positive errors ($\%Er_i > 0$) -termed early prediction-were regarded as indicative of proficient performance. In contrast, negative errors ($\%Er_i \leq 0$), or what is referred to as late predictions, warranted more considerable deductions. Consequently, the experimental accuracy score (A_i) can be determined utilising Equation 3.15.

$$A_i = \begin{cases} exp^{-ln(0.5).(\frac{Er_i}{5})} & if\ Er_i \leq 0 \\ exp^{+ln(0.5).(\frac{Er_i}{20})} & if\ Er_i > 0 \end{cases} \tag{3.15}$$

Within the scope of this research, eleven test sets encompassing three conditions were used to evaluate the model. Hence, the final score, the aggregate efficacy of the prediction model, was determined using Equation 3.16.

$$Score = \frac{1}{11}\sum_{i=1}^{11}(A_i) \tag{3.16}$$

3.4.2 Network configuration and tuning

3.4.2.1 Trials and errors

The foundational methodology for network configuration utilised in this research is an iterative trial and error approach. This method was applied to determine the best LSTM hyperparameters for condition three, the first experiment. The following five variables were considered: learning rates, number of hidden units, batch sizes, drop rate factors and drop periods. The assigned values for each variable are presented in Table 3.11.

Table 3.11 LSTM configuration of hyperparameters for trial and error

Hyperparameters	Assigned value
1. Learning rates	10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}
2. Drop rate factors	0.2, 0.5, 0.9, 0.95, 0.99, 1
3. Drop periods (%)	25, 50, 75, 80, 85, 90, 95, 100
4. Hidden units	10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200
5. Batch sizes	32, 64, 128, 256, 512, 1024, 2048

3.4.2.2 Taguchi method

Due to the extensive time required to examine combinations of all the hyperparameters, an alternative strategy, the Taguchi approach [191] was adopted to handle the remaining models and datasets. The initial phase identified the factors that

substantially influence the process. Five hyperparameters of single DLs, namely, learning rates, drop rate factors, drop periods, number of hidden units and batch sizes, were determined with four assigned values of each hyperparameter.

Six hyperparameters were identified and evaluated in the context of a combination of DL techniques. These hyperparameters include the number of filters, filter sizes, dropout, number of hidden units, learning rates, and batch size. Each hyperparameter was assigned five distinct values for analysis and experimentation. The configurations for the Taguchi experiments are outlined in Tables 3.12 and 3.13.

Table 3.12 Configuration of hyperparameters for a single DL model

Factors	Level 1	Level 2	Level 3	Level 4
(1) Learning rates	10^{-3}	10^{-4}	10^{-5}	10^{-6}
(2) Drop rate factors	0.90	0.95	0.99	1
(3) Drop periods (%)	30	60	90	100
(4) Hidden units	50	100	150	200
(5) Batch sizes	64	128	256	512

Table 3.13 Configuration of hyperparameters in combination with the DL model

Factors	Level 1	Level 2	Level 3	Level 4	Level 5
(1) Number of filters	32	64	128	256	512
(2) Filter sizes	3	5	7	9	11
(3) Dropout	0.1	0.3	0.5	0.9	0.99
(4) Hidden units	50	100	150	200	250
(5) Learning rates	0.0005	0.0001	0.005	0.001	0.01
(6) Batch sizes	32	64	128	256	512

Subsequently, an orthogonal array that depends on the number of factors and levels was selected to reduce the number of experiments. Based on the number of factors and levels in Tables 3.12 and 3.13, 16 experiments (L_{16}) and 25 experiments (L_{25}) were chosen, as illustrated in Tables 3.14 and 3.15, respectively.

Table 3.14 An orthogonal array for a single DL model hyperparameter (L_{16})

Experiment	Learning rates	Drop rate factors	Drop periods (%)	Hidden units	Batch sizes
1	10^{-3}	0.9	30	50	64
2	10^{-3}	0.95	60	100	128
3	10^{-3}	0.99	90	150	256
4	10^{-3}	1	100	200	512
5	10^{-4}	0.9	60	150	512
6	10^{-4}	0.95	30	200	256
7	10^{-4}	0.99	100	50	128
8	10^{-4}	1	90	100	64
9	10^{-5}	0.9	90	200	128
10	10^{-5}	0.95	100	150	64
11	10^{-5}	0.99	30	100	512
12	10^{-5}	1	60	50	256
13	10^{-6}	0.9	100	100	256
14	10^{-6}	0.95	90	50	512
15	10^{-6}	0.99	60	200	64
16	10^{-6}	1	30	150	128

Table 3.15 An orthogonal array for a combination of DL model hyperparameters (L_{25})

Experiment	Number of filters	Filter sizes	Dropouts	Hidden units	Learning rates	Batch sizes
1	32	3	0.1	50	0.0005	32
2	32	5	0.3	100	0.0001	64
3	32	7	0.5	150	0.005	128
4	32	9	0.9	200	0.001	256
5	32	11	0.99	250	0.01	512
6	64	3	0.3	150	0.001	512
7	64	5	0.5	200	0.01	32
8	64	7	0.9	250	0.0005	64
9	64	9	0.99	50	0.0001	128
10	64	11	0.1	100	0.005	256
11	128	3	0.5	250	0.0001	256
12	128	5	0.9	50	0.005	512
13	128	7	0.99	100	0.001	32
14	128	9	0.1	150	0.01	64
15	128	11	0.3	200	0.0005	128
16	256	3	0.9	100	0.01	128
17	256	5	0.99	150	0.0005	256
18	256	7	0.1	200	0.0001	512
19	256	9	0.3	250	0.005	32
20	256	11	0.5	50	0.001	64
21	512	3	0.99	200	0.005	64
22	512	5	0.1	250	0.001	128
23	512	7	0.3	50	0.01	256
24	512	9	0.5	100	0.0005	512
25	512	11	0.9	150	0.0001	32

The experiments started by configuring the hyperparameter provided by the orthogonal array. The models were trained, and the percentages of classification accuracy were collected. After completing the experiments, the mean, variance, and signal-to-noise ratios (S/N ratios) were calculated for each experimental result to evaluate the impact of each factor. The mean represents the average result, whereas the variance measures the degree of fluctuation in the results relative to the conditions.

The S/N ratio quantifies the influence of control factors on noise. The ratio is computed using Equation 3.17, where MSD symbolises the mean square deviation from the desired value of the quality characteristic [192].

$$S/N = -10 \log_{10}(MSD) \quad (3.17)$$

The formula for an MSD varies based on the experimental objectives of maximising, minimising or achieving a target of the experiment. These objectives are represented in Equations 3.18-3.20.

For "Larger is Better", $$MSD = \left(\frac{1}{y_1^2} + \frac{1}{y_2^2} + \cdots\right)/n \quad (3.18)$$

For "Smaller is better", $$MSD = (y_1^2 + y_2^2 + \cdots)/n \quad (3.19)$$

For "Normal is best", $$MSD = [(y_1 - y_0)^2 + (y_2 - y_0)^2 + \cdots]/n \quad (3.20)$$

In these equations, y_0 denotes the target value of the result, y_1, y_2, etc., represent the results of the experiments, and *n* is the number of repetitions (y_i).

The "larger is better" criterion was selected for the experimental process. A comprehensive analysis of variance (ANOVA) was conducted to determine the significance of the effect of each factor on all the results. Subsequently, the main effects of the mean and S/N ratio were plotted to visualise the interaction between each factor and the quality characteristic. The example results derived from the Taguchi method for LSTM model with dataset condition two using Minitab software are displayed in Figures 3.13-3.15.

Response Table for Signal to Noise Ratios

Larger is better

Level	Learning rates	Drop rate factors	Drop periods (%)	Hidden units	Batch sizes
1	39.91	39.94	39.95	39.95	39.95
2	39.91	39.94	39.96	39.95	39.95
3	39.98	39.95	39.95	39.93	39.94
4	39.97	39.94	39.91	39.94	39.94
Delta	0.07	0.01	0.04	0.02	0.01
Rank	1	5	2	3	4

Response Table for Means

Level	Learning rates	Drop rate factors	Drop periods (%)	Hidden units	Batch sizes
1	98.94	99.34	99.47	99.42	99.39
2	99.00	99.35	99.50	99.48	99.42
3	99.80	99.39	99.41	99.22	99.26
4	99.62	99.28	98.98	99.25	99.30
Delta	0.86	0.11	0.51	0.26	0.16
Rank	1	5	2	3	4

Figure 3.13 Taguchi response table results

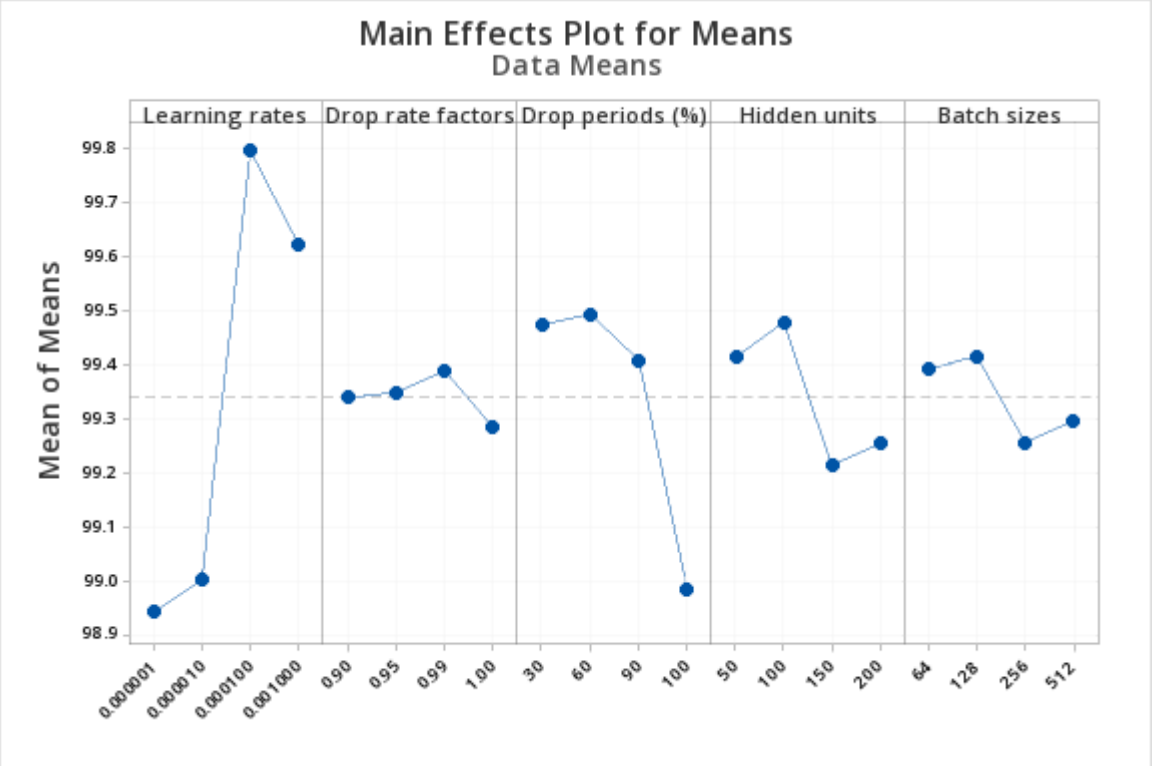


Figure 3.14 Main effects plot for means for the 2nd condition

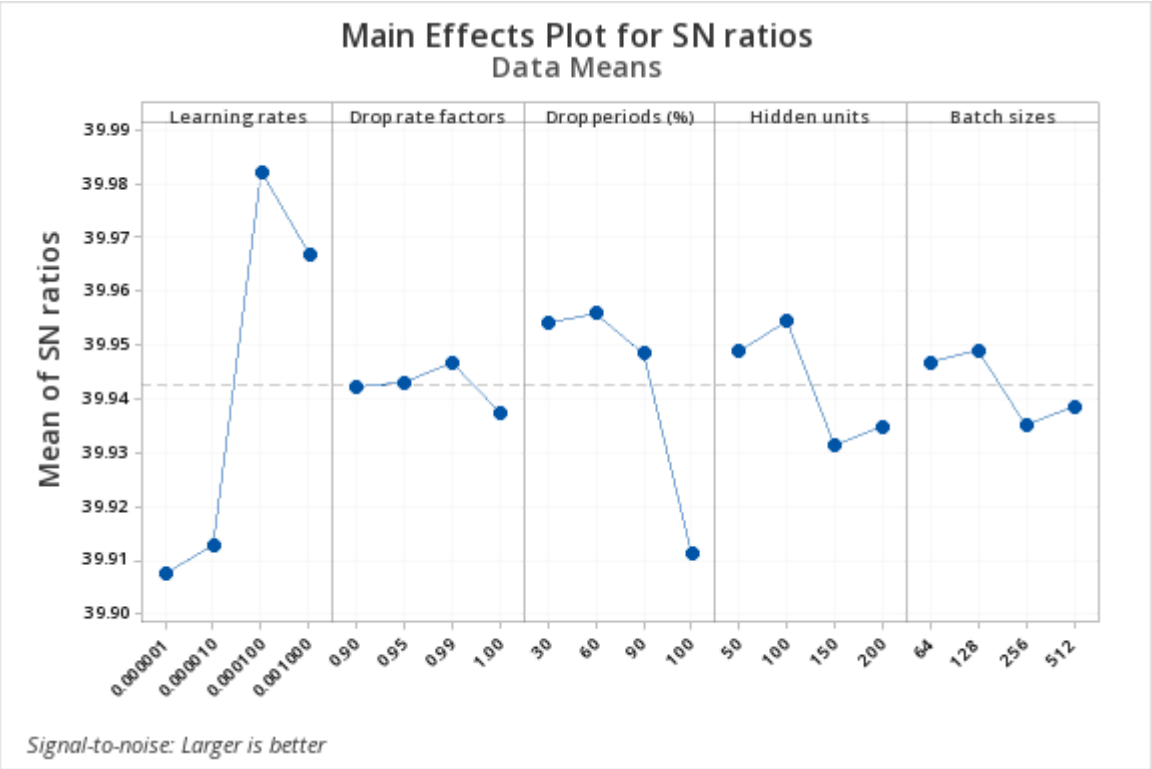


Figure 3.15 Main effects plot for S/N ratios for the 2nd condition

As depicted in Figure 3.13, the most significant hyperparameter is the learning rate, followed by the drop period, number of hidden units, batch size, and drop rate factor. Utilising the "larger is better" criterion, the highest points in Figures 3.14 and 3.15 were selected. The optimal values for the learning rate, drop rate factor, drop period, number of hidden units, and batch size are 0.0001, 0.99, 60, 100, and 128, respectively.

In the last phase, the optimal level of each factor was determined. To determine the optimal factor setting, repeated experiments were executed. This process facilitates verification that specific settings indeed generate the best results. The optimal hyperparameter configurations, derived using trial and error and the Taguchi method, are exhibited in Tables 3.16 and 3.17.

Table 3.16 The optimal hyperparameter configuration of an individual DL model

Factors	LSTM			GRU		
	Condition 1	Condition 2	Condition 3	Condition 1	Condition 2	Condition 3
(1) Learning rates	0.0001	0.0001	0.00001	0.001	0.001	0.00001
(2) Drop rate factors	0.95	0.99	0.95	0.9	1	1
(3) Drop periods (%)	90	60	80	90	60	30
(4) Hidden units	150	100	40	200	200	150
(5) Batch sizes	128	128	128	64	256	128

Table 3.17 The optimal hyperparameter configuration of a combination DL model

Factors	CNN+LSTM			CNN+GRU		
	Condition 1	Condition 2	Condition 3	Condition 1	Condition 2	Condition 3
(1) Number of filters	128	64	512	256	32	128
(2) Filter sizes	7	7	9	3	7	11
(3) Dropout	0.9	0.9	0.5	0.9	0.9	0.3
(4) Hidden units	250	250	50	100	50	100
(5) Learning rates	0.0010	0.0100	0.005	0.0005	0.001	0.0010
(6) Batch sizes	32	256	32	64	256	128

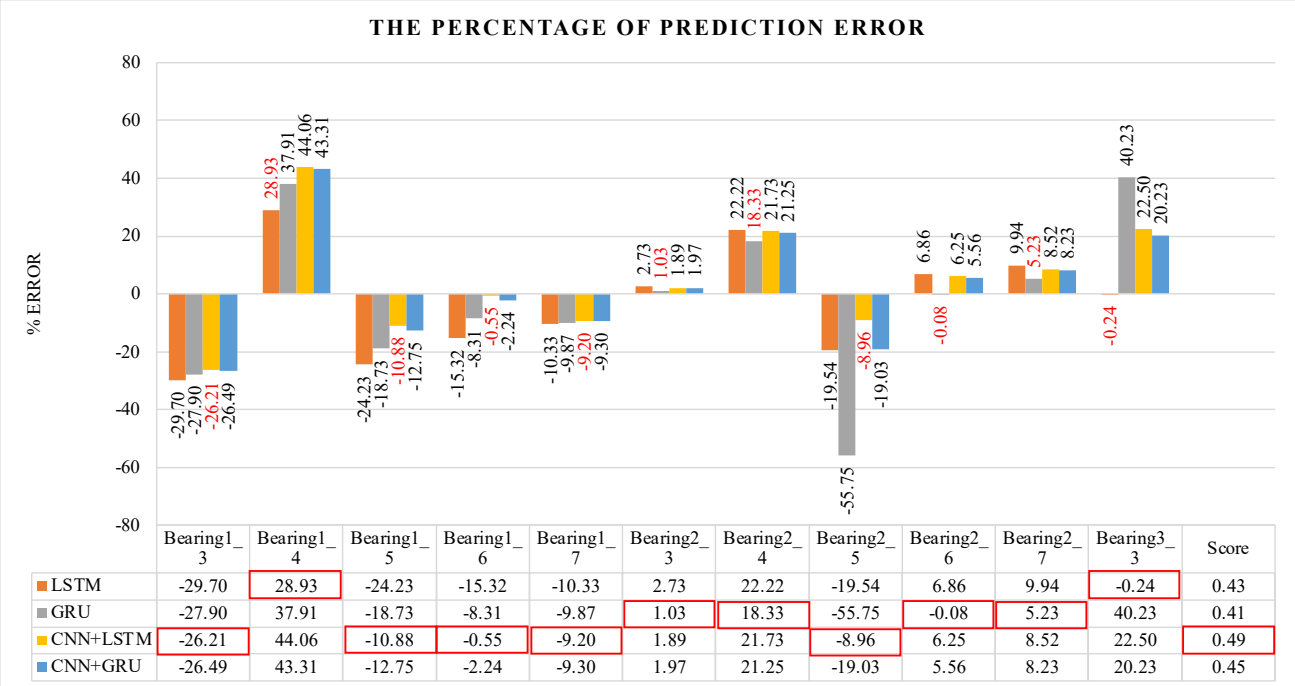
3.5 Results and discussion

The present chapter introduces four DL techniques, namely, LSTM, GRU, a combination of CNN with LSTM and a combination of CNN with GRU, which are applied to forecast the RULs of components before determining subsequent procedures. However, each method's respective number of hyperparameters necessitates careful configuration and fine-tuning of the network to yield optimal outcomes.

This chapter delineates two manual tuning strategies: an iterative process founded on a trial-and-error methodology and the Taguchi experimental design method. Each model condition was trained with ten repetitions for the evaluation and analysis, and the RUL prediction was computed as an average, as revealed in Table 3.18. The performance comparison of each model was then based on the actual RUL data derived from the IEEE PHM 2012 Prognostic Challenge [175]. The error percentage was calculated using Equation 3.14, as shown in Figure 3.16.

Table 3.18 The results of RUL prediction

Test Sets	Actual RUL[175] (Sec.)	LSTM		GRU		CNN+LSTM		CNN+GRU	
		(Sec.)	(%Er)	(Sec.)	(%Er)	(Sec.)	(%Er)	(Sec.)	(%Er)
Bearing1_3	5730	4418	-29.70	4480	-27.90	4540	-26.21	4530	-26.49
Bearing1_4	339	477	28.93	546	37.91	606	44.06	598	43.31
Bearing1_5	1610	1296	-24.23	1356	-18.73	1452	-10.88	1428	-12.75
Bearing1_6	1460	1266	-15.32	1348	-8.31	1452	-0.55	1428	-2.24
Bearing1_7	7570	6861	-10.33	6890	-9.87	6932	-9.20	6926	-9.30
Bearing2_3	7530	7741	2.73	7608	1.03	7675	1.89	7681	1.97
Bearing2_4	1390	1787	22.22	1702	18.33	1776	21.73	1765	21.25
Bearing2_5	3090	2585	-19.54	1984	-55.75	2836	-8.96	2596	-19.03
Bearing2_6	1290	1385	6.86	1289	-0.08	1376	6.25	1366	5.56
Bearing2_7	580	644	9.94	612	5.23	634	8.52	632	8.23
Bearing3_3	820	818	-0.24	1372	40.23	1058	22.50	1028	20.23



	Bearing1_3	Bearing1_4	Bearing1_5	Bearing1_6	Bearing1_7	Bearing2_3	Bearing2_4	Bearing2_5	Bearing2_6	Bearing2_7	Bearing3_3	Score
LSTM	-29.70	28.93	-24.23	-15.32	-10.33	2.73	22.22	-19.54	6.86	9.94	-0.24	0.43
GRU	-27.90	37.91	-18.73	-8.31	-9.87	1.03	18.33	-55.75	-0.08	5.23	40.23	0.41
CNN+LSTM	-26.21	44.06	-10.88	-0.55	-9.20	1.89	21.73	-8.96	6.25	8.52	22.50	0.49
CNN+GRU	-26.49	43.31	-12.75	-2.24	-9.30	1.97	21.25	-19.03	5.56	8.23	20.23	0.45

Figure 3.16 Percentage of prediction errors

As demonstrated in Figure 3.16, the performance of each DL model varies across different operating conditions. For the first condition, the combined DL techniques, CNN with LSTM and CNN with GRU, are two of the most effective prediction models, outperforming the individual DL models, GRU and LSTM, except for Bearing 1_4, which displays contrasting results compared to other test sets. The best percentage of prediction error is observed for Bearings 1_3, 1_5, 1_6 and 1_7, with values of -26.21, -10.88, -0.55 and -9.20, respectively, achieved by the CNN combined with LSTM. Conversely, the LSTM model yields the best result for the Bearing 1_4 test set, with a prediction error percentage of 28.93.

In contrast to the first condition, the single DL model performs better than the combination techniques for the second and third operating conditions. The GRU model achieves the lowest percentage of prediction error for bearings 2_3, 2_4, 2_6 and 2_7, with values of 1.03, 18.33, -0.08 and 5.23, respectively. However, the Bearing 2_5 test set exhibits distinct behaviour, where the combination of the CNN with LSTM outperforms the other models, obtaining the lowest prediction error of -8.96.

LSTM is the best performance prediction model for the third operating condition, with the lowest percentage of prediction error recorded at -0.24. The combinations of CNN with GRU, CNN with LSTM and GRU follow, with prediction error percentages of 20.23, 22.50 and 40.23, respectively.

To further compare the performance of the DL techniques, scores were calculated using Equations 3.15-3.16. The integrated CNN with the LSTM model achieves the highest score of 0.49, followed by the CNN with the GRU at 0.45, the LSTM at 0.43, and the GRU at 0.41. The CNN with LSTM combination is the best model in this study, while the GRU model has the lowest performance. Overall, the

combination of DL models proves more effective in predicting the RUL of bearings, with a performance superior to that of a single DL model.

3.6 Summary

This chapter has presented a comprehensive framework for predicting the RUL of bearings in rotating machinery. The dataset preparation process was described in detail, and the health status of the bearings was defined. Subsequently, four DL models, comprising two individual and two combined models, were selected for the experiments, and their performances were compared. The network configurations were established through manual hyperparameter tuning, trial and error adjustment and the Taguchi method. The experiments presented in this chapter offer fundamental insights and demonstrate the efficiency of DL models, particularly when combined with other DL models, in accurately predicting RULs. These accurate prediction models underscore the potential to reduce uncertainty in the remanufacturing process of returned products, increasing the effectiveness of the management of these products.

Chapter 4

Hyperparameter configuration using the Bees Algorithm

4.1 Preliminaries

Given the nature of DL architectures, several hyperparameters necessitate management. Considering and implementing configuration and tuning methods within DL networks are imperative for attaining superior network performance. A metaheuristic algorithm, the Two-Parameter Bees Algorithm (BA_2), is presented for optimising a set of DL hyperparameters. The benefits of implementing this method include the ability to elaborate on the hyperparameters and reduce the need for a time-consuming manual approach.

4.2 DL configuration and tuning

The hyperparameter configuration can be divided into two primary classes: manual and automatic. The manual modulation of hyperparameters through a labour-intensive process of trial and error necessitates extensive knowledge to discern the optimal hyperparameters conducive to high performance. Conversely, automatic selection of hyperparameters requires a less comprehensive understanding of the most suitable hyperparameters; nevertheless, this approach is generally more computationally demanding than manual methods [180].

While automatic hyperparameter selection algorithms are known to be computationally intensive, a variety of techniques, such as grid search, random search, and model-based hyperparameter optimisation [180], [188], [193]-[196], are routinely employed to optimise hyperparameters within DL frameworks. In addition,

metaheuristic algorithms inspired by animal behaviour patterns, such as the artificial bee colony (ABC) algorithm [197]-[200], firefly algorithm (FA) [197], [201], bat algorithm (BA) [197], grey wolf optimisation (GWO) [144], [152], [202], ant colony [203], [204] and cuckoo search (CS) [205], have gained traction for the purpose of hyperparameter fine-tuning.

The complexity of numerous hyperparameters necessitates automatic hyperparameter selection algorithms to streamline the manual configuration process. A recent research survey presented the Bees Algorithm (BA) for hyperparameter tuning [160], [206]. Three DL hyperparameters were successfully optimised: the learning rate factor of BiLSTM, the number of epochs and the number of LSTM units. In this chapter, an enhanced BA version, known as BA_2, is introduced as a new approach to optimise the DL configuration because BA_2 is a better BA version, and they are currently no publications utilising this algorithm for optimising DL configurations.

4.3 The Two-parameter Bees Algorithm (BA_2)

Another network configuration methodology in this study is an advanced variant of the Bees Algorithm (BA), termed the two-parameter Bees Algorithm (BA_2). The BA was originally conceived by Pham *et al*. in 2005 [207], [208], with BA_2 later introduced by Ismail and Pham [209]. As referenced in an earlier section, studies exploring the use of either the BA or BA_2 in hyperparameter optimisation are lacking. The present experiment aimed to showcase the potential of BA_2 for hyperparameter selection, given that only two parameters are needed: the number of scout bees (n) and the number of worker bees on the elite patches (nep). As a result, BA_2 presents a more straightforward approach to defining initial parameters than does the foundational BA.

An enhanced version integrates explorative and exploitative search strategies, drawing on the traplining metaphor. The recruitment and search procedures distinctly differentiate BA_2 from BA, as the recruited bees are generated only once, following the ranking of each patch in comparison with others. The distribution of worker bee populations within each patch was determined by a linear function, as delineated in Equation 4.1.

$$w_k = w_{max} + (k-1)\left(\frac{w_{min}-w_{max}}{n-1}\right) \tag{4.1}$$

In this equation, w_k signifies the number of worker bees in the k-th patch, k represents the patch's rank, w_{max} is indicative of the maximum worker count, and w_{min} represents the minimum worker count, which corresponds to nep and 1, respectively.

As stated earlier, the foraging mechanism in BA_2 does not demarcate the exploration or exploitation process. Bees tend to cluster in areas with high probabilities of discovering abundant nectar sources, whereas in regions with lower probabilities, they exhibit a propensity to spread over larger expanses. Consequently, each size has its neighbourhood size (ngh_k). Importantly, the radius of the neighbourhood does not represent the maximal extent; instead, it signifies the region with the highest population density. The corresponding equation is provided below.

$$foraging\ point_{k,w_k} = \begin{cases} patch\ point \pm T[0, ngh_k, 1] * (x_{max} - x_{min}) \\ x_{min;}\ foraging\ point_{k,w_k} < x_{min} \\ x_{max;}\ foraging\ point_{k,w_k} > x_{max} \end{cases} \tag{4.2}$$

The $foraging\ point$ represents a position within a specified range where BA_2 is actively searching for resources or solutions. The equilibrium between exploitation and exploration is regulated by the probability distribution ($T[0, ngh_k, 1]$), which depends on the value of ngh_k. The search space is circumscribed by maximum (x_{max})

and minimum (x_{min}) search bounds. An optimal patch typically demonstrates a high concentration of worker bees and a constrained swarm radius ($ngh_k = 0$). In contrast, a patch of lower quality would be identified by a diminished count of worker bees and an expanded swarming radius ($ngh_k = 1$). The BA$_2$ algorithm is visually represented in the flowchart illustrated in Figure 4.1.

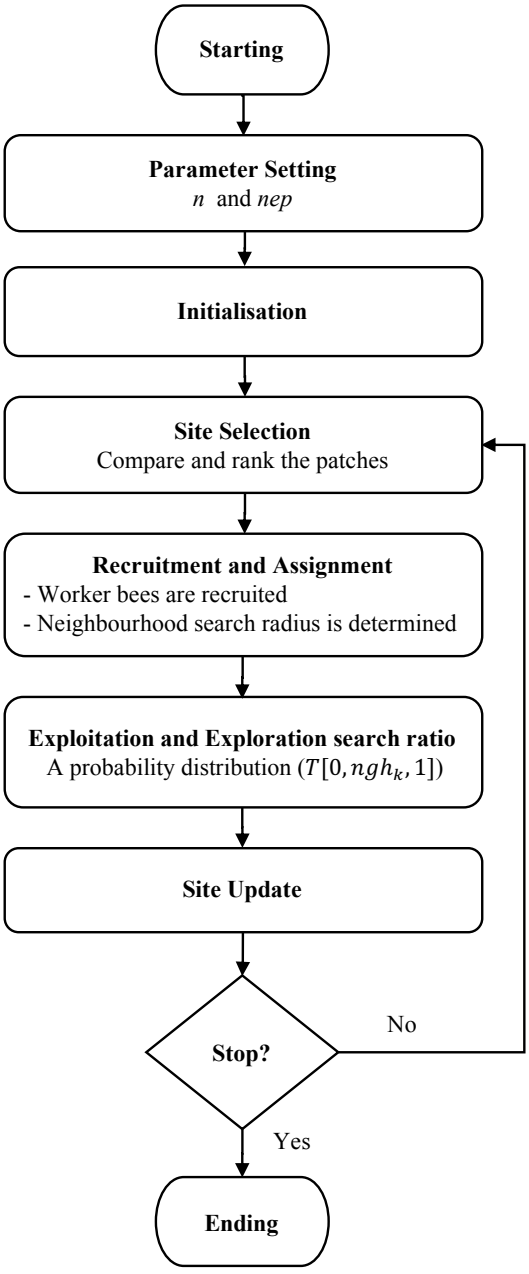


Figure 4.1 The BA$_2$ flowchart

The parameters n and nep and the specification of the stopping criterion are defined for BA_2. The current study adopted the parameters for each prediction model, as presented in Table 4.1.

Table 4.1 Initial BA_2 parameter setting

Prediction Model	**Scout bees(n)**	**Worker bees on the elite patches(nep)**	**Maximum epochs**
LSTM GRU	3	6	300
CNN with LSTM CNN with GRU	10	20	100

The range of hyperparameters, incorporating both maximum and minimum values, was established for the unique hyperparameters of each model, which was used as the search space. Subsequently, a set of n models is generated by modulating the hyperparameters within the specified range. Each model is then subjected to training for a predefined number of epochs. Bees are assigned to conduct both exploratory and exploitative searches at each site commensurate with the fitness of the patch. The aforementioned sites are subsequently ordered in ascending order, with the site incurring the minimum loss identified as the most favourable.

Typically, sites with the highest returns undergo more exploitative searches, while those incurring notable losses are generally subjected to exploratory searches. This procedure is reiterated for a fixed number of cycles, where the optimal bee is subsequently conserved in the predetermined variable, referred to as the "best bee", for each respective iteration. Finally, the hyperparameters are automatically determined and optimised to achieve the highest accuracy.

4.4 BA_2 for hyperparameter optimisation

For this study, the four prediction models discussed in section 3.4 were utilised. The search space for hyperparameter tuning was set for individual and combined DL techniques, encompassing five hyperparameters for the former and seven for the latter. The range of maximum and minimum values was employed to define the search space for hyperparameters such as the number of filters, filter sizes, number of hidden units, dropout probability, output size for fully connected units, initial learning rate, mini-batch size, drop rate factor and drop period. Detailed information on these search spaces can be found in Table 4.2.

Table 4.2 The search space of the hyperparameters

Hyperparameters	LSTM/GRU		CNN+LSTM/CNN+GRU	
	Lower Limit	Upper Limit	Lower Limit	Upper Limit
(1) Number of filters	-	-	4	128
(2) Filter sizes	-	-	2	11
(3) Number of hidden units	10	200	10	200
(4) Dropout probability	-	-	0.1	0.9
(5) Output size for fully connected	-	-	10	200
(6) Learning rate	0.00001	0.01	0.00001	0.01
(7) Mini-batch size	32	512	32	512
(8) Drop rate factor	0	1	-	-
(9) Drop period	10	300	-	-

4.5 Results and discussion

4.5.1 RUL prediction results using BA_2 hyperparameter tuning

BA_2 is a metaheuristic algorithm inspired by animal behavioural patterns and was designed for automatic tuning to optimise the selection of hyperparameters. According to the hyperparameter search space in Table 4.2, the algorithm searches this domain to discern the best hyperparameter configuration. Through iterations of BA_2, the

loss of the model decreased and remained constant while the optimal hyperparameter combinations were achieved. Figure 4.2 visually presents the training model along with the loss plot for each iteration of BA_2. The results of the BA_2 exploration are detailed in Tables 4.3 and 4.4.

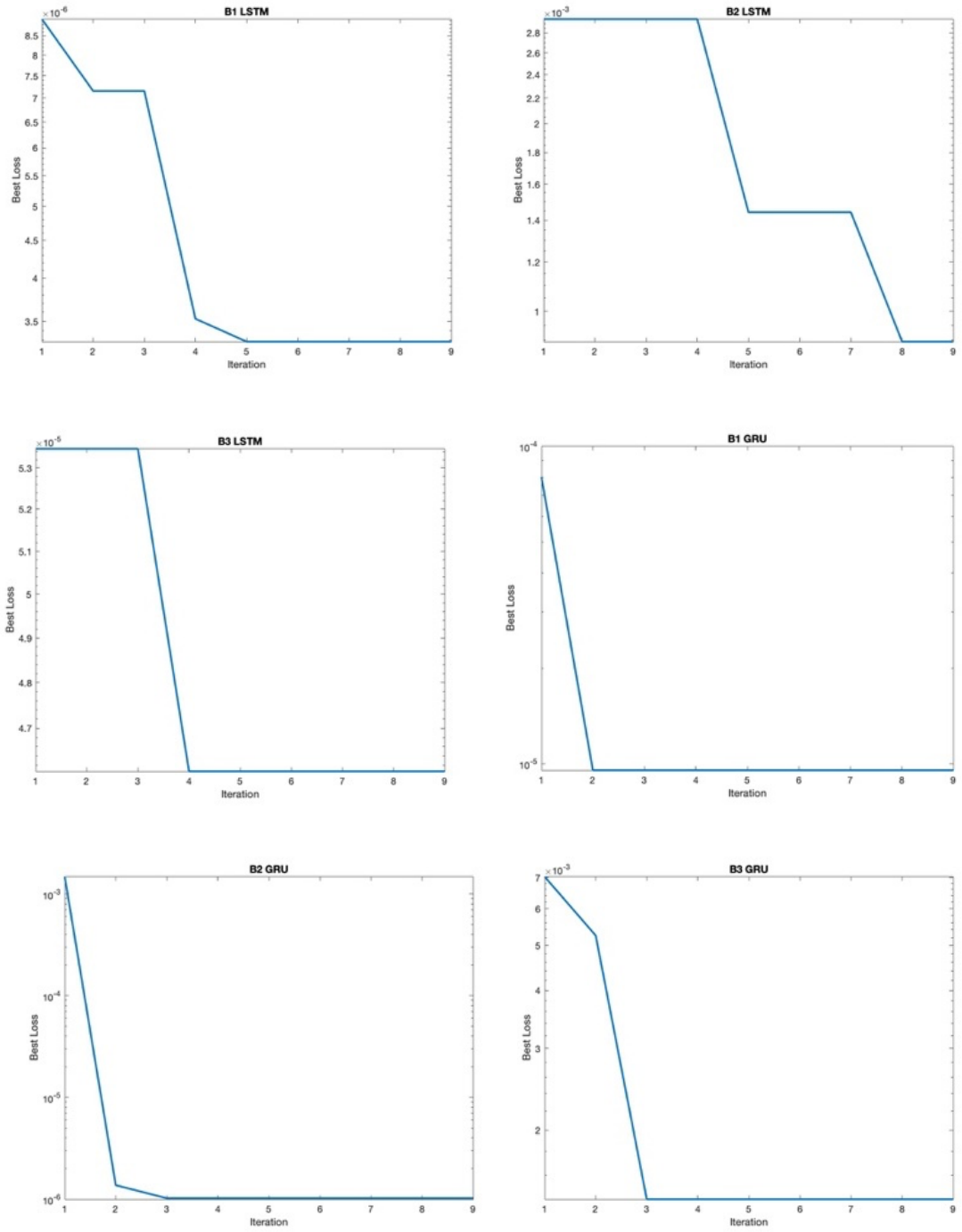


Figure 4.2 Model loss using BA_2

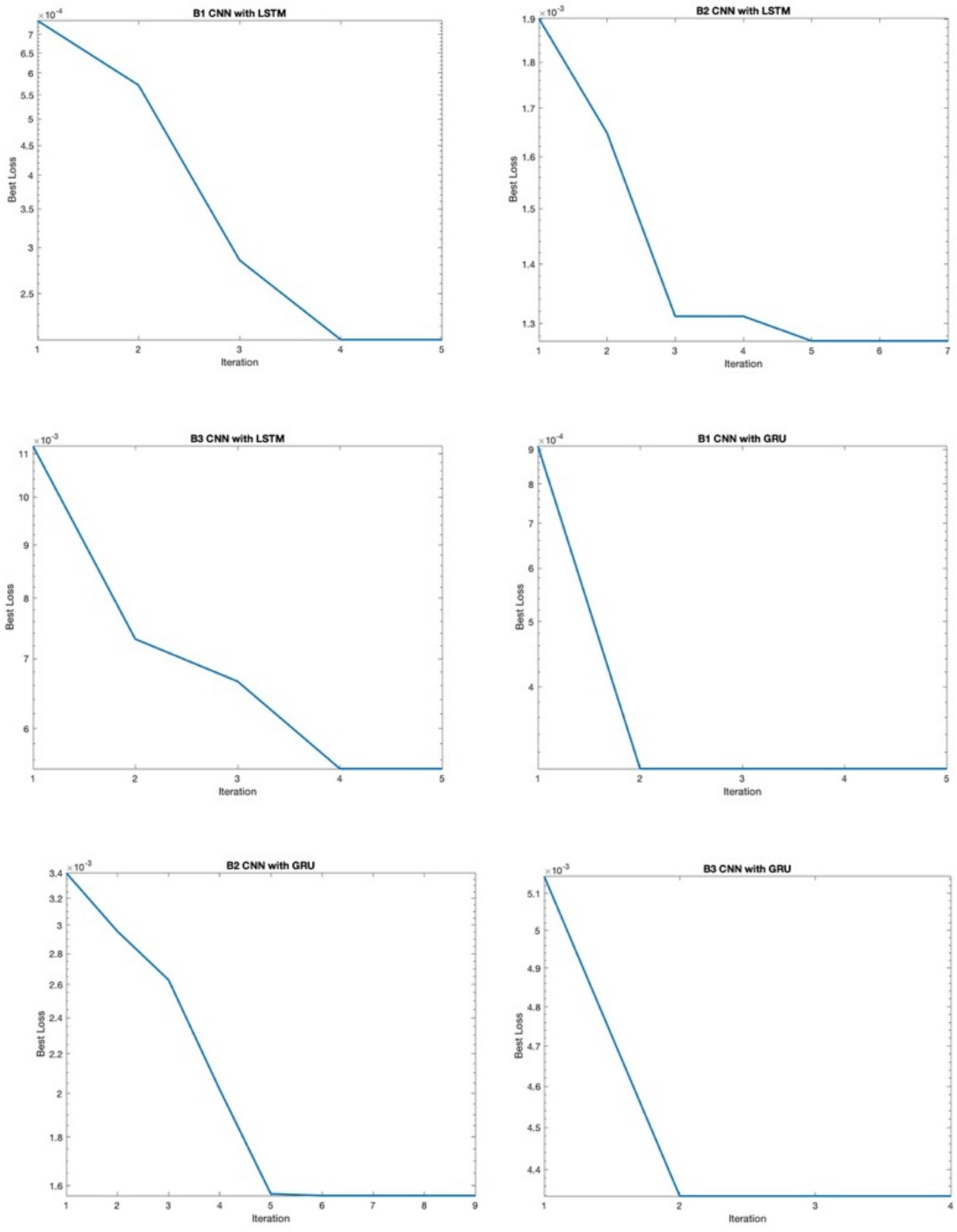


Figure 4.3 Model loss using BA_2 (Cont.)

Table 4.3 The optimal hyperparameter configuration of an individual DL model

Hyperparameters	LSTM			GRU		
	Condition 1	Condition 2	Condition 3	Condition 1	Condition 2	Condition 3
(1) Number of hidden units	200	128	200	128	200	200
(2) Learning rate	0.0005	0.0091	0.0045	0.0075	0.0096	0.0098
(3) Mini-batch size	64	32	256	256	32	512
(4) Drop rate factor	0.1669	0.2962	0.4565	0.0499	0.7287	0.7003
(5) Drop period	190	47	134	172	10	59

Table 4.4 The optimal hyperparameter configuration of a combination DL model

Hyperparameters	CNN+LSTM			CNN+GRU		
	Condition 1	Condition 2	Condition 3	Condition 1	Condition 2	Condition 3
(1) Number of filters	128	4	128	8	16	128
(2) Filter sizes	9	5	2	5	11	7
(3) Number of hidden units	10	131	10	124	10	200
(4) Dropout probability	0.9000	0.9000	0.1000	0.9000	0.9000	0.9000
(5) Output size for fully connected	10	131	10	10	10	10
(6) Learning rate	0.0018	0.0012	0.0044	0.0067	0.0100	0.0012
(7) Mini-batch size	32	32	32	32	32	32

The DL models, which rely on selecting optimal hyperparameters, classify bearing vibration signals into normal or abnormal stages. The first transition point, which splits the state of the bearing signal, has been identified as the initial point of the RUL period. An illustrative example of the predicted bearing health status is presented in Figure 4.3.

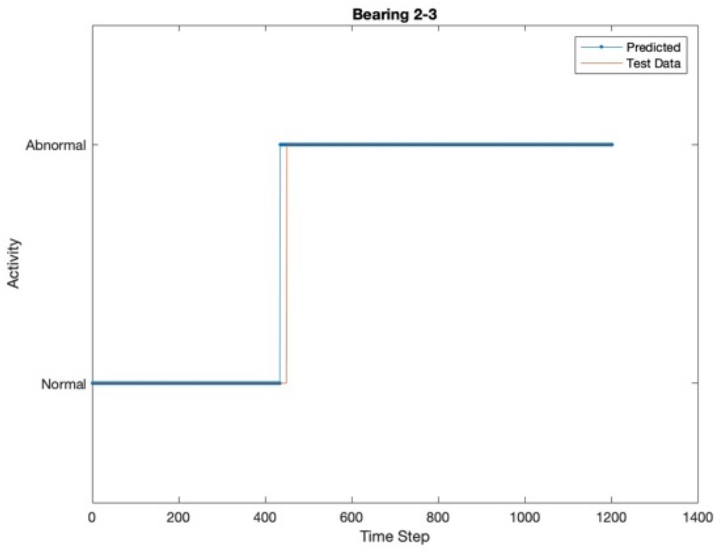
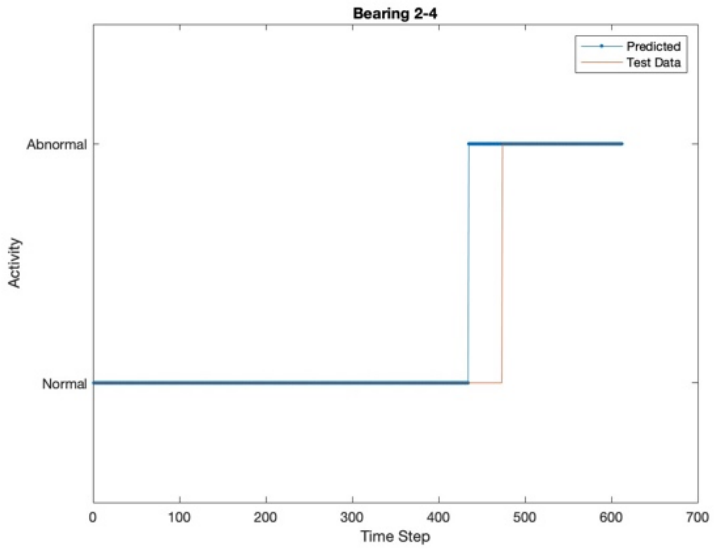


Figure 4.4 Predicted bearing health status



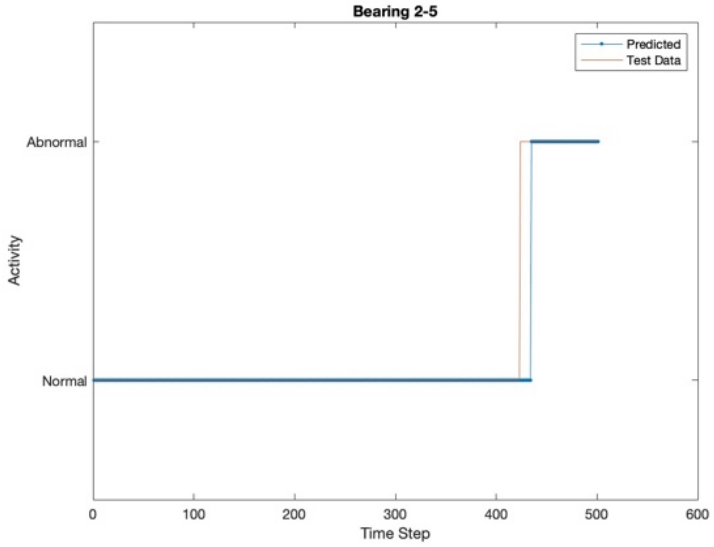


Figure 4.5 Predicted bearing health status (Cont.)

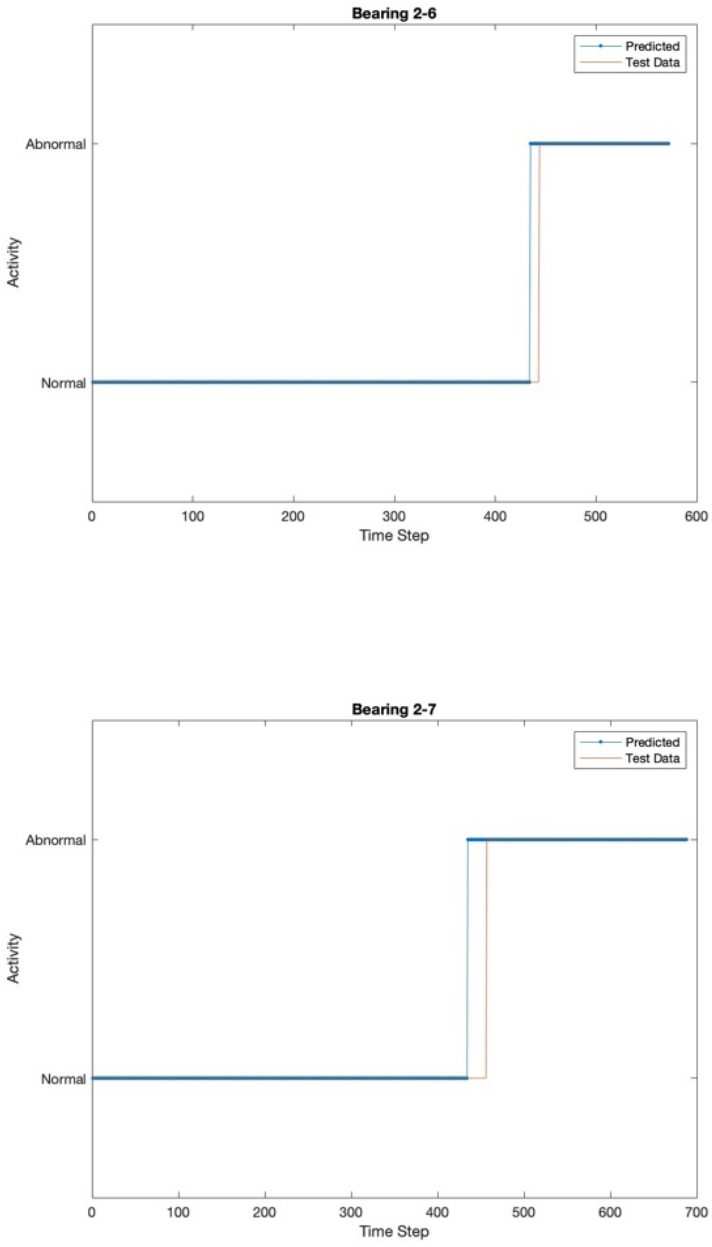


Figure 4.6 Predicted bearing health status (Cont.)

RUL prediction is derived by averaging ten training repetitions for each model condition, during which a conversion from a time step scale to a clock time scale is performed for evaluation and analysis. These results are presented in Table 4.5. Compared to the results published in the IEEE PHM 2012 challenge [175], the percentage prediction error computed using Equation 3.14 is represented in Figure 4.4.

Table 4.5 RUL prediction results

Test Sets	Actual RUL[175] (Sec.)	LSTM		GRU		CNN+LSTM		CNN+GRU	
		(Sec.)	(%Er)	(Sec.)	(%Er)	(Sec.)	(%Er)	(Sec.)	(%Er)
Bearing1_3	5730	4500	-27.33	4525	-26.63	4515	-26.91	4575	-25.25
Bearing1_4	339	566	40.11	594	42.93	590	42.54	634	46.53
Bearing1_5	1610	1404	-14.67	1428	-12.75	1412	-14.02	1508	-6.76
Bearing1_6	1460	1388	-5.19	1428	-2.24	1412	-3.40	1508	3.18
Bearing1_7	7570	6899	-9.73	6923	-9.35	6917	-9.44	6953	-8.87
Bearing2_3	7530	7978	5.62	7765	3.03	7695	2.14	7687	2.04
Bearing2_4	1390	1911	27.26	1854	25.03	1795	22.56	1787	22.22
Bearing2_5	3090	3164	2.34	2972	-3.97	2720	-13.60	2672	-15.64
Bearing2_6	1290	1510	14.57	1453	11.22	1395	7.53	1387	6.99
Bearing2_7	580	668	13.17	654	11.31	639	9.23	637	8.95
Bearing3_3	820	1069	23.29	1056	22.35	1071	23.44	1038	21.00

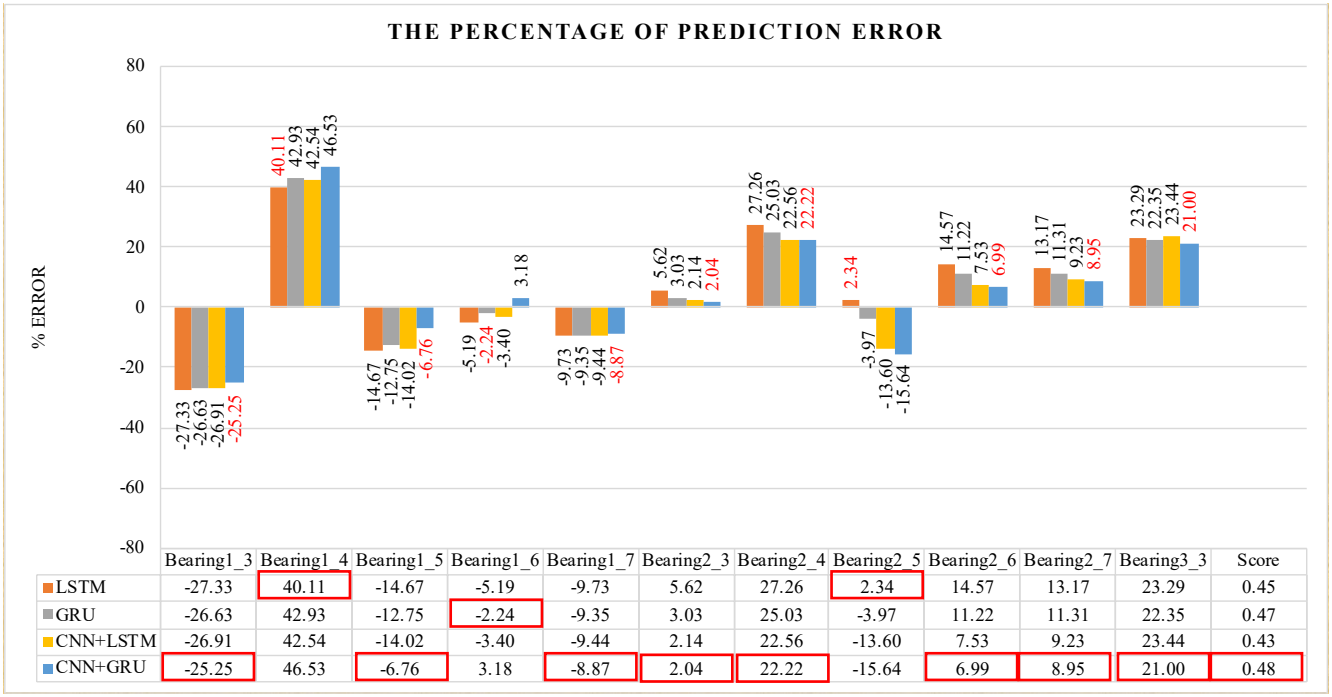


Figure 4.7 Percentage of prediction errors

The best prediction models for each condition vary concerning the percentage of prediction errors. As depicted in Figure 4.4, the combination model, CNN with GRU, yields the best performance for all conditions. However, two specific test sets from the first and second conditions, bearings 1_4 and 2_5, have contrasting results. In this case, a single DL technique, namely, LSTM, proves more suitable for these test sets.

While LSTM performs well with bearings 1_4 and 2_5, it exhibits inferior prediction performance for the remaining test sets within the first and second conditions. However, it should be noted that its prediction performance improves for the third condition. The sorting of prediction errors, ordered by bearing identifiers ranging from bearings 1_3 to 3_3, reveals the lowest percentages of prediction errors as follows: -25.25, 40.11, -6.76, -2.24, -8.87, 2.04, 22.22, 2.34, 6.99, 8.95 and 21.00.

To evaluate the overall prediction performance of the models, prediction scores were computed using Equations 3.15 and 3.16. Among the models, the CNN with a GRU emerges as the top performer, achieving the highest score of 0.48. This is closely followed by GRU and LSTM, with scores of 0.47 and 0.45, respectively. In contrast, the combination of the CNN with the LSTM model demonstrated poorer predictive performance, achieving a score of 0.43.

4.5.2 Comparison of hyperparameter tuning methods

Two network configuration strategies, manual and automatic, were applied in this study. The trial and error and Taguchi methods, which are mentioned in chapter 3, are manual, whereas the BA_2 algorithm represents automatic hyperparameter configuration tuning. This section compares the performances of both approaches. The

percentage of prediction error and the prediction score are utilised as benchmarks for each condition, as depicted in Figures 4.5-4.7.

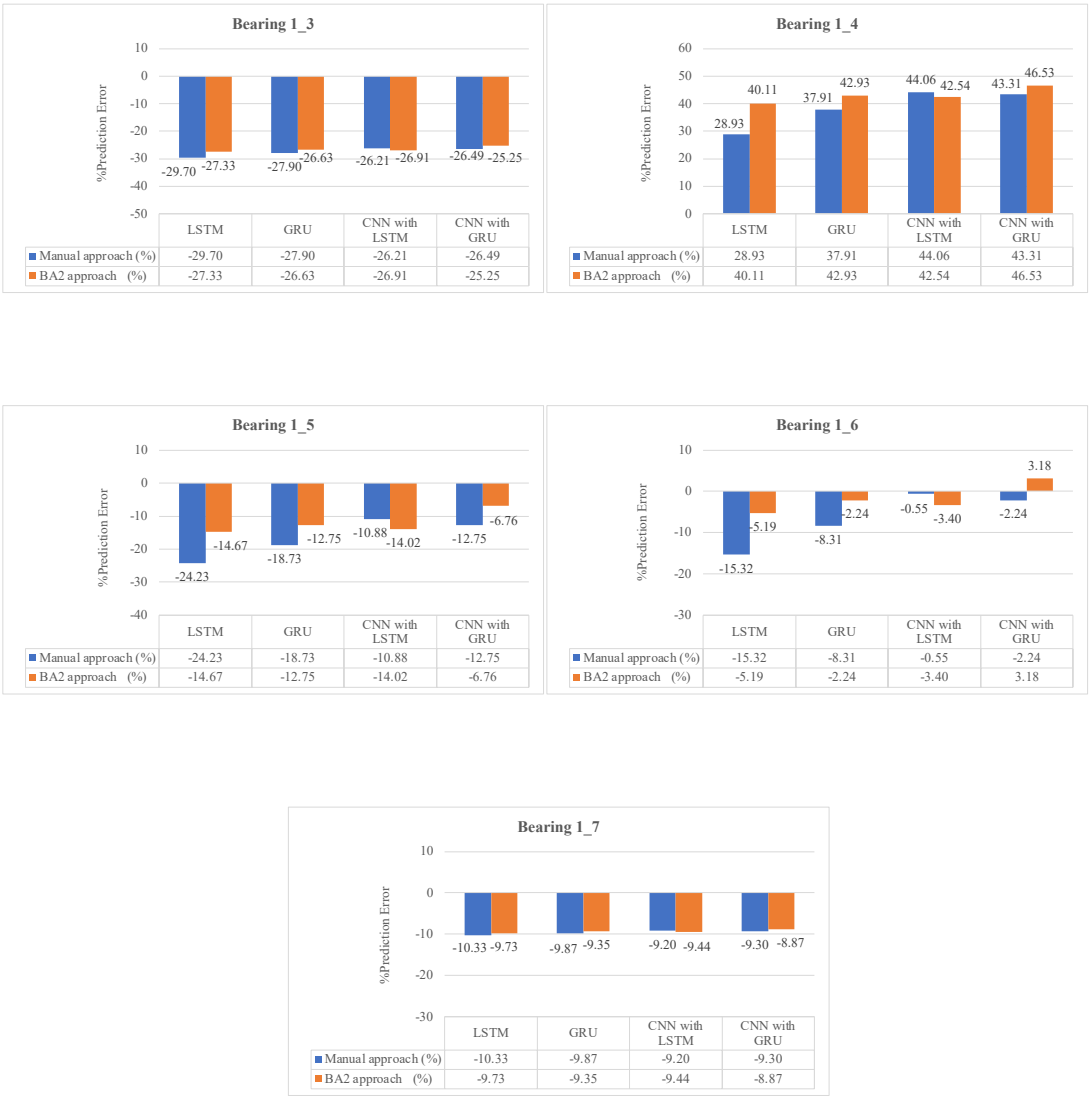


Figure 4.8 Percentage of prediction errors for the 1st condition

Figure 4.9 Percentage of prediction errors for the 2nd condition

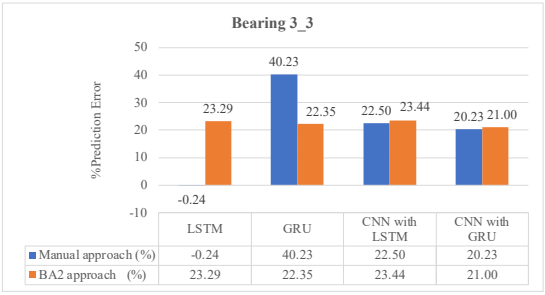


Figure 4.10 Percentage of prediction errors for the 3rd condition

As illustrated in Figure 4.5, BA_2 hyperparameter optimisation demonstrated superior results over manual techniques across the LSTM, GRU and CNN

combined with GRU models, except for bearing 1_4 and bearing 1_6. However, it is evident that both bearing 1_4 and bearing 1_6 in the CNN with the GRU model showed divergent results compared with the other test sets. In contrast, for the CNN combined with the LSTM model, the manual approach provided the best results for the first bearing condition.

In contrast, manual techniques exhibited superior prediction results compared to the utilisation of BA_2 hyperparameter optimisation for all prediction models of the second bearing condition, as depicted in Figure 4.6. Except for bearing 2_5, this pattern revealed divergent outcomes for individual DL models and for the combined CNN with GRU model.

Moving to the final condition, presented in Figure 4.7, it becomes evident that manual network configurations consistently outperformed the performance of the BA_2 technique, with the exception of the GRU model.

A comparison with the computed scores illustrates the prediction models' performance in Figure 4.8. Autotuning of the hyperparameter configurations effectively improved upon that of the LSTM, GRU and combined CNN with GRU models. On the other hand, within the combination model CNN with LSTM, the manual approach yielded a better outcome than did the automated counterpart.

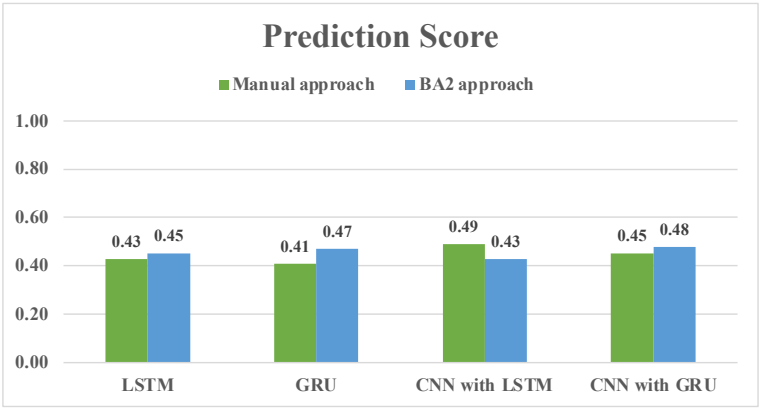


Figure 4.11 Comparison of prediction scores

As a result, the best configurations of hyperparameters were determined by selecting those with the highest prediction score for each method. The best configurations resulting from automatic tuning were chosen for three prediction models: LSTM, GRU and CNN combined with GRU, whereas the configuration for CNN combined with LSTM was derived from manual tuning. The best hyperparameters for each prediction model are summarised in Tables 4.6 and 4.7.

Table 4.6 The best set of hyperparameters for a single DL model

Hyperparameters	LSTM			GRU		
	Condition 1	Condition 2	Condition 3	Condition 1	Condition 2	Condition 3
(1) Number of hidden units	200	128	200	128	200	200
(2) Learning rate	0.0005	0.0091	0.0045	0.0075	0.0096	0.0098
(3) Mini-batch size	64	32	256	256	32	512
(4) Drop rate factor	0.1669	0.2962	0.4565	0.0499	0.7287	0.7003
(5) Drop period	190	47	134	172	10	59

Table 4.7 The best set of hyperparameters for a combination of DL models

Hyperparameters	CNN+LSTM			CNN+GRU		
	Condition 1	Condition 2	Condition 3	Condition 1	Condition 2	Condition 3
(1) Number of filters	128	64	512	8	16	128
(2) Filter sizes	7	7	9	5	11	7
(3) Number of hidden units	250	250	50	124	10	200
(4) Dropout probability	0.9000	0.9000	0.5000	0.9000	0.9000	0.9000
(5) Output size for fully connected	250	250	50	10	10	10
(6) Learning rate	0.0010	0.0010	0.0050	0.0067	0.0100	0.0012
(7) Mini-batch size	32	256	32	32	32	32

In conclusion, despite the inability of the BA_2 hyperparameter optimiser to excel fully across all model instances, this methodology is still better than manual techniques 75% of the time.

First, employing BA_2 for model configuration does not necessitate in-depth knowledge to ascertain the most effective hyperparameters conducive to optimal performance. Even though the computational time associated with BA_2 is considerably high, this approach still allows for faster attainment of results than manual methods.

Finally, managing a multitude of hyperparameters for a complex DL model is considerably simpler with BA_2 than with manual tuning. For these compelling reasons, optimising the hyperparameter search using BA_2 could substantially enhance the prediction efficiency and be effectively integrated into predictive modelling.

4.6 Summary

The hyperparameter configuration plays a vital role in selecting suitable hyperparameters for DL architectures to achieve the highest performance. The introduction of automatic configuration tuning via BA_2 is presented in this chapter. The networks were established using the optimally selected hyperparameter sets and employed for predicting the RUL of bearings. The assessment of DL model performance was conducted through the analysis of prediction error percentages.

The hyperparameter optimisation of BA_2 exhibits limitations in achieving optimal performance across some models. Nonetheless, the advantages of this approach continue to surpass those of manual techniques in terms of configuration simplicity and hyperparameter management.

Chapter 5

Deep Learning Techniques with the Bees Algorithm for Remaining Useful Life Prediction

5.1 Preliminaries

The Bees Algorithm (BA), inspired by patterns observable in animal behaviour, was implemented with DL techniques. The objective was to circumvent local optima and expedite the convergence toward global optima. It is anticipated that this research will contribute to developing DL models designed to enhance prediction accuracy, thereby mitigating the uncertainties associated with the return of products to the remanufacturing system.

5.2 Basic Bees Algorithm (BA)

The BA, conceived by simulating the foraging behaviour of honey bee swarms, was initially introduced by Pham *et al*. in 2005 [207], [208]. Foraging in bee colonies begins with scout bees searching for promising flower patches. These scouts traverse randomly among patches, and upon locating a patch that meets a certain quality threshold, based on aspects such as sugar content, they return to the hive to perform a unique form of communication known as a "waggle dance." This dance is instrumental in transmitting three crucial details about the flower patch: its direction, the distance from the hive, and quality rating. Learning from this dance, bees in the hive are equipped with the information to navigate these food sources accurately.

After the dance, the scout bee revisits the flower patch, accompanied by follower bees. The number of follower bees deployed depends on the patch’s promise, optimising

the colony's food-gathering efficiency. Bees also monitor the food available at the patch, determining whether to promote the source in future dances based on its sustained value as a food supply. Figure 5.1 provides an overview of the BA.

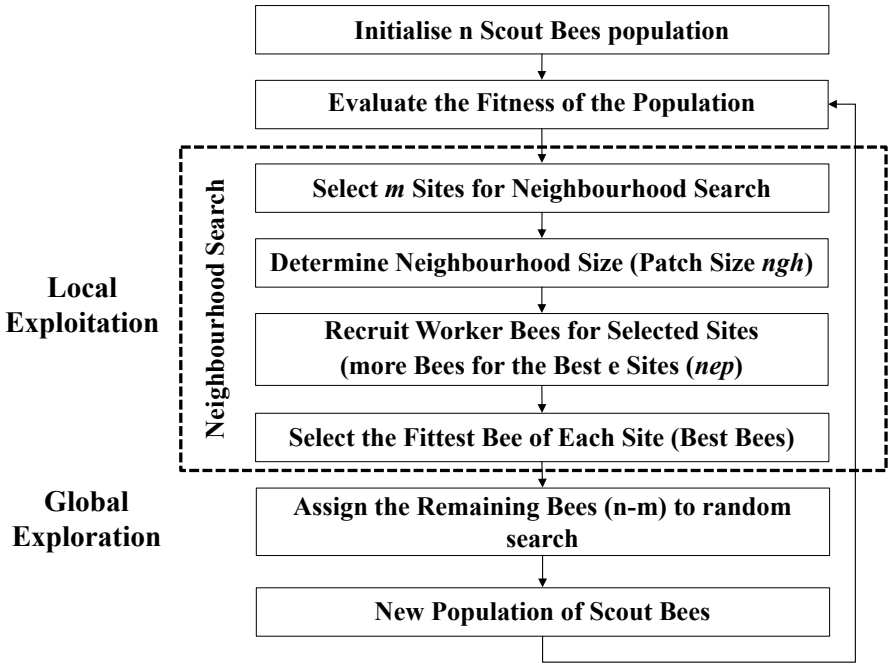


Figure 5.1 The BA flowchart

The algorithm integrates neighbourhood search with random search to find the optimal solution. This process commences by designating a specified number of scout bees (n), which proceeds to conduct a random exploration of the solution spaces. The potential solution at the visited site was evaluated for each of the scout bees using a predefined fitness function. Based on the fitness information gathered by the scout bees, the visited sites are ranked, and the (m) best sites demonstrating the highest fitness is selected for local exploitation.

The scout bees that return from these selected best sites perform a "waggle dance" to recruit other bees for localised exploitation. This recruitment process results in a number of forager bees following the scout bees to the selected sites. The number

of foragers assigned to each selected site is determined using a deterministic method. Priority for recruitment of foragers (nep) is given to the elite site (e), the highest rated among the selected best sites. The remaining sites, determined by $m - e$, recruit some foragers (nsp) for a neighbourhood search, where nsp is less than nep.

A more significant proportion of the bee population is assigned to search the solution space around the e sites. This leads to a concentrated local search in these elite neighbourhoods, regarded as the most promising solution locations. The fitness of the locations within each flower patch was subsequently assessed. If a recruited bee discovers a location with a higher fitness value than the scout bee, it assumes the role of the new scout bee. Only the bee with the highest fitness within each patch is retained at the end of this process. This bee, having identified the fittest solution, represents the entire flower patch and transitions into the 'dancer' bee upon returning to the hive.

During the global search phase, the algorithm randomly utilises the remaining $(n - m)$ bees to scout the fitness landscape for new flower patches. The algorithm continues to iterate until a stopping criterion is reached. Upon the conclusion of each iteration, a refreshed bee colony population is formed that comprises two distinct groups. The first group, representing the results of the local exploitative search, consists of the scout bees. The second group, symbolising the global explorative search results, consists of the $(n - m)$ scout bees, each associated with a randomly generated solution. The colony size was calculated by using Equation 5.1 [209], and Figure 5.2 shows the pseudocode for the BA.

$$\text{Colony size} = (n - m) + (e * nep) + \big((m - e) * nsp\big) \qquad (5.1)$$

1. Initialise population with random solutions.
2. Evaluate fitness of the population.
3. While (stopping criterion not met)
   //Forming new population.
4. Select sites for neighbourhood search.
5. Recruit bees for selected sites (more bees for best e sites) and evaluate fitnesses.
6. Select the fittest bee from each patch.
7. Assign remaining bees to search randomly and evaluate their fitnesses.
8. End While.

Figure 5.2 The BA pseudocode [208]

5.3 DL learnable parameters

In deep learning, learnable parameters denote the variables within a model subject to adaptation through training with datasets. These parameters undergo iterative modifications throughout training to minimise a designated loss function. This loss function quantifies the disparity between the predictions from the model and the actual target values. The objective of training a deep learning model is to discern the configuration of parameters that yield the lowest attainable loss function value for the training data.

There exist two categories of learnable parameters: weights and biases. Weights constitute the principal determinants of neural network behaviour and are employed in the computational operations of each network layer. Typically, weights are used to multiply the input data to compute the output. Each neuron within a layer is endowed with a vector of weights, which is subject to adjustment throughout the training process to enhance the accuracy of the model's predictions. After applying weights, biases are incorporated into the outputs of the computational operations of each layer. Analogous

to weights, biases undergo adjustments during training. They give the model additional freedom, facilitating the independent shifting of a neuron's output relative to its input.

In this study, the learnable parameters that have been updated include weights and biases in various layers, including the convolutional layer, batch normalisation layer, fully connected layers, LSTM layer and GRU layer. In the convolutional layer, weights are represented as three-dimensional (3D) volumes (width × height × depth) convolved with the input volume to yield an output feature map. Each feature map has a corresponding bias term. The weights undergo modification contingent on the error backpropagated from the output, while the biases are parameters that are appended to the outcome of the convolutional operation. During backpropagation, the gradients for weights and biases are computed and employed to adjust these parameters. The cumulative parameter count in a convolutional layer can be calculated as depicted in Equations 5.2 and 5.3 [210].

$$\text{Weights} = \text{Filter size} * \text{Number of channels} * \text{Number of filters} \quad (5.2)$$

$$\text{Biases} = 1 * \text{Number of filters} \quad (5.3)$$

The learnable parameters within the batch normalisation layer encompass the scale and offset parameters. They are updated during backpropagation to normalise the outputs of the previous layer. The scale parameter adjusts the standard deviation of the activations while the offset adjusts its mean. This contributes to a reduction in internal covariate shifts, thereby augmenting the effectiveness of the training process. The dimensions of the scale and offset are contingent on the nature of the layer input. In this study, the aggregate number of parameters is represented in Equation 5.4 [211].

$$\text{Scale or offset} = \text{Number of channels} * 1 \quad (5.4)$$

The methodology for adjusting weights and biases within the fully connected layers resembles that within the convolutional layer. Updating is executed based on the error that is backpropagated from the output. The gradients for weights and biases are derived during backpropagation and subsequently harnessed to adjust these parameters. Equations 5.5 and 5.6 [212] present the calculations for the number of parameters.

Weights = An output size * Input size (5.5)

Biases = An output size * 1 (5.6)

Within the LSTM and GRU layers, the input weights, recurrent weights, and biases are subject to modification during backpropagation through time (BPTT), a variant of backpropagation designed for sequences. Input weights are used when the current input is channelled into the LSTM and GRU, recurrent weights are applied to the previous hidden state, and biases are appended after the multiplication of weight and input. The gradients for these parameters are computed and utilised to effect adjustments. Nonetheless, a salient distinction is evident in the quantity and configuration of these parameters. LSTMs possess more learnable parameters than GRUs and can be calculated employing Equations 5.7 through 5.12 [213], [214].

Input weights:

LSTM = 4 * Number of Hidden Units * Input Size (5.7)

GRU = 3 * Number of Hidden Units * Input Size (5.8)

Recurrent weights:

LSTM = 4 * Number of hidden units * Number of hidden units (5.9)

GRU = 3 * Number of Hidden Units * Number of Hidden Units (5.10)

Biases:

LSTM = 4 * Number of Hidden Units * 1 (5.11)

GRU = 3 * Number of Hidden Units * 1 (5.12)

5.4 Experimental design

5.4.1 BA-Adaptive Moments (ADAM) for optimising DL learnable parameters

As described in section 5.2, the Bees Algorithm is an algorithm based on swarm intelligence that simulates the foraging behaviour of honey bees when identifying food sources. The algorithm employs a population of virtual bees to systematically probe regions and ascertain optimal food sources by utilising stochastic search techniques and selection procedures that prioritise high-quality food sources. The BA exhibits versatility and applies to optimisation problems and optimal pathfinding tasks.

Adam (Kingma and Ba,2014 [27]) is introduced as an adaptive learning rate optimisation algorithm, deriving its name from “adaptive moments”. It can be conceptualised as a synthesis of RMSProp and momentum with distinct characteristics. Primarily, ADAM incorporates momentum by estimating the first moment of the gradient via exponential weighting. Additionally, ADAM implements bias corrections for the first and second moments to mitigate initialisation biases. Generally, ADAM is considered resilient concerning hyperparameter selection, although modifications to the learning rate are sometimes necessary [77].

Khan *et al*. [215] reports successfully integrating the ADAM algorithm with a swarm intelligence-based algorithm. Specifically, ADAM was utilised to augment the Beetle Antennae Search (BAS) algorithm by modulating the BAS step size in each iteration; this algorithm is termed BAS-ADAM. This amalgamation demonstrated an accelerated convergence rate, particularly in constrained valley scenarios. One

prominent feature of the ADAM update rule is its ability to independently adjust the step size for each dimension instead of employing a uniform step size.

This chapter introduces a methodology to incorporate the ADAM optimisation algorithm within the BA, drawing inspiration from Khan *et al*. [215], intending to augment the efficiency of the search procedure of the Bees Algorithm. This amalgamated algorithm optimises DL learnable parameters to enhance RUL prediction performance. By integrating the BA with ADAM optimisation, the trajectory of the virtual bees is more effectively oriented toward propitious regions within the search space. Moreover, the adaptive learning rates furnished by ADAM facilitate expedited and efficient convergence to the optimal solution. The flowchart of BA-ADAM is illustrated in Figure 5.3

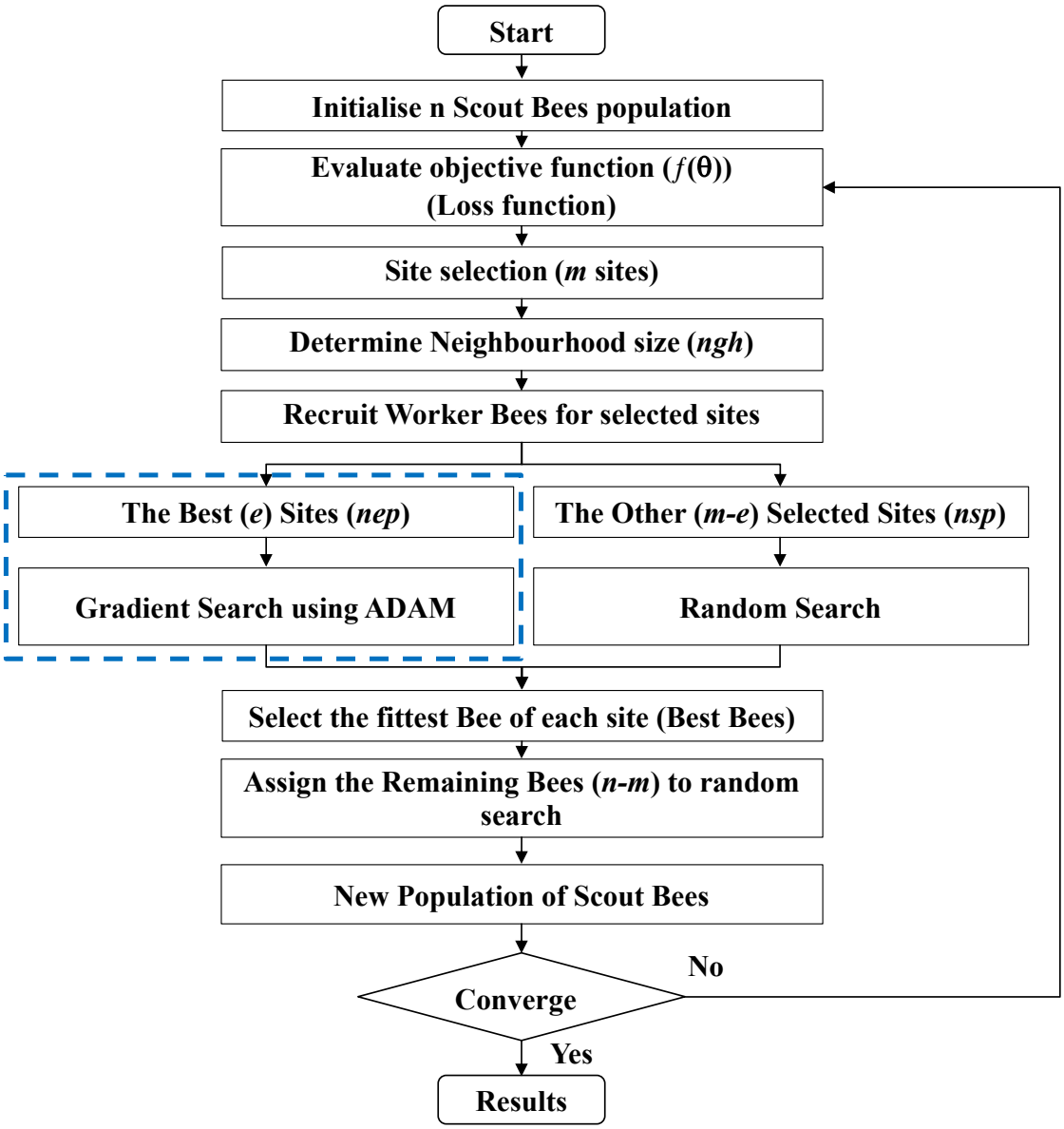


Figure 5.3 Flowchart of the BA-ADAM algorithm

5.4.2 Experimental and Parameter Setup

The experiment commenced by configuring four DL model architectures, namely, LSTM, a GRU, a CNN with LSTM, and a CNN with a GRU. These parameters were established utilising the optimal hyperparameters delineated in Tables 4.3-4.4 and the BA parameters provided in Table 5.1.

Table 5.1 BA parameters

Parameter	Value
nScoutBee(n)	30
nEliteSiteBee(nep)	10
nSelectedSiteBee(nsp)	5
nEliteSite(e)	1
nSelectedSite(m)	5
Neighbourhood Size(ngh)	0.02
MaxIt	20

The initial parameters, including weights and biases derived from the DL network, were transformed into a format amenable to integration with the BA. The aggregate count of DL parameters for each model can be computed employing the equations outlined in section 5.3.

As depicted in Figure 5.3, the BA optimises the model through iterative processing in the main loop. Within each iteration, new DL parameters were synthesised from the initial parameters, employing a random search mechanism attributable to scout bees. With respect to the elite sites (e), parameter updating is executed through the ADAM optimiser, which calculates gradients to minimise the binary cross-entropy losses based on the objective function via the employment of selected bees at the elite site (nep). Conversely, parameter updating at the other selected sites ($m - e$) was achieved through conventional foraging mechanisms. After this, the bees are ranked based on their losses, and the optimal solution is updated.

After completing BA iterations, the best parameters were extracted and allocated to the DL model. Subsequently, the DL networks were subjected to training employing the optimised weights and biases in conjunction with the ADAM optimiser. Finally, the

trained network represented the final output of the algorithm. The experimental procedure is depicted in Figure 5.4.

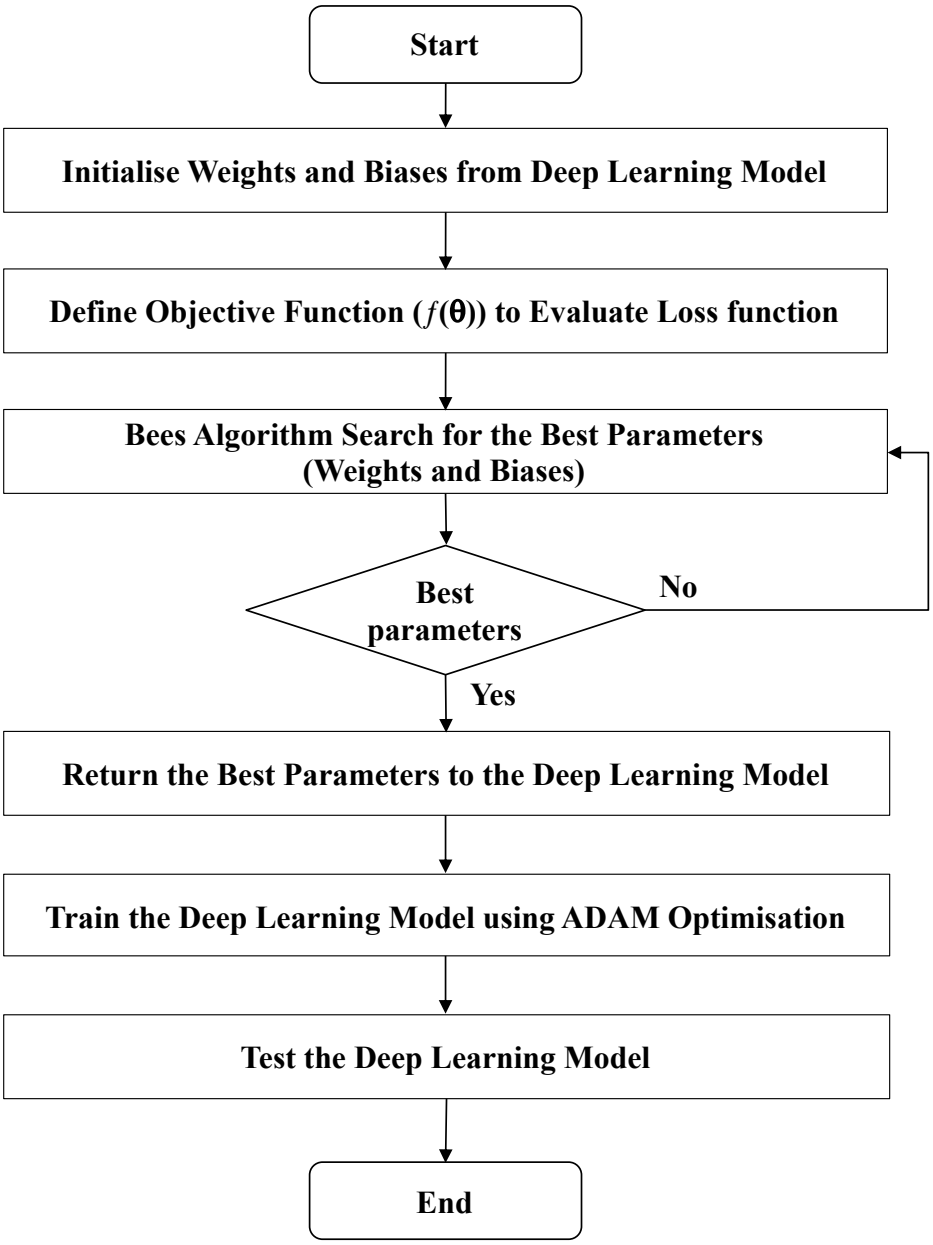


Figure 5.4 Experimental procedure

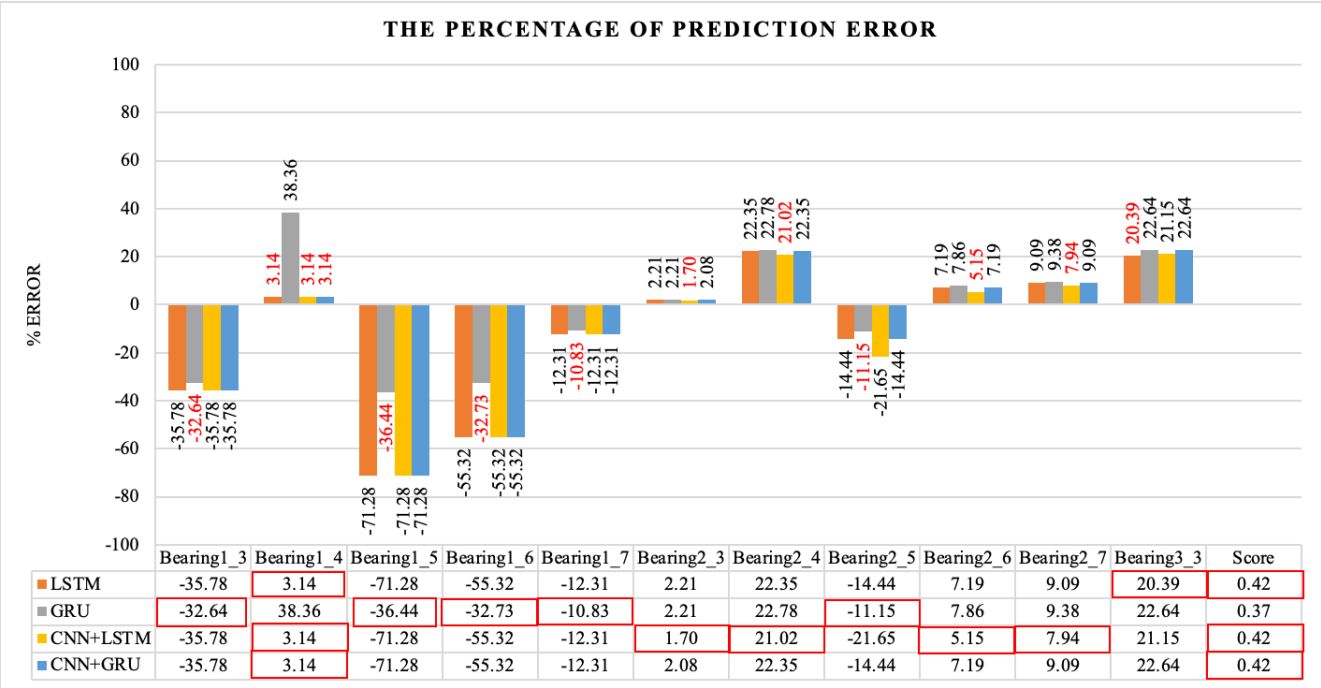
5.5 Results and discussion

5.5.1 RUL prediction results using BA-ADAM Optimisation

Combining the BA with ADAM optimisation capitalises on the respective advantages of both algorithms. The capacity of BA for exploration and enhancement, when coupled with the speed and efficiency of ADAM optimisation in updating weights and biases, has the potential to enhance the efficacy of training and fine-tuning DL models. The RUL predictions are illustrated in Table 5.2, and the percentage of prediction errors for each test set with ten repetitions was calculated using Equation 3.14 and compared to the actual RUL reported in the IEEE PHM 2012 challenge [175]. Concerning the different weight assessments of the prediction error percentage, the scores that benchmark each DL technique's overall performance were determined using Equations 3.15-3.16 and are represented in Figure 5.5.

Table 5.2 RUL prediction results using BA-ADAM Optimisation

Test Sets	Actual RUL[175] (Sec.)	LSTM		GRU		CNN+LSTM		CNN+GRU	
		(Sec.)	(%Er)	(Sec.)	(%Er)	(Sec.)	(%Er)	(Sec.)	(%Er)
Bearing1_3	5730	4220	-35.78	4320	-32.64	4220	-35.78	4220	-35.78
Bearing1_4	339	350	3.14	550	38.36	350	3.14	350	3.14
Bearing1_5	1610	940	-71.28	1180	-36.44	940	-71.28	940	-71.28
Bearing1_6	1460	940	-55.32	1100	-32.73	940	-55.32	940	-55.32
Bearing1_7	7570	6740	-12.31	6830	-10.83	6740	-12.31	6740	-12.31
Bearing2_3	7530	7700	2.21	7700	2.21	7660	1.70	7690	2.08
Bearing2_4	1390	1790	22.35	1800	22.78	1760	21.02	1790	22.35
Bearing2_5	3090	2700	-14.44	2780	-11.15	2540	-21.65	2700	-14.44
Bearing2_6	1290	1390	7.19	1400	7.86	1360	5.15	1390	7.19
Bearing2_7	580	638	9.09	640	9.38	630	7.94	638	9.09
Bearing3_3	820	1030	20.39	1060	22.64	1040	21.15	1060	22.64



	Bearing1_3	Bearing1_4	Bearing1_5	Bearing1_6	Bearing1_7	Bearing2_3	Bearing2_4	Bearing2_5	Bearing2_6	Bearing2_7	Bearing3_3	Score
LSTM	-35.78	3.14	-71.28	-55.32	-12.31	2.21	22.35	-14.44	7.19	9.09	20.39	0.42
GRU	-32.64	38.36	-36.44	-32.73	-10.83	2.21	22.78	-11.15	7.86	9.38	22.64	0.37
CNN+LSTM	-35.78	3.14	-71.28	-55.32	-12.31	1.70	21.02	-21.65	5.15	7.94	21.15	0.42
CNN+GRU	-35.78	3.14	-71.28	-55.32	-12.31	2.08	22.35	-14.44	7.19	9.09	22.64	0.42

Figure 5.5 Percentage of prediction errors

According to Figure 5.5 and considering each condition, it can be observed that the individual approaches, GRU and LSTM, perform well in the first and third conditions, respectively. On the other hand, the hybrid approach, which combines a CNN with LSTM, yields the lowest prediction error under the second condition. However, a different result is observed for two test sets, bearing 1_4 and bearing 2_5. LSTM, CNN combined with LSTM, and CNN combined with GRU outperformed GRU for bearing 1_4, whereas LSTM and CNN combined with GRU were the best prediction models for bearing 2_5.

The percentages of prediction errors, arranged from the first to the third condition of the test set, are as follows: -32.64, 3.14, -36.44, -32.73, -10.83, 1.70, 21.02, -11.15, 5.15, 7.94, and 20.39. The results indicate that the predictions exhibit both overestimation and underestimation. Therefore, prediction scores are applied to evaluate the overall model performance. Although the GRU model performs best for the first

condition, it performs the worst overall, with a score of 0.37. Conversely, the remaining prediction models, LSTM and both hybrid techniques, achieved the highest score, with a quality score of 0.42 using this hybrid optimisation.

5.5.2 Comparison of optimisation algorithms

The adaptive learning rate (ADAM) optimisation technique is widely used in DL model training to update weights and biases iteratively. The hybrid approach, which integrates the ADAM optimisation algorithm into the BA framework, aims to enhance the efficiency of training DL models. This section presents a performance comparison of DL models between ADAM and the proposed optimisation algorithm. The results for each condition are depicted in Figures 5.6-5.8.

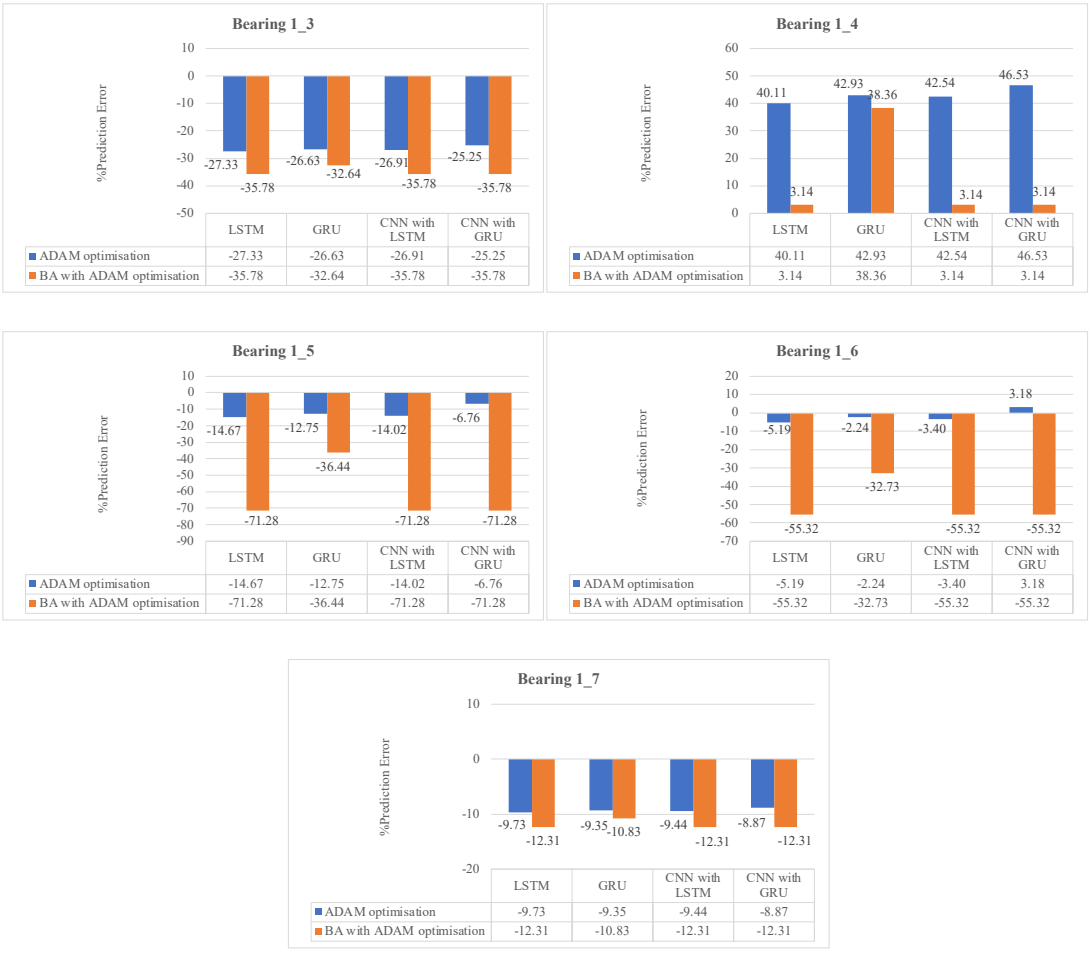


Figure 5.6 Percentage of prediction errors for the 1st condition

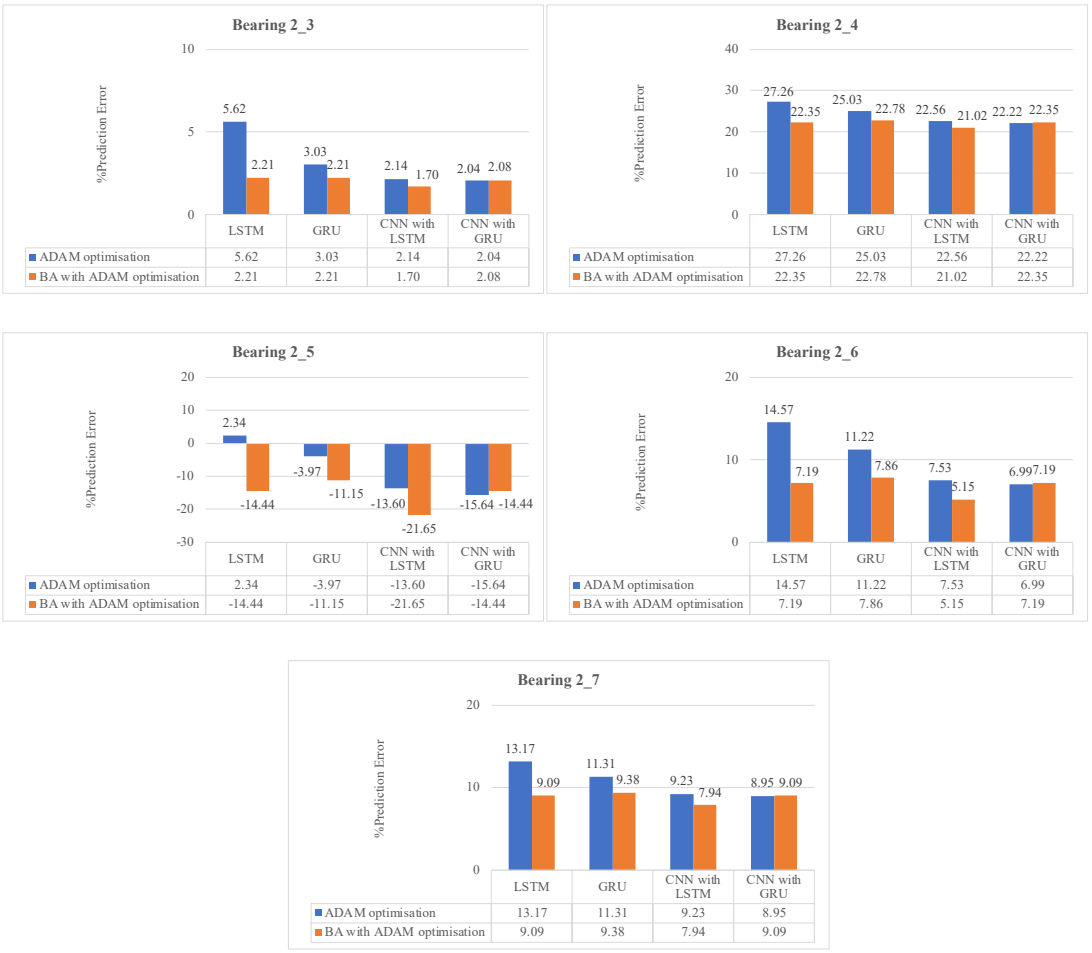


Figure 5.7 Percentage of prediction errors for the 2nd condition

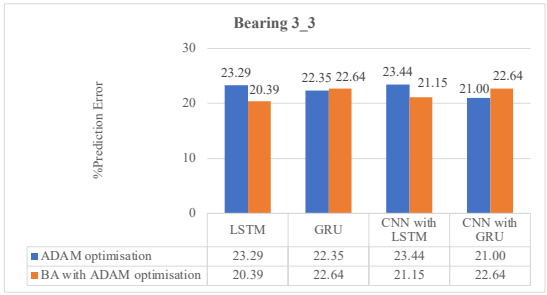


Figure 5.8 Percentage of prediction errors for the 3rd condition

In the context of the first condition, training DL models using ADAM optimisation generally outperformed the hybrid approach, except for bearing 1_4, as shown in Figure 5.6. In contrast, the proposed algorithm demonstrated superior performance in most models of the second condition, as revealed in Figure 5.7.

However, it is worth noting that, compared with those of the other models and test sets, only the combinations of the CNN with the GRU model and bearing 2_5 exhibited contrasting results.

The results of the last condition differed from those of the first and second conditions. The observed algorithm outperformed ADAM optimisation in two of the four proposed models, namely, LSTM and CNN combined with LSTM. Moreover, the single GRU model and the combination of the CNN with the GRU models exhibited good performance when using ADAM optimisation, as represented in Figure 5.8.

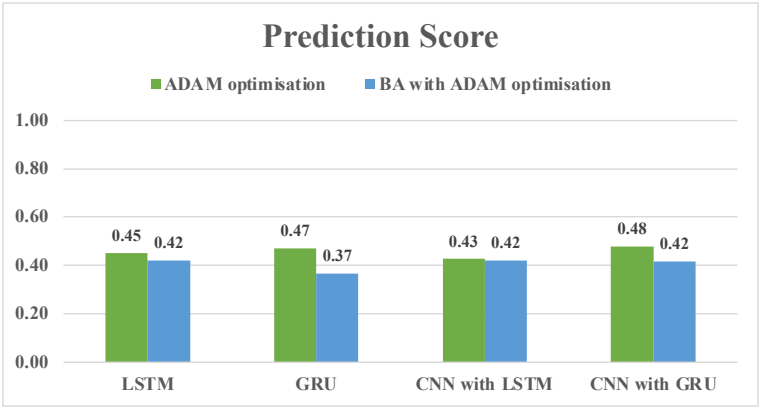


Figure 5.9 Comparison of prediction scores

Figure 5.9 presents an overall performance comparison using a prediction score. Training DL models with BA with ADAM optimisation was less effective than training with ADAM optimisation alone, particularly for the GRU model, which had the lowest score (0.37). While the scores of the proposed method do not meet this goal, it is apparent from the second condition that there is an opportunity for further development in combining BA with ADAM optimisation to train DL models for improved performance.

5.5.3 Comparison of RUL prediction results

Using the same dataset obtained from the FEMTO-ST Institute laboratory, this research compared the RUL predictions with the results from nine related publications, as shown in Table 5.3. Based on the prediction score, the best performance in this study was achieved by combining a CNN with a GRU using ADAM optimisation and BA_2 to set up hyperparameters, resulting in the highest score of 0.48. This model effectively predicts the RUL of bearings, with seven underestimated and four overestimated predictions. For six out of the eleven test sets, 54.55% of the prediction results were accounted for, yielding a percentage of prediction errors lower than ten.

Furthermore, the best-proposed model outperforms six of the related research studies, including the winner of the IEEE PHM 2012 competition [216], [217], [218], [90], [38] and [219], achieving superior performances of 57.70, 36.19, 12.84, 83.80, 26.29 and 10.29%, respectively. However, this model falls short when compared to three other publications, [112] [220] and [221], which attained scores of 0.54, 0.62 and 0.49, respectively.

Turning to the other proposed models that incorporate ADAM optimisation into the BA, the highest-performing models, including LSTM, GRU, and CNN combined with GRU, outperform four related models, namely, [216], [217], [90] and [38], achieving superior performances of 36.69, 18.05, 59.31 and 9.46%, respectively. Although these proposed models may not demonstrate exceptional performance, further development is needed to enhance their effectiveness.

Table 5.3 Comparison of prediction performance with related research

		Prediction Error (%)										Score
		B13	**B14**	**B15**	**B16**	**B17**	**B23**	**B24**	**B25**	**B26**	**B27**	**B33**	
[216]	The winner of IEEE PHM 2012 competition	37.00	80.00	9.00	-5.00	-2.00	64.00	10.00	-440.00	49.00	-317.00	90.00	**0.31**
[217]	Gaussian Process Regression (GPR) algorithm	-1.04	-20.94	-278.26	19.18	-7.13	10.49	51.80	28.80	-20.93	44.83	-3.66	**0.36**
[218]	Particle filtering-based prediction algorithm	-0.35	5.60	100.00	28.08	-19.55	-20.19	8.63	23.30	58.91	5.17	40.24	**0.43**
[90]	RNN-HI	43.28	67.55	-22.98	21.23	17.83	37.84	-19.42	54.37	-13.95	-55.17	3.66	**0.26**
[38]	ConvLSTM	54.73	38.69	-99.40	-120.07	70.65	75.53	19.81	8.20	17.87	1.69	2.93	**0.38**
[112]	CNN with GPR	1.05	20.35	11.18	34.93	229.19	57.24	-1.44	-0.07	-42.64	8.62	-1.22	**0.54**
[219]	RNN based on encoder–decoder framework	7.62	-157.71	-72.57	0.93	85.99	81.24	9.04	28.19	24.92	19.06	2.09	**0.44**
[220]	Integrated Laplacian-LSTM	-0.69	3.10	-2.42	-2.05	7.00	1.99	3.60	-19.40	-16.27	-5.17	-9.76	**0.62**

		Prediction Error (%)											Score
		B13	**B14**	**B15**	**B16**	**B17**	**B23**	**B24**	**B25**	**B26**	**B27**	**B33**	
[221]	MSWR-LRCN	8.90	9.31	-147.82	8.22	0.13	12.48	-14.39	-12.94	41.86	-10.34	-2.44	**0.49**
Proposed Models	LSTM (BA_2 Hyperparameter optimisation)	-27.33	40.11	-14.67	-5.19	-9.73	5.62	27.26	2.34	14.57	13.17	23.29	**0.45**
	GRU (BA_2 Hyperparameter optimisation)	-26.63	42.93	-12.75	-2.24	-9.35	3.03	25.03	-3.97	11.22	11.31	22.35	**0.47**
	CNN+LSTM (BA_2 Hyperparameter optimisation)	-26.91	42.54	-14.02	-3.40	-9.44	2.14	22.56	-13.60	7.53	9.23	23.44	**0.43**
	CNN+GRU (BA_2 Hyperparameter optimisation)	-25.25	46.53	-6.76	3.18	-8.87	2.04	22.22	-15.64	6.99	8.95	21.00	**0.48**
	LSTM (BA-ADAM optimisation)	-35.78	3.14	-71.28	-55.32	-12.31	2.21	22.35	-14.44	7.19	9.09	20.39	**0.42**
	GRU (BA-ADAM optimisation)	-32.64	38.36	-36.44	-32.73	-10.83	2.21	22.78	-11.15	7.86	9.38	22.64	**0.37**
	CNN+LSTM (BA-ADAM optimisation)	-35.78	3.14	-71.28	-55.32	-12.31	1.70	21.02	-21.65	5.15	7.94	21.15	**0.42**
	CNN+GRU (BA-ADAM optimisation)	-35.78	3.14	-71.28	-55.32	-12.31	2.08	22.35	-14.44	7.19	9.09	22.64	**0.42**

5.6 Summary

The integration of ADAM optimisation into the BA to enhance the search efficiency of the BA for optimising the learnable parameters of DL models is presented in this chapter. The proposed algorithm iteratively updated the weights and biases of the DL models, and the optimal parameters were extracted and assigned to the DL models to predict the RUL.

The results obtained show that the proposed integrated DL-BA system did not always perform better than architectures trained using conventional DL methods. This study acknowledges that integrating two distinct algorithms may lead to an overlapping process, potentially diminishing the overall efficiency of the integration. However, there is potential for further development and improvement to enhance the effectiveness of these methods in future models.

Chapter 6

Conclusion

6.1 Conclusion

This thesis focuses on techniques for predicting the remaining useful life (RUL) of components in a product to mitigate uncertainties related to the timing of product returns for remanufacturing. The primary objective of this work is to apply DL algorithms to predict the RUL of components. This prediction could assist in making well-informed decisions regarding the ideal timing for shipping products for remanufacturing.

Three studies are reported in the thesis.

The first study utilises four DL techniques—LSTM, a GRU, a CNN combined with LSTM, and a CNN combined with a GRU—to predict the RUL of bearings. These methodologies are selected because LSTM and GRU are variants of RNNs and are particularly suitable for handling time series data. Additionally, hybrid techniques, CNNs with LSTM and CNNs with GRUs, combine the potential of each algorithm. The combination techniques can effectively acquire and extract prominent features from unprocessed time series or sequential data while demonstrating proficiency in comprehending and analysing these features throughout temporal intervals. The overall performance of each DL approach is benchmarked using prediction scores. The highest score of 0.49 is achieved by the integrated CNN with LSTM model, followed by the CNN with GRU at 0.45, the LSTM at 0.43, and the GRU at 0.41. The findings of this study indicate that the utilisation of DL models, especially combination techniques,

which exhibit superior performance compared to individual DL models, demonstrates greater efficacy in predicting RUL.

The swarm-based optimisation algorithm (BA) is introduced for integration within DL networks. The second study employs the advanced variant of BA called BA_2 to optimise a set of DL hyperparameters. This methodology allows hyperparameters to be elaborated, and a time-consuming manual approach can be reduced. Among several models, the CNN with the GRU model is the most successful, attaining a maximum score of 0.48. The subsequent GRU and LSTM models exhibit scores of 0.47 and 0.45, respectively. On the other hand, the integration of the CNN with the LSTM model demonstrated inferior predictive ability, attaining a score of 0.43.

Additionally, the performances of prediction models with autotuning hyperparameter configurations are compared with those of human approaches involving trial and error and the Taguchi method. Autotuning for hyperparameter configurations demonstrated notable efficacy in most models, except for a coupling CNN with LSTM. However, the hyperparameter optimisation of BA_2 has limitations in attaining optimal performance across specific models. The benefits of this approach consistently exceed those of manual techniques in terms of simplicity in configuring and controlling hyperparameters.

Furthermore, the third study integrates BA with DL techniques to bypass the occurrence of local optima and accelerate the process of reaching global optima. The DL learnable parameters, including weights and biases, are updated by incorporating the ADAM optimisation algorithm into the BA. This integration enhances the search efficiency of the algorithm. The observed algorithm yields varying results, with the three

proposed models, namely, LSTM, CNN with LSTM and CNN with GRU, achieving the best score of 0.42, while the GRU model performs less optimally, with a score of 0.37.

Compared to using individual ADAM optimisation, combining BA with ADAM optimisation demonstrates reduced efficiency and yields worse results for all models. This study revealed that integrating two disparate algorithms may lead to overlapping responsibilities and duplication within the search procedure, potentially compromising the overall effectiveness of the integration. However, it is essential to note that the proposed models still outperform those of four related research studies using the same IEEE PHM 2012 Data Challenge dataset. While acknowledging the limitations of the algorithm presented, it is evident that there is potential for performance enhancement in future models through further development and improvement.

Thus, the aim and objectives of the research set out in Chapter 1 have been achieved. This thesis has shown that DL models can accurately predict the RUL of bearings in rotating machinery and that the most effective approach identified involves combining LSTM and CNN. The studies conducted in this research have also demonstrated the effectiveness of the Bees Algorithm as a tool for automatically configuring and tuning DL models.

6.2 Contributions

The contributions of this thesis are outlined below:

1) The integration of the principle of predictive maintenance (PdM) using DL techniques within the remanufacturing system can reduce the uncertainties associated with core returns. Accurate RUL prediction enables remanufacturing planners to determine the exact timing of product retrieval and establish a well-structured schedule for subsequent processes.

2) The implementation of BA for DL network configuration and hyperparameter tuning enhances prediction performance, fine-tunes hyperparameter values, and streamlines the process of DL configurations, leading to time savings compared to manual tuning.

3) The fusion of BA with the ADAM optimisation algorithm represents a novel approach that combines the advantages of both algorithms to mitigate the risk of becoming trapped in local optima and to improve DL performance.

6.3 Limitations of the study

Utilising DL and metaheuristic algorithms necessitates high-performance computing systems due to the inherent complexities and technical challenges in both fields. The adoption of graphics processing units (GPUs) has emerged as a pivotal technique in DL, resulting in substantial improvements in the computational efficiency of training and inference procedures. Like DL, BA frequently deals with extensive state spaces and employs complex search methods. Conducting computational tests is essential for parameter adjustment and evaluating algorithmic performance. Therefore, substantial computational resources are required for simulations and testing to facilitate this research.

A limitation of this study is associated with the computational capability of the University of Birmingham's supercomputers, known as BlueBEAR. These resources exhibit relatively slow computing performance when handling large datasets, which can extend the time required to complete numerous iterations. Furthermore, BlueBEAR has limited memory capacity, which can constrain certain experiments, particularly those involving training datasets using a combination of DL techniques. Additionally, unexpected shutdowns occur during the experiment using MATLAB programs without

prior notification. Consequently, the range of DL hyperparameter values and parameters in the BA, such as the number of iterations, number of scout bees, and population sizes, is restricted based on the computational capabilities available on BlueBEAR at the University of Birmingham.

6.4 Future work

Using DL techniques as a decision-making tool to predict the RUL of returned cores can reduce uncertainties related to the timing of returns upstream in the remanufacturing process. Future work plans to utilise DL techniques in remanufacturing as follows:

DL techniques can be used to effectively detect various conditions in returned cores during midstream manufacturing. These methods allow for the classification of the quality of returned parts before they enter the disassembly process, facilitating easier planning and scheduling for planners.

In the downstream process, DL techniques can predict the timing of the disassembly process, enabling better preparation for parts matching in the reassembly process. Reassembly planning can be executed with reduced uncertainty regarding the recovered parts, thereby mitigating the need for frequent replanning. Additionally, DL techniques can be applied to evaluate newly recovered parts before they are reintroduced to the market.

While integrating the BA with DL techniques has proven successful in network configurations, enhancing prediction performance by incorporating ADAM optimisation with BA has not yielded the expected results. A potential avenue for future work involves exploring other variants of BA, such as the two-parameter Bees Algorithm (BA_2) or the hybrid modified BA (MBA), in combination with alternative

adaptive learning rate algorithms such as the RMSProp algorithm and the AdaDelta algorithm. In this approach, DL learnable parameters are updated to avoid local optima and progress toward the global optimum.

References

[1] M. Matsumoto and S. Komatsu, “Demand forecasting for production planning in remanufacturing,” *The International Journal of Advanced Manufacturing Technology*, vol. 79, no. 1, pp. 161–175, 2015, doi: 10.1007/s00170-015-6787-x.

[2] W. Zhang, J. Wang, X. Liu, and S. Zhang, “A new uncertain remanufacturing scheduling model with rework risk using hybrid optimization algorithm,” *Environmental Science and Pollution Research*, vol. 30, no. 22, pp. 62744–62761, 2023, doi: 10.1007/s11356-023-26219-7.

[3] V. D. R. Guide, “Production planning and control for remanufacturing: Industry practice and research needs,” *Journal of Operations Management*, vol. 18, no. 4, pp. 467–483, 2000, doi: 10.1016/S0272-6963(00)00034-6.

[4] H. C. Fang, S. K. Ong, and A. Y. C. Nee, “Use of Embedded Smart Sensors in Products to Facilitate Remanufacturing,” in *Handbook of Manufacturing Engineering and Technology*, A. Y. C. Nee, Ed., London: Springer London, 2015, pp. 3265–3290. doi: 10.1007/978-1-4471-4670-4_85.

[5] J. Kurilova-Palisaitiene, E. Sundin, and B. Poksinska, “Remanufacturing challenges and possible lean improvements,” *J Clean Prod*, vol. 172, pp. 3225–3236, 2018, doi: 10.1016/j.jclepro.2017.11.023.

[6] Y. Ren, X. Lu, H. Guo, Z. Xie, H. Zhang, and C. Zhang, “A Review of Combinatorial Optimization Problems in Reverse Logistics and Remanufacturing for End-of-Life Products,” *Mathematics*, vol. 11, no. 2, p. 298, 2023, doi: 10.3390/math11020298.

[7] M. I. Mazhar, S. Kara, and H. Kaebernick, “Remaining life estimation of used components in consumer products: Life cycle data analysis by Weibull and artificial neural networks,” *Journal of Operations Management*, vol. 25, no. 6, pp. 1184–1193, 2007, doi: 10.1016/j.jom.2007.01.021.

[8] Y. Hu, S. Liu, H. Lu, and H. Zhang, “Remaining useful life assessment and its application in the decision for remanufacturing,” *Procedia CIRP*, vol. 15, pp. 212–217, 2014, doi: 10.1016/j.procir.2014.06.052.

[9] H. Y. Zhang H., Liu S., Lu H., Zhang Y., “Remanufacturing and Remaining Useful Life Assessment,” in *HandBook of Manufacturing Engineering and Technology*, 2015, pp. 3137–3192. doi: 10.1007/978-1-4471-4670-4.

[10] Y. Hu, S. Liu, and H. Zhang, “Remanufacturing decision based on RUL assessment,” in *Procedia CIRP*, Elsevier B.V., 2015, pp. 764–768. doi: 10.1016/j.procir.2015.01.027.

[11] X. Zhang, H. Zhang, Z. Jiang, and Y. Wang, “A decision-making approach for end-of-life strategies selection of used parts,” *The International Journal of Advanced Manufacturing Technology*, vol. 87, no. 5, pp. 1457–1464, 2016, doi: 10.1007/s00170-013-5234-0.

[12] R. F. Fofou, Z. Jiang, and Y. Wang, “A Review on the Lifecycle Strategies Enhancing Remanufacturing,” *Applied Sciences*, vol. 11, no. 13, 2021, doi: 10.3390/app11135937.

[13] F. Calabrese, A. Regattieri, M. Bortolini, M. Gamberi, and F. Pilati, “Predictive Maintenance: A Novel Framework for a Data-Driven, Semi-Supervised, and Partially Online Prognostic Health Management Application in Industries,” *Applied Sciences*, vol. 11, no. 8, 2021, doi: 10.3390/app11083380.

[14] M. Zhang *et al.*, “Predictive Maintenance for Remanufacturing Based on Hybrid-Driven Remaining Useful Life Prediction,” *Applied Sciences (Switzerland)*, vol. 12, no. 7, 2022, doi: 10.3390/app12073218.

[15] R. Zhao, R. Yan, Z. Chen, K. Mao, P. Wang, and R. X. Gao, “Deep learning and its applications to machine health monitoring,” *Mech Syst Signal Process*, vol. 115, pp. 213–237, Jan. 2019, doi: 10.1016/j.ymssp.2018.05.050.

[16] B. Rezaeianjouybari and Y. Shang, “Deep learning for prognostics and health management: State of the art, challenges, and opportunities,” *Measurement*, vol. 163, p. 107929, 2020, doi: https://doi.org/10.1016/j.measurement.2020.107929.

[17] O. Fink, Q. Wang, M. Svensén, P. Dersin, W.-J. Lee, and M. Ducoffe, “Potential, challenges and future directions for deep learning in prognostics and health management applications,” *Eng Appl Artif Intell*, vol. 92, p. 103678, 2020, doi: https://doi.org/10.1016/j.engappai.2020.103678.

[18] Y. Zhang and Y.-F. Li, “Prognostics and health management of Lithium-ion battery using deep learning methods: A review,” *Renewable and Sustainable Energy Reviews*, vol. 161, p. 112282, 2022, doi: https://doi.org/10.1016/j.rser.2022.112282.

[19] L. Alzubaidi *et al.*, *Review of deep learning: concepts, CNN architectures, challenges, applications, future directions*, vol. 8, no. 1. Springer International Publishing, 2021. doi: 10.1007/s13244-018-0639-9.

[20] I. H. Sarker, “Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions,” *SN Comput Sci*, vol. 2, no. 6, p. 420, 2021.

[21] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” *Insights Imaging*, vol. 9, no. 4, pp. 611–629, 2018, doi: 10.1007/s13244-018-0639-9.

[22] Y. Yoo, “Hyperparameter optimization of deep neural network using univariate dynamic encoding algorithm for searches,” *Knowl Based Syst*, vol. 178, pp. 74–83, 2019, doi: https://doi.org/10.1016/j.knosys.2019.04.019.

[23] J. Wu, S. Chen, and X. Liu, “Efficient hyperparameter optimization through model-based reinforcement learning,” *Neurocomputing*, vol. 409, pp. 381–393, 2020, doi: https://doi.org/10.1016/j.neucom.2020.06.064.

[24] R. Hossain and D. D. Timmer, “Machine learning model optimization with hyper parameter tuning approach,” *Global Journal of Computer Science and Technology*, vol. 21, no. 2, pp. 7–13, 2021.

[25] B. Bischl *et al.*, “Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges,” *Wiley Interdiscip Rev Data Min Knowl Discov*, vol. 13, no. 2, p. e1484, 2023.

[26] Z. Beheshti and S. M. Shamsuddin, “A review of population-based meta-heuristic algorithm,” *International Journal of Advances in Soft Computing and Its Applications*, vol. 5, pp. 1–35, 2013.

[27] D. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *International Conference on Learning Representations*, 2014.

[28] V. D. R. Guide, M. E. Krausb, and R. Srivastavac, “Scheduling policies for remanufacturing,” *Int J Prod Econ*, vol. 48, no. 2, pp. 187–204, 1997.

[29] N. Sakib and T. Wuest, “Challenges and Opportunities of Condition-based Predictive Maintenance: A Review,” *Procedia CIRP*, vol. 78, pp. 267–272, 2018, doi: 10.1016/j.procir.2018.08.318.

[30] F. Afrinaldi, Z. Liu, Taufik, H. C. Zhang, and A. Hasan, “The Advantages of Remanufacturing from the Perspective of Eco-efficiency Analysis: A Case Study,” *Procedia CIRP*, vol. 61, pp. 223–228, 2017, doi: 10.1016/j.procir.2016.11.161.

[31] N. C. Y. Yeo, H. Pepin, and S. S. Yang, “Revolutionizing Technology Adoption for the Remanufacturing Industry,” *Procedia CIRP*, vol. 61, pp. 17–21, 2017, doi: 10.1016/j.procir.2016.11.262.

[32] T. Sakao and E. Sundin, “How to Improve Remanufacturing? - A Systematic Analysis of Practices and Theories,” *Journal of Manufacturing Science and Engineering, Transactions of the ASME*, vol. 141, no. 2, pp. 1–13, 2019, doi: 10.1115/1.4041746.

[33] M. Lage and M. G. Filho, “Production planning and control for remanufacturing: Literature review and analysis,” *Production Planning and Control*, vol. 23, no. 6, pp. 419–435, 2012, doi: 10.1080/09537287.2011.561815.

[34] K. Meng, P. Lou, X. Peng, and V. Prybutok, “Quality-driven recovery decisions for used components in reverse logistics,” *Int J Prod Res*, vol. 55, no. 16, pp. 4712–4728, 2017, doi: 10.1080/00207543.2017.1287971.

[35] M. L. Bentaha, A. Voisin, and P. Marangé, “A decision tool for disassembly process planning under end-of-life product quality,” *Int J Prod Econ*, vol. 219, no. December 2018, pp. 386–401, 2020, doi: 10.1016/j.ijpe.2019.07.015.

[36] P. Y. Li X., Lu W.F., Zhai L., Er M.J., “Remaining Life Prediction of Cores Based on Data-driven and Physical Modeling Methods,” in *HandBook of Manufacturing Engineering and Technology*, 2015, pp. 3239–3290. doi: 10.1007/978-1-4471-4670-4.

[37] C. Okoh, R. Roy, J. Mehnen, and L. Redding, “Overview of Remaining Useful Life prediction techniques in Through-life Engineering Services,” in *Procedia CIRP*, Elsevier, 2014, pp. 158–163. doi: 10.1016/j.procir.2014.02.006.

[38] A. Z. Hinchi and M. Tkiouat, “Rolling element bearing remaining useful life estimation based on a convolutional long-short-Term memory network,” in *Procedia Computer Science*, Elsevier B.V., 2018, pp. 123–132. doi: 10.1016/j.procs.2018.01.106.

[39] O. Motaghare and A. S. Pillai, “Predictive Maintenance Architecture,” pp. 2–5.

[40] “BS EN 13306 : 2017 BSI Standards Publication Maintenance — Maintenance terminology,” 2017.

[41] E. I. Basri, I. H. A. Razak, H. Ab-Samat, and S. Kamaruddin, “Preventive maintenance (PM) planning: A review,” *J Qual Maint Eng*, vol. 23, no. 2, pp. 114–143, 2017, doi: 10.1108/JQME-04-2016-0014.

[42] H. N. Teixeira, I. Lopes, and A. C. Braga, “Condition-based maintenance implementation: A literature review,” *Procedia Manuf*, vol. 51, no. 2019, pp. 228–235, 2020, doi: 10.1016/j.promfg.2020.10.033.

[43] A. K. S. Jardine, D. Lin, and D. Banjevic, “A review on machinery diagnostics and prognostics implementing condition-based maintenance,” *Mech Syst Signal Process*, vol. 20, no. 7, pp. 1483–1510, 2006, doi: 10.1016/j.ymssp.2005.09.012.

[44] K. Efthymiou, N. Papakostas, D. Mourtzis, and G. Chryssolouris, “On a predictive maintenance platform for production systems,” *Procedia CIRP*, vol. 3, no. 1, pp. 221–226, 2012, doi: 10.1016/j.procir.2012.07.039.

[45] A. Jimenez-Cortadi, I. Irigoien, F. Boto, B. Sierra, and G. Rodriguez, “Predictive maintenance on the machining process and machine tool,” *Applied Sciences (Switzerland)*, vol. 10, no. 1, 2020, doi: 10.3390/app10010224.

[46] P. Nectoux *et al.*, “PRONOSTIA: An experimental platform for bearings accelerated degradation tests,” 2012.

[47] R. Gao *et al.*, “Cloud-enabled prognosis for manufacturing,” *CIRP Ann Manuf Technol*, vol. 64, no. 2, pp. 749–772, 2015, doi: 10.1016/j.cirp.2015.05.011.

[48] A. K. Mahamad, S. Saon, and T. Hiyama, “Predicting remaining useful life of rotating machinery based artificial neural network,” *Computers and Mathematics with Applications*, vol. 60, no. 4, pp. 1078–1087, Aug. 2010, doi: 10.1016/j.camwa.2010.03.065.

[49] H. Yan, J. Wan, C. Zhang, S. Tang, Q. Hua, and Z. Wang, “Industrial Big Data Analytics for Prediction of Remaining Useful Life Based on Deep Learning,” *IEEE Access*, vol. 6, pp. 17190–17197, Feb. 2018, doi: 10.1109/ACCESS.2018.2809681.

[50] X. Li, Q. Ding, and J. Q. Sun, “Remaining useful life estimation in prognostics using deep convolution neural networks,” *Reliab Eng Syst Saf*, vol. 172, pp. 1–11, Apr. 2018, doi: 10.1016/j.ress.2017.11.021.

[51] C. Zhang, P. Lim, A. K. Qin, and K. C. Tan, “Multiobjective Deep Belief Networks Ensemble for Remaining Useful Life Estimation in Prognostics,”

IEEE Trans Neural Netw Learn Syst, vol. 28, no. 10, pp. 2306–2318, Oct. 2017, doi: 10.1109/TNNLS.2016.2582798.

[52] Y. Wang, Y. Zhao, and S. Addepalli, "Remaining useful life prediction using deep learning approaches: A review," *Procedia Manuf*, vol. 49, no. 2019, pp. 81–88, 2020, doi: 10.1016/j.promfg.2020.06.015.

[53] L. Ren, J. Cui, Y. Sun, and X. Cheng, "Multi-bearing remaining useful life collaborative prediction: A deep learning approach," *J Manuf Syst*, vol. 43, pp. 248–256, Apr. 2017, doi: 10.1016/j.jmsy.2017.02.013.

[54] K. Medjaher, N. Zerhouni, and J. Baklouti, "Data-driven prognostics based on health indicator construction: Application to PRONOSTIA's data," *2013 European Control Conference, ECC 2013*, pp. 1451–1456, 2013, doi: 10.23919/ecc.2013.6669223.

[55] S. Selcuk, "Predictive maintenance, its implementation and latest trends," *Proc Inst Mech Eng B J Eng Manuf*, vol. 231, no. 9, pp. 1670–1679, 2017, doi: 10.1177/0954405415601640.

[56] S. Wu, N. Gebraeel, M. a Lawley, and Y. Yih, "A Neural Network Integrated Decision Support System for Condition-Based Optimal Predictive Maintenance Policy," *IEEE Trans Syst Man Cybern*, vol. 37, no. 2, pp. 226–236, 2007.

[57] X. Cao, P. Li, and S. Ming, "Remaining useful life prediction-based maintenance decision model for stochastic deterioration equipment under data-driven," *Sustainability (Switzerland)*, vol. 13, no. 15, 2021, doi: 10.3390/su13158548.

[58] X. Han, Z. Wang, M. Xie, Y. He, Y. Li, and W. Wang, "Remaining useful life prediction and predictive maintenance strategies for multi-state manufacturing

systems considering functional dependence," *Reliab Eng Syst Saf*, vol. 210, no. February, p. 107560, 2021, doi: 10.1016/j.ress.2021.107560.

[59] P. Coandă, M. Avram, and V. Constantin, "A state of the art of predictive maintenance techniques," *IOP Conf Ser Mater Sci Eng*, vol. 997, no. 1, 2020, doi: 10.1088/1757-899X/997/1/012039.

[60] T. Zonta, C. A. da Costa, R. da Rosa Righi, M. J. de Lima, E. S. da Trindade, and G. P. Li, "Predictive maintenance in the Industry 4.0: A systematic literature review," *Comput Ind Eng*, vol. 150, no. April 2019, p. 106889, 2020, doi: 10.1016/j.cie.2020.106889.

[61] S. Sankararaman and K. Goebel, "Why is the remaining useful life prediction uncertain?," *PHM 2013 - Proceedings of the Annual Conference of the Prognostics and Health Management Society 2013*, pp. 337–349, 2013.

[62] Y. Lei, N. Li, L. Guo, N. Li, T. Yan, and J. Lin, "Machinery health prognostics: A systematic review from data acquisition to RUL prediction," *Mech Syst Signal Process*, vol. 104, pp. 799–834, May 2018, doi: 10.1016/j.ymssp.2017.11.016.

[63] X. Li, W. Zhang, and Q. Ding, "Deep learning-based remaining useful life estimation of bearings using multi-scale feature extraction," *Reliab Eng Syst Saf*, vol. 182, pp. 208–218, Feb. 2019, doi: 10.1016/j.ress.2018.11.011.

[64] C. Chen, J. Shi, N. Lu, Z. H. Zhu, and B. Jiang, "Data-driven predictive maintenance strategy considering the uncertainty in remaining useful life prediction," *Neurocomputing*, vol. 494, pp. 79–88, 2022, doi: 10.1016/j.neucom.2022.04.055.

[65] J. Wang, G. Wen, S. Yang, and Y. Liu, “Remaining Useful Life Estimation in Prognostics Using Deep Bidirectional LSTM Neural Network,” *Proceedings - 2018 Prognostics and System Health Management Conference, PHM-Chongqing 2018*, pp. 1037–1042, 2019, doi: 10.1109/PHM-Chongqing.2018.00184.

[66] Y. Hu *et al.*, “A prediction method for the real-time remaining useful life of wind turbine bearings based on the Wiener process,” *Renew Energy*, vol. 127, pp. 452–460, 2018, doi: 10.1016/j.renene.2018.04.033.

[67] W.-A. Yang, M. Xiao, W. Zhou, Y. Guo, and W. Liao, “A Hybrid Prognostic Approach for Remaining Useful Life Prediction of Lithium-Ion Batteries,” *Shock and Vibration*, vol. 2016, pp. 1–15, Apr. 2016, doi: 10.1155/2016/3838765.

[68] J. Deutsch, M. He, and D. He, “Remaining Useful Life Prediction of Hybrid Ceramic Bearings Using an Integrated Deep Learning and Particle Filter Approach,” *Applied Sciences*, vol. 7, no. 7, p. 649, Jun. 2017, doi: 10.3390/app7070649.

[69] P. Wang, R. X. Gao, and R. Yan, “A deep learning-based approach to material removal rate prediction in polishing,” *CIRP Ann Manuf Technol*, vol. 66, no. 1, pp. 429–432, 2017, doi: 10.1016/j.cirp.2017.04.013.

[70] J. Zhang, P. Wang, R. Yan, and R. X. Gao, “Deep Learning for Improved System Remaining Life Prediction,” in *Procedia CIRP*, Elsevier B.V., 2018, pp. 1033–1038. doi: 10.1016/j.procir.2018.03.262.

[71] H. Yan, J. Wan, C. Zhang, S. Tang, Q. Hua, and Z. Wang, “Industrial Big Data Analytics for Prediction of Remaining Useful Life Based on Deep Learning,”

IEEE Access, vol. 6, pp. 17190–17197, 2018, doi: 10.1109/ACCESS.2018.2809681.

[72] S. Khan and T. Yairi, "A review on the application of deep learning in system health management," *Mech Syst Signal Process*, vol. 107, pp. 241–265, 2018, doi: 10.1016/j.ymssp.2017.11.024.

[73] L. Zhang, J. Lin, B. Liu, Z. Zhang, X. Yan, and M. Wei, "A Review on Deep Learning Applications in Prognostics and Health Management," *IEEE Access*, vol. 7, pp. 162415–162438, 2019, doi: 10.1109/ACCESS.2019.2950985.

[74] P. Lokhande and B. S. Tiple, "A step towards advanced machine learning approach: Deep Learning," in *International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS-2017)*, 2017, pp. 1112–1116.

[75] J. Wang, Y. Ma, L. Zhang, R. X. Gao, and D. Wu, "Deep learning for smart manufacturing: Methods and applications," *J Manuf Syst*, vol. 48, pp. 144–156, 2018, doi: 10.1016/j.jmsy.2018.01.003.

[76] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, Apr. 2017, doi: 10.1016/j.neucom.2016.12.038.

[77] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. 2017. doi: 10.1038/nmeth.3707.

[78] D. Weimer, B. Scholz-Reiter, and M. Shpitalni, "Design of deep convolutional neural network architectures for automated feature extraction in industrial inspection," *CIRP Ann Manuf Technol*, vol. 65, no. 1, pp. 417–420, 2016, doi: 10.1016/j.cirp.2016.04.072.

[79] L. Ren, Y. Sun, H. Wang, and L. Zhang, “Prediction of bearing remaining useful life with deep convolution neural network,” *IEEE Access*, vol. 6, pp. 13041–13049, Feb. 2018, doi: 10.1109/ACCESS.2018.2804930.

[80] L. Ren, Y. Sun, J. Cui, and L. Zhang, “Bearing remaining useful life prediction based on deep autoencoder and deep neural networks,” *J Manuf Syst*, vol. 48, pp. 71–77, Jul. 2018, doi: 10.1016/j.jmsy.2018.04.008.

[81] D. T. Hoang and H. J. Kang, “A survey on Deep Learning based bearing fault diagnosis,” *Neurocomputing*, vol. 335, pp. 327–335, Mar. 2019, doi: 10.1016/j.neucom.2018.06.078.

[82] C. Morariu and T. Borangiu, “Time series forecasting for dynamic scheduling of manufacturing processes,” in *2018 IEEE International Conference on Automation, Quality and Testing, Robotics, AQTR 2018 - THETA 21st Edition, Proceedings*, Institute of Electrical and Electronics Engineers Inc., Jul. 2018, pp. 1–6. doi: 10.1109/AQTR.2018.8402748.

[83] H. Shao, H. Jiang, X. Zhang, and M. Niu, “Rolling bearing fault diagnosis using an optimization deep belief network,” *Meas Sci Technol*, vol. 26, no. 11, Sep. 2015, doi: 10.1088/0957-0233/26/11/115002.

[84] S. Deng, Z. Cheng, C. Li, X. Yao, Z. Chen, and R. V. Sanchez, “Rolling bearing fault diagnosis based on deep boltzmann machines,” in *Proceedings of 2016 Prognostics and System Health Management Conference, PHM-Chengdu 2016*, Institute of Electrical and Electronics Engineers Inc., Jan. 2016. doi: 10.1109/PHM.2016.7819840.

[85] C. Lu, Z. Y. Wang, W. L. Qin, and J. Ma, “Fault diagnosis of rotary machinery components using a stacked denoising autoencoder-based health state

identification," *Signal Processing*, vol. 130, pp. 377–388, Jan. 2017, doi: 10.1016/j.sigpro.2016.07.028.

[86] W. Fuan, J. Hongkai, S. Haidong, D. Wenjing, and W. Shuaipeng, "An adaptive deep convolutional neural network for rolling bearing fault diagnosis," *Meas Sci Technol*, vol. 28, no. 9, Aug. 2017, doi: 10.1088/1361-6501/aa6e22.

[87] Z. Chen, S. Deng, X. Chen, C. Li, R. V. Sanchez, and H. Qin, "Deep neural networks-based rolling bearing fault diagnosis," *Microelectronics Reliability*, vol. 75, pp. 327–333, 2017, doi: 10.1016/j.microrel.2017.03.006.

[88] F. O. Heimes, "Recurrent neural networks for remaining useful life estimation," in *2008 International Conference on Prognostics and Health Management, PHM 2008*, 2008. doi: 10.1109/PHM.2008.4711422.

[89] A. Malhi, R. Yan, and R. X. Gao, "Prognosis of defect propagation based on recurrent neural networks," *IEEE Trans Instrum Meas*, vol. 60, no. 3, pp. 703–711, 2011, doi: 10.1109/TIM.2010.2078296.

[90] L. Guo, N. Li, F. Jia, Y. Lei, and J. Lin, "A recurrent neural network based health indicator for remaining useful life prediction of bearings," *Neurocomputing*, vol. 240, pp. 98–109, 2017, doi: 10.1016/j.neucom.2017.02.045.

[91] S. Zheng, K. Ristovski, A. Farahat, and C. Gupta, "Long Short-Term Memory Network for Remaining Useful Life estimation," in *2017 IEEE International Conference on Prognostics and Health Management, ICPHM 2017*, Institute of Electrical and Electronics Engineers Inc., Jul. 2017, pp. 88–95. doi: 10.1109/ICPHM.2017.7998311.

[92] C. S. Hsu and J. R. Jiang, “Remaining useful life estimation using long short-term memory deep learning,” in *Proceedings of 4th IEEE International Conference on Applied System Innovation 2018, ICASI 2018*, Institute of Electrical and Electronics Engineers Inc., Jun. 2018, pp. 58–61. doi: 10.1109/ICASI.2018.8394326.

[93] Z. Chen, M. Wu, R. Zhao, F. Guretno, R. Yan, and X. Li, “Machine Remaining Useful Life Prediction via an Attention-Based Deep Learning Approach,” *IEEE Transactions on Industrial Electronics*, vol. 68, no. 3, pp. 2521–2531, 2021, doi: 10.1109/TIE.2020.2972443.

[94] M. Yuan, Y. Wu, and L. Lin, “Fault diagnosis and remaining useful life estimation of aero engine using LSTM neural network,” in *2016 IEEE/CSAA International Conference on Aircraft Utility System (AUS)*, 2016, pp. 135–140.

[95] Y. Wu, M. Yuan, S. Dong, L. Lin, and Y. Liu, “Remaining useful life estimation of engineered systems using vanilla LSTM neural networks,” *Neurocomputing*, vol. 275, pp. 167–179, Jan. 2018, doi: 10.1016/j.neucom.2017.05.063.

[96] O. Aydin and S. Guldamlasioglu, “Using LSTM Networks to Predict Engine Condition on Large Scale Data Processing Framework,” in *2017 4th International Conference on Electrical and Electronics Engineering Using*, 2017, pp. 281–285.

[97] Y. Zhang, P. Hutchinson, N. A. J. Lieven, and J. Nunez-Yanez, “Remaining useful life estimation using long short-term memory neural networks and deep fusion,” *IEEE Access*, vol. 8, pp. 19033–19045, 2020, doi: 10.1109/ACCESS.2020.2966827.

[98] J. Wu, K. Hu, Y. Cheng, H. Zhu, X. Shao, and Y. Wang, “Data-driven remaining useful life prediction via multiple sensor signals and deep long short-term memory neural network,” *ISA Trans*, vol. 97, pp. 241–250, 2020, doi: 10.1016/j.isatra.2019.07.004.

[99] F. Wang, X. Liu, G. Deng, X. Yu, H. Li, and Q. Han, “Remaining Life Prediction Method for Rolling Bearing Based on the Long Short-Term Memory Network,” *Neural Process Lett*, vol. 50, no. 3, pp. 2437–2454, 2019, doi: 10.1007/s11063-019-10016-w.

[100] Z. H. Liu *et al.*, “A Regularized LSTM Method for Predicting Remaining Useful Life of Rolling Bearings,” *International Journal of Automation and Computing*, vol. 18, no. 4, pp. 581–593, 2021, doi: 10.1007/s11633-020-1276-6.

[101] J. Yang, Y. Peng, J. Xie, and P. Wang, “Remaining Useful Life Prediction Method for Bearings Based on LSTM with Uncertainty Quantification,” *Sensors*, vol. 22, no. 12, 2022, doi: 10.3390/s22124549.

[102] Y. Zhou, Y. Huang, J. Pang, and K. Wang, “Remaining useful life prediction for supercapacitor based on long short-term memory neural network,” *J Power Sources*, vol. 440, no. August, p. 227149, 2019, doi: 10.1016/j.jpowsour.2019.227149.

[103] S. Xiang, Y. Qin, C. Zhu, Y. Wang, and H. Chen, “Long short-term memory neural network with weight amplification and its application into gear remaining useful life prediction,” *Eng Appl Artif Intell*, vol. 91, no. February, p. 103587, 2020, doi: 10.1016/j.engappai.2020.103587.

[104] S. Xiang, Y. Qin, C. Zhu, Y. Wang, and H. Chen, “LSTM networks based on attention ordered neurons for gear remaining life prediction,” *ISA Trans*, vol. 106, pp. 343–354, 2020, doi: 10.1016/j.isatra.2020.06.023.

[105] J. Zhang, P. Wang, R. Yan, and R. X. Gao, “Long short-term memory for machine remaining life prediction,” *J Manuf Syst*, vol. 48, pp. 78–86, Jul. 2018, doi: 10.1016/j.jmsy.2018.05.011.

[106] A. Zhang *et al.*, “Transfer Learning with Deep Recurrent Neural Networks for Remaining Useful Life Estimation,” *Applied Sciences*, vol. 8, no. 12, p. 2416, Nov. 2018, doi: 10.3390/app8122416.

[107] G. S. Babu, P. Zhao, and X.-L. Li, “Deep Convolutional Neural Network Based Regression Approach for Estimation of Remaining Useful Life,” in *Database Systems for Advanced Applications: 21st International Conference, DASFAA 2016 Dallas, TX, USA, April 16–19, 2016 Proceedings, Part I*, Springer Verlag, 2016, pp. 214–228. doi: 10.1007/978-3-319-32025-0.

[108] J. Zhu, N. Chen, and W. Peng, “Estimation of Bearing Remaining Useful Life Based on Multiscale Convolutional Neural Network,” *IEEE Transactions on Industrial Electronics*, vol. 66, no. 4, pp. 3208–3216, Apr. 2019, doi: 10.1109/TIE.2018.2844856.

[109] J. R. Jiang and C. K. Kuo, “Enhancing convolutional neural network deep learning for remaining useful life estimation in smart factory applications,” in *Proceedings of the 2017 IEEE International Conference on Information, Communication and Engineering: Information and Innovation for Modern Technology, ICICE 2017*, Institute of Electrical and Electronics Engineers Inc., Oct. 2017, pp. 120–123. doi: 10.1109/ICICE.2017.8478928.

[110] Q. Yao, T. Yang, Z. Liu, and Z. Zheng, “Remaining useful life estimation by empirical mode decomposition and ensemble deep convolution neural networks,” *2019 IEEE International Conference on Prognostics and Health Management, ICPHM*, pp. 1–6, 2019, doi: 10.1109/ICPHM.2019.8819373.

[111] D. Verstraete, A. Ferrada, E. L. Droguett, V. Meruane, and M. Modarres, “Deep learning enabled fault diagnosis using time-frequency image analysis of rolling element bearings,” *Shock and Vibration*, vol. 2017, 2017, doi: 10.1155/2017/5067651.

[112] Y. Yoo and J. G. Baek, “A novel image feature for the remaining useful lifetime prediction of bearings based on continuous wavelet transform and convolutional neural network,” *Applied Sciences (Switzerland)*, vol. 8, no. 7, 2018, doi: 10.3390/app8071102.

[113] W. Zhang, C. Li, G. Peng, Y. Chen, and Z. Zhang, “A deep convolutional neural network with new training methods for bearing fault diagnosis under noisy environment and different working load,” *Mech Syst Signal Process*, vol. 100, pp. 439–453, 2018, doi: 10.1016/j.ymssp.2017.06.022.

[114] G. Li, C. Deng, J. Wu, X. Xu, X. Shao, and Y. Wang, “Sensor Data-Driven Bearing Fault Diagnosis Based on Deep Convolutional Neural Networks and S-Transform,” *Sensors (Basel)*, vol. 19, no. 12, 2019, doi: 10.3390/s19122750.

[115] C. W. Yeh and R. Chen, “Using convolutional neural network for vibration fault diagnosis monitoring in machinery,” *Proceedings of the 2018 IEEE International Conference on Advanced Manufacturing, ICAM 2018*, pp. 246–249, 2019, doi: 10.1109/AMCON.2018.8614967.

[116] S. Suh, J. Jang, S. Won, M. S. Jha, and Y. O. Lee, “Supervised health stage prediction using convolutional neural networks for bearing wear,” *Sensors (Switzerland)*, vol. 20, no. 20, pp. 1–19, 2020, doi: 10.3390/s20205846.

[117] X. Wang, D. Mao, and X. Li, “Bearing fault diagnosis based on vibro-acoustic data fusion and 1D-CNN network,” *Measurement (Lond)*, vol. 173, no. June 2020, p. 108518, 2021, doi: 10.1016/j.measurement.2020.108518.

[118] J. Deutsch and D. He, “Using Deep Learning Based Approaches for Bearing Remaining Useful Life Prediction,” in *ANNUAL CONFERENCE OF THE PROGNOSTICS AND HEALTH MANAGEMENT SOCIETY 2016*, 2016, pp. 1–7.

[119] J. Deutsch and D. He, “Using Deep Learning-Based Approach to Predict Remaining Useful Life of Rotating Components,” *IEEE Trans Syst Man Cybern Syst*, vol. 48, no. 1, pp. 11–20, May 2018, doi: 10.1109/TSMC.2017.2697842.

[120] G. Zhao, G. Zhang, Y. Liu, B. Zhang, and C. Hu, “Lithium-ion battery remaining useful life prediction with Deep Belief Network and Relevance Vector Machine,” in *2017 IEEE International Conference on Prognostics and Health Management, ICPHM 2017*, Institute of Electrical and Electronics Engineers Inc., Jul. 2017, pp. 7–13. doi: 10.1109/ICPHM.2017.7998298.

[121] A. Listou Ellefsen, E. Bjørlykhaug, V. Æsøy, S. Ushakov, and H. Zhang, “Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture,” *Reliab Eng Syst Saf*, vol. 183, pp. 240–251, Mar. 2019, doi: 10.1016/j.ress.2018.11.027.

[122] M. Xia, T. Li, T. Shu, J. Wan, C. W. de silva, and Z. Wang, “A Two-Stage Approach for the Remaining Useful Life Prediction of Bearings using Deep

Neural Networks," *IEEE Trans Industr Inform*, vol. 15, no. 6, pp. 3703–3711, Sep. 2019, doi: 10.1109/TII.2018.2868687.

[123] C. Sun, M. Ma, Z. Zhao, S. Tian, R. Yan, and X. Chen, "Deep Transfer Learning Based on Sparse Autoencoder for Remaining Useful Life Prediction of Tool in Manufacturing," *IEEE Trans Industr Inform*, vol. 15, no. 4, pp. 2416–2425, Apr. 2019, doi: 10.1109/TII.2018.2881543.

[124] G. Tang, Y. Zhou, H. Wang, and G. Li, "Prediction of bearing performance degradation with bottleneck feature based on LSTM network," *I2MTC 2018 - 2018 IEEE International Instrumentation and Measurement Technology Conference: Discovering New Horizons in Instrumentation and Measurement, Proceedings*, pp. 1–6, 2018, doi: 10.1109/I2MTC.2018.8409564.

[125] Y. Song, G. Shi, L. Chen, X. Huang, and T. Xia, "Remaining Useful Life Prediction of Turbofan Engine Using Hybrid Model Based on Autoencoder and Bidirectional Long Short-Term Memory," *J Shanghai Jiaotong Univ Sci*, vol. 23, pp. 85–94, 2018, doi: 10.1007/s12204-018-2027-5.

[126] D. Yao, B. Li, H. Liu, J. Yang, and L. Jia, "Remaining useful life prediction of roller bearings based on improved 1D-CNN and simple recurrent unit," *Measurement (Lond)*, vol. 175, no. January, p. 109166, 2021, doi: 10.1016/j.measurement.2021.109166.

[127] W. Mao, J. He, J. Tang, and Y. Li, "Predicting remaining useful life of rolling bearings based on deep feature representation and long short-term memory neural network," *Advances in Mechanical Engineering*, vol. 10, no. 12, pp. 1–18, Dec. 2018, doi: 10.1177/1687814018817184.

[128] J. Niu, C. Liu, L. Zhang, and Y. Liao, “Remaining Useful Life Prediction of Machining Tools by 1D-CNN LSTM Network,” *2019 IEEE Symposium Series on Computational Intelligence, SSCI 2019*, pp. 1056–1063, 2019, doi: 10.1109/SSCI44817.2019.9002993.

[129] A. Al-Dulaimi, S. Zabihi, A. Asif, and A. Mohammadi, “A multimodal and hybrid deep neural network model for Remaining Useful Life estimation,” *Comput Ind*, vol. 108, pp. 186–196, 2019, doi: 10.1016/j.compind.2019.02.004.

[130] J. Li, X. Li, and D. He, “A Directed Acyclic Graph Network Combined With CNN and LSTM for Remaining Useful Life Prediction,” *IEEE Access*, vol. 7, pp. 75464–75475, 2019, doi: 10.1109/ACCESS.2019.2919566.

[131] H. Li, Z. Wang, and Z. Li, “An enhanced CNN-LSTM remaining useful life prediction model for aircraft engine with attention mechanism,” *PeerJ Comput Sci*, vol. 8, pp. 1–19, 2022, doi: 10.7717/PEERJ-CS.1084.

[132] H. Mo, F. Lucca, J. Malacarne, and G. Iacca, “Multi-Head CNN-LSTM with Prediction Error Analysis for Remaining Useful Life Prediction,” *Conference of Open Innovation Association, FRUCT*, vol. 2020-Septe, pp. 164–171, 2020, doi: 10.23919/FRUCT49677.2020.9211058.

[133] Q. An, Z. Tao, X. Xu, M. El Mansori, and M. Chen, “A data-driven model for milling tool remaining useful life prediction with convolutional and stacked LSTM network,” *Measurement (Lond)*, vol. 154, p. 107461, 2020, doi: 10.1016/j.measurement.2019.107461.

[134] K. Abdelli, H. Grießer, and S. Pachnicke, “A Hybrid CNN-LSTM Approach for Laser Remaining Useful Life Prediction,” *2021 Opto-Electronics and*

Communications Conference, OECC 2021, no. July, pp. 1–4, 2021, doi: 10.1364/oecc.2021.s3d.3.

[135] X. Zhang, X. Lu, W. Li, and S. Wang, "Prediction of the remaining useful life of cutting tool using the Hurst exponent and CNN-LSTM," *International Journal of Advanced Manufacturing Technology*, vol. 112, no. 7–8, pp. 2277–2299, 2021, doi: 10.1007/s00170-020-06447-8.

[136] D. Li *et al.*, "Remaining Useful Life Prediction of Lithium Battery Based on Sequential CNN–LSTM Method," *Journal of Electrochemical Energy Conversion and Storage*, vol. 154, no. 4, pp. 371–383, 2022, doi: 10.1177/00202940221103622.

[137] A. Rastegarpanah, Y. Wang, and R. Stolkin, "Predicting the Remaining Life of Lithium-ion Batteries Using a CNN-LSTM Model," *2022 8th International Conference on Mechatronics and Robotics Engineering, ICMRE 2022*, pp. 73–78, 2022, doi: 10.1109/ICMRE54455.2022.9734081.

[138] D. Chen, X. Zheng, C. Chen, and W. Zhao, "Remaining useful life prediction of the lithium-ion battery based on CNN-LSTM fusion model and grey relational analysis," *Electronic Research Archive*, vol. 31, no. 2, pp. 633–655, 2022, doi: 10.3934/ERA.2023031.

[139] J.-R. Jiang, J.-E. Lee, and Y.-M. Zeng, "Time Series Multiple Channel Convolutional Neural Network with Attention-Based Long Short-Term Memory for Predicting Bearing Remaining Useful Life," *Sensors*, vol. 20(1), no. 166, pp. 1–19, 2020.

[140] T. Xia, Y. Song, Y. Zheng, E. Pan, and L. Xi, "An ensemble framework based on convolutional bi-directional LSTM with multiple time windows for

remaining useful life estimation," *Comput Ind*, vol. 115, p. 103182, 2020, doi: https://doi.org/10.1016/j.compind.2019.103182.

[141] D. Gao, X. Liu, Z. Zhu, and Q. Yang, "A hybrid CNN-BiLSTM approach for remaining useful life prediction of EVs lithium-Ion battery," *Measurement and Control (United Kingdom)*, vol. 56, pp. 371–383, 2022, doi: 10.1177/00202940221103622.

[142] B. Zraibi, C. Okar, H. Chaoui, and M. Mansouri, "Remaining Useful Life Assessment for Lithium-Ion Batteries Using CNN-LSTM-DNN Hybrid Method," *IEEE Trans Veh Technol*, vol. 70, no. 5, pp. 4252–4261, 2021, doi: 10.1109/TVT.2021.3071622.

[143] L. Ren, J. Dong, X. Wang, Z. Meng, L. Zhao, and M. J. Deen, "A Data-Driven Auto-CNN-LSTM Prediction Model for Lithium-Ion Battery Remaining Useful Life," *IEEE Trans Industr Inform*, vol. 17, no. 5, pp. 3478–3487, 2021, doi: 10.1109/TII.2020.3008223.

[144] B. Z. Aufa, S. Suyanto, and A. Arifianto, "Hyperparameter Setting of LSTM-based Language Model using Grey Wolf Optimizer," *2020 International Conference on Data Science and Its Applications, ICoDSA 2020*, 2020, doi: 10.1109/ICoDSA50139.2020.9213031.

[145] M. Kaveh and M. S. Mesgari, "Application of Meta-Heuristic Algorithms for Training Neural Networks and Deep Learning Architectures: A Comprehensive Review," *Neural Process Lett*, 2022, doi: 10.1007/s11063-022-11055-6.

[146] J. L. Olmo, J. R. Romero, and S. Ventura, "Swarm-based metaheuristics in automatic programming: A survey," *Wiley Interdiscip Rev Data Min Knowl Discov*, vol. 4, no. 6, pp. 445–469, 2014, doi: 10.1002/widm.1138.

[147] B. Velusamy and S. C. Pushpan, “A review on swarm intelligence based routing approaches,” *International Journal of Engineering and Technology Innovation*, vol. 9, no. 3, pp. 182–195, 2019.

[148] S. Mishra, R. Sagban, A. Yakoob, and N. Gandhi, “Swarm intelligence in anomaly detection systems: an overview,” *International Journal of Computers and Applications*, vol. 43, no. 2, pp. 109–118, 2021, doi: 10.1080/1206212X.2018.1521895.

[149] P. R. Lorenzo, J. Nalepa, M. Kawulok, L. S. Ramos, and J. R. Pastor, “Particle swarm optimization for hyper-parameter selection in deep neural networks,” *GECCO 2017 - Proceedings of the 2017 Genetic and Evolutionary Computation Conference*, pp. 481–488, 2017, doi: 10.1145/3071178.3071208.

[150] T. Y. Kim and S. B. Cho, “Particle Swarm Optimization-based CNN-LSTM Networks for Forecasting Energy Consumption,” *2019 IEEE Congress on Evolutionary Computation, CEC 2019 - Proceedings*, pp. 1510–1516, 2019, doi: 10.1109/CEC.2019.8789968.

[151] X. Song *et al.*, “Time-series well performance prediction based on Long Short-Term Memory (LSTM) neural network model,” *J Pet Sci Eng*, vol. 186, no. November 2019, p. 106682, 2020, doi: 10.1016/j.petrol.2019.106682.

[152] T. Cuong-Le, H.-L. Minh, T. Sang-To, S. Khatir, S. Mirjalili, and M. Abdel Wahab, “A novel version of Grey Wolf Optimizer based on a balance function and its application for hyperparameters optimization in Deep neural network (DNN) for Structural damage identification,” *Eng Fail Anal*, vol. 142, no. June, p. 106829, 2022, doi: 10.1016/j.engfailanal.2022.106829.

[153] C. Lin, K. Weng, Y. Lin, T. Zhang, Q. He, and Y. Su, “Time Series Prediction of Dam Deformation Using a Hybrid STL–CNN–GRU Model Based on Sparrow Search Algorithm Optimization,” *Applied Sciences (Switzerland)*, vol. 12, no. 23, 2022, doi: 10.3390/app122311951.

[154] X. Li, X. Ma, F. Xiao, C. Xiao, F. Wang, and S. Zhang, “Time-series production forecasting method based on the integration of Bidirectional Gated Recurrent Unit (Bi-GRU) network and Sparrow Search Algorithm (SSA),” *J Pet Sci Eng*, vol. 208, no. PA, p. 109309, 2022, doi: 10.1016/j.petrol.2021.109309.

[155] Y. Fan, Y. Zhang, B. Guo, X. Luo, Q. Peng, and Z. Jin, “A Hybrid Sparrow Search Algorithm of the Hyperparameter Optimization in Deep Learning,” *Mathematics*, vol. 10, no. 16, 2022, doi: 10.3390/math10163019.

[156] A. Gaspar, D. Oliva, E. Cuevas, D. Zaldívar, M. Pérez, and G. Pajares, “Hyperparameter Optimization in a Convolutional Neural Network Using Metaheuristic Algorithms,” *Studies in Computational Intelligence*, vol. 967, no. July, pp. 37–59, 2021, doi: 10.1007/978-3-030-70542-8_2.

[157] S. Zeybek, D. T. Pham, E. Koç, and A. Seçer, “An improved bees algorithm for training deep recurrent networks for sentiment classification,” *Symmetry (Basel)*, vol. 13, no. 8, pp. 1–26, 2021, doi: 10.3390/sym13081347.

[158] S. Zeybek, “Prediction of the Remaining Useful Life of Engines for Remanufacturing Using a Semi-supervised Deep Learning Model Trained by the Bees Algorithm,” in *Intelligent Production and Manufacturing Optimisation---The Bees Algorithm Approach*, D. T. Pham and N. Hartono,

Eds., Cham: Springer International Publishing, 2023, pp. 383–397. doi: 10.1007/978-3-031-14537-7_21.

[159] N. M. H. Alamri, M. Packianather, and S. Bigot, “Deep Learning: Parameter Optimization Using Proposed Novel Hybrid Bees Bayesian Convolutional Neural Network,” *Applied Artificial Intelligence*, vol. 36, no. 1, 2022, doi: 10.1080/08839514.2022.2031815.

[160] N. M. H. Alamri, M. Packianather, and S. Bigot, “Optimizing the Parameters of Long Short-Term Memory Networks Using the Bees Algorithm,” *Applied Sciences (Switzerland)*, vol. 13, no. 4, 2023, doi: 10.3390/app13042536.

[161] J. L. Berral-Garcia, “When and How to Apply Statistics, Machine Learning and Deep Learning Techniques,” *International Conference on Transparent Optical Networks*, pp. 1–4, 2018, doi: 10.1109/ICTON.2018.8473910.

[162] M. R. Costa-jussà, A. Allauzen, L. Barrault, K. Cho, and H. Schwenk, “Introduction to the special issue on deep learning approaches for machine translation,” *Comput Speech Lang*, vol. 46, pp. 367–373, 2017, doi: 10.1016/j.csl.2017.03.001.

[163] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, and R. Tachibana, “Deep Reinforcement Learning for High Precision Assembly Tasks,” *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 819–825, 2017, doi: 10.1109/IROS.2017.8202244.

[164] T. Ren, Y. Dong, D. Wu, and K. Chen, “Learning-Based Variable Compliance Control for Robotic Assembly,” *J Mech Robot*, vol. 10, no. 6, pp. 1–10, 2018, doi: 10.1115/1.4041331.

[165] Q. Lin, Z. H. Dai, and B. Meng, “Fault detection and experiment research of remanufacturing automatic transmission based on BP neural network,” in *2011 International Conference on Electrical and Control Engineering, ICECE 2011 - Proceedings*, 2011, pp. 191–194. doi: 10.1109/ICECENG.2011.6057130.

[166] Y. Zheng, H. Mamledesai, H. Imam, and R. Ahmad, “A novel deep learning-based automatic damage detection and localization method for remanufacturing/repair,” *Comput Aided Des Appl*, vol. 18, no. 6, pp. 1359–1372, 2021, doi: 10.14733/cadaps.2021.1359-1372.

[167] M. Schlüter *et al.*, “AI-enhanced Identification, Inspection and Sorting for Reverse Logistics in Remanufacturing,” *Procedia CIRP*, vol. 98, pp. 300–305, 2021, doi: 10.1016/j.procir.2021.01.107.

[168] M. Schlüter, C. Niebuhr, J. Lehr, and J. Krüger, “Vision-based Identification Service for Remanufacturing Sorting,” in *Procedia Manufacturing*, Elsevier B.V., 2018, pp. 384–391. doi: 10.1016/j.promfg.2018.02.135.

[169] C. E. Nwankpa, S. C. Eze, and W. Lijomah, “Deep Learning Based Visual Automated Sorting System for Remanufacturing,” *IEEE Green Technologies Conference*, vol. 2020-April, pp. 196–198, 2020, doi: 10.1109/GreenTech46478.2020.9289823.

[170] C. Nwankpa, S. Eze, W. Ijomah, A. Gachagan, and S. Marshall, “Achieving remanufacturing inspection using deep learning,” *Journal of Remanufacturing*, vol. 11, no. 2, pp. 89–105, 2021, doi: 10.1007/s13243-020-00093-9.

[171] L. Wang, X. Xia, J. Cao, and X. Liu, “Modeling and predicting remanufacturing time of equipment using deep belief networks,” *Cluster Comput*, pp. 1–12, Dec. 2017, doi: 10.1007/s10586-017-1430-2.

[172] Y. Wang, S. Wang, B. Yang, L. Zhu, and F. Liu, “Big data driven Hierarchical Digital Twin Predictive Remanufacturing paradigm: Architecture, control mechanism, application scenario and benefits,” *J Clean Prod*, vol. 248, p. 119299, 2020, doi: 10.1016/j.jclepro.2019.119299.

[173] W. Pan and L. Miao, “Dynamics and risk assessment of a remanufacturing closed-loop supply chain system using the internet of things and neural network approach,” *Journal of Supercomputing*, vol. 79, no. 4, pp. 3878–3901, 2022, doi: 10.1007/s11227-022-04727-6.

[174] S. Moon *et al.*, “Remanufacturing Decision-Making for Gas Insulated Switchgear with Remaining Useful Life Prediction,” *Sustainability (Switzerland)*, vol. 14, no. 19, pp. 1–13, 2022, doi: 10.3390/su141912357.

[175] IEEE, “IEEE PHM 2012 Prognostic challenge - Outline , Experiments , Scoring of results , Winners,” *IEEE PHM*, pp. 1–11, 2012.

[176] P. Nectoux *et al.*, “PRONOSTIA : An experimental platform for bearings accelerated degradation tests.,” *IEEE International Conference on Prognostics and Health Management, PHM’12*, no. June, pp. 1–8, 2012, [Online]. Available: http://hal-obspm.ccsd.cnrs.fr/UNIV-BM/hal-00719503

[177] Mathworks, “Sequence-to-Sequence Classification Using Deep Learning.” Accessed: Oct. 05, 2020. [Online]. Available: https://uk.mathworks.com/help/deeplearning/ug/sequence-to-sequence-classification-using-deep-learning.html

[178] J. Brownlee, *Deep learning for time series forecasting: predict the future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery, 2018.

[179] J. Brownlee, *Long Short-term Memory Networks with Python: Develop Sequence Prediction Models with Deep Learning*. Jason Brownlee, 2017. [Online]. Available: https://books.google.co.uk/books?id=ONpdswEACAAJ

[180] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning Book*. 2017. [Online]. Available: http://www.deeplearningbook.org/

[181] K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks," *ArXiv e-prints*, 2015.

[182] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *Proceedings of 2017 International Conference on Engineering and Technology, ICET 2017*, IEEE, 2017, pp. 1–6. doi: 10.1109/ICEngTechnol.2017.8308186.

[183] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[184] C. Olah, "Understanding LSTM Networks [Blog]," Web Page. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[185] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[186] W. Xu, H. Miao, Z. Zhao, J. Liu, C. Sun, and R. Yan, "Multi-Scale Convolutional Gated Recurrent Unit Networks for Tool Wear Prediction in Smart Manufacturing," *Chinese Journal of Mechanical Engineering (English Edition)*, vol. 34, no. 1, 2021, doi: 10.1186/s10033-021-00565-4.

[187] R. El Shawi, M. Maher, and S. Sakr, "Automated Machine Learning: State-of-The-Art and Open Challenges," *ArXiv*, vol. abs/1906.0, 2019.

[188] T. Yu and H. Zhu, “Hyper-Parameter Optimization: A Review of Algorithms and Applications,” pp. 1–56, 2020, [Online]. Available: http://arxiv.org/abs/2003.05689

[189] H. J. P. Weerts, A. C. Mueller, and J. Vanschoren, “Importance of tuning hyperparameters of machine learning algorithms,” *arXiv preprint arXiv:2007.07588*, 2020.

[190] L. Yang and A. Shami, “On hyperparameter optimization of machine learning algorithms: Theory and practice,” *Neurocomputing*, vol. 415, pp. 295–316, 2020, doi: 10.1016/j.neucom.2020.07.061.

[191] M. Tjantelé, “Parameter design using the Taguchi methodology,” *Microelectron Eng*, vol. 10, no. 3–4, pp. 277–286, 1991.

[192] R. K. Roy, *A primer on the Taguchi method*. Society of Manufacturing Engineers, 2010.

[193] S. Kaur, H. Aggarwal, and R. Rani, “Hyper-parameter optimization of deep learning model for prediction of Parkinson’s disease,” *Mach Vis Appl*, vol. 31, no. 5, pp. 1–15, 2020, doi: 10.1007/s00138-020-01078-1.

[194] J. Wu, X. Y. Chen, H. Zhang, L. D. Xiong, H. Lei, and S. H. Deng, “Hyperparameter optimization for machine learning models based on Bayesian optimization,” *Journal of Electronic Science and Technology*, vol. 17, no. 1, pp. 26–40, 2019, doi: 10.11989/JEST.1674-862X.80904120.

[195] D. Kong, S. Wang, and P. Ping, “State-of-health estimation and remaining useful life for lithium-ion battery based on deep learning with Bayesian hyperparameter optimization,” *Int J Energy Res*, vol. 46, no. 5, pp. 6081–6098, 2022, doi: 10.1002/er.7548.

[196] M. Subramanian, K. Shanmugavadivel, and P. S. Nandhini, “On fine-tuning deep learning models using transfer learning and hyper-parameters optimization for disease identification in maize leaves,” *Neural Comput Appl*, vol. 34, no. 16, pp. 13951–13968, 2022, doi: 10.1007/s00521-022-07246-w.

[197] N. Bacanin, C. Stoean, M. Zivkovic, M. Rakic, R. Strulak-Wójcikiewicz, and R. Stoean, “On the Benefits of Using Metaheuristics in the Hyperparameter Tuning of Deep Learning Models for Energy Load Forecasting,” *Energies (Basel)*, vol. 16, no. 3, 2023, doi: 10.3390/en16031434.

[198] A. D. Yuliyono and A. S. Girsang, “Artificial bee colony-optimized LSTM for bitcoin price prediction,” *Advances in Science, Technology and Engineering Systems*, vol. 4, no. 5, pp. 375–383, 2019, doi: 10.25046/aj040549.

[199] U. Erkan, A. Toktas, and D. Ustun, “Hyperparameter optimization of deep CNN classifier for plant species identification using artificial bee colony algorithm,” *J Ambient Intell Humaniz Comput*, no. Dudani 1976, 2022, doi: 10.1007/s12652-021-03631-w.

[200] S. Mittal and O. P. Sangwan, “ABC LSTM Optimizing Parameters of Deep LSTM using ABC Algorithm for Big Datasets,” *Int J Eng Adv Technol*, vol. 9, no. 5, pp. 221–226, 2020, doi: 10.35940/ijeat.d7649.069520.

[201] I. Strumberger, E. Tuba, N. Ba\vcanin, M. Zivkovic, M. Beko, and M. Tuba, “Designing Convolutional Neural Network Architecture by the Firefly Algorithm,” *2019 International Young Engineers Forum (YEF-ECE)*, pp. 59–65, 2019.

[202] S. Nematzadeh, F. Kiani, M. Torkamanian-Afshar, and N. Aydin, “Tuning hyperparameters of machine learning algorithms and deep neural networks

using metaheuristics: A bioinformatics study on biomedical and biological cases," *Comput Biol Chem*, vol. 97, p. 107619, 2022, doi: https://doi.org/10.1016/j.compbiolchem.2021.107619.

[203] M. Lohvithee, W. Sun, S. Chretien, and M. Soleimani, "Ant Colony-Based Hyperparameter Optimisation in Total Variation Reconstruction in X-ray Computed Tomography," *Sensors*, vol. 21, no. 2, 2021, doi: 10.3390/s21020591.

[204] A. Trajkovski and G. Madjarov, "Model Hyper Parameter Tuning using Ant Colony Optimization," 2022, pp. 37–41.

[205] Y. Tong and B. Yu, "Research on Hyper-Parameter Optimization of Activity Recognition Algorithm Based on Improved Cuckoo Search," *Entropy*, vol. 24, no. 6, 2022, doi: 10.3390/e24060845.

[206] M. A. Shaaban, M. Kashkash, M. Alghfeli, and A. Ibrahim, "Optimizing Deep Learning Model Parameters with the Bees Algorithm for Improved Medical Text Classification," *ArXiv*, vol. abs/2303.0, 2023, [Online]. Available: https://api.semanticscholar.org/CorpusID:257504979

[207] D. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, S. Rahim, and M. Zaidi, "The Bees Algorithm Technical Note," *Manufacturing Engineering Centre, Cardiff University, UK*, pp. 1–57, 2005.

[208] D. T. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, S. Rahim, and M. Zaidi, "The Bees Algorithm - A Novel Tool for Complex Optimisation Problems," *Intelligent Production Machines and Systems - 2nd I*PROMS Virtual International Conference 3-14 July 2006*, no. December, pp. 454–459, 2006, doi: 10.1016/B978-008045157-2/50081-X.

[209] A. H. Ismail and D. T. Pham, “Bees Traplining Metaphors for the Vehicle Routing Problem Using a Decomposition Approach BT - Intelligent Production and Manufacturing Optimisation—The Bees Algorithm Approach,” D. T. Pham and N. Hartono, Eds., Cham: Springer International Publishing, 2023, pp. 261–287. doi: 10.1007/978-3-031-14537-7_16.

[210] Mathworks, “1-D convolutional layer.” Accessed: Jun. 09, 2023. [Online]. Available: https://uk.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.convolution1dlayer.html

[211] MathWorks, “Batch normalization layer.” Accessed: Jun. 09, 2023. [Online]. Available: https://uk.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.batchnormalizationlayer.html

[212] MathWorks, “Fully connected layer.” Accessed: Jun. 09, 2023. [Online]. Available: https://uk.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.fullyconnectedlayer.html#mw_bad3166a-93e7-42c0-8bd2-740cd2a842ad

[213] Mathworks, “Long short-term memory (LSTM) layer for recurrent neural network (RNN).” Accessed: Jun. 09, 2023. [Online]. Available: https://uk.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.lstmlayer.html?s_tid=srchtitle_CNN layer_1

[214] Mathworks, “Gated recurrent unit (GRU) layer for recurrent neural network (RNN).” Accessed: Jun. 09, 2023. [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.grulayer.html

[215] A. H. Khan, X. Cao, S. Li, V. N. Katsikis, and L. Liao, “BAS-ADAM: An ADAM based approach to improve the performance of beetle antennae search optimizer,” *IEEE/CAA Journal of Automatica Sinica*, vol. 7, no. 2, pp. 461–471, 2020, doi: 10.1109/JAS.2020.1003048.

[216] E. Sutrisno, H. Oh, A. S. S. Vasan, and M. Pecht, “Estimation of remaining useful life of ball bearings using data driven methodologies,” *PHM 2012 - 2012 IEEE Int. Conf.on Prognostics and Health Management: Enhancing Safety, Efficiency, Availability, and Effectiveness of Systems Through PHM Technology and Application, Conference Program*, vol. 2, pp. 1–7, 2012, doi: 10.1109/ICPHM.2012.6299548.

[217] S. Hong, Z. Zhou, E. Zio, and K. Hong, “Condition assessment for the performance degradation of bearing based on a combinatorial feature extraction method,” *Digit Signal Process*, vol. 27, pp. 159–166, 2014, doi: https://doi.org/10.1016/j.dsp.2013.12.010.

[218] Y. Lei, N. Li, S. Gontarz, J. Lin, S. Radkowski, and J. Dybala, “A Model-Based Method for Remaining Useful Life Prediction of Machinery,” *IEEE Trans Reliab*, vol. 65, no. 3, pp. 1314–1326, 2016, doi: 10.1109/TR.2016.2570568.

[219] Y. Chen, G. Peng, Z. Zhu, and S. Li, “A novel deep learning method based on attention mechanism for bearing remaining useful life prediction,” *Appl Soft Comput*, vol. 86, p. 105919, 2020, doi: https://doi.org/10.1016/j.asoc.2019.105919.

[220] M. S. R. Mohd Saufi and K. A. Hassan, “Remaining useful life prediction using an integrated Laplacian-LSTM network on machinery components,” *Appl Soft*

Comput, vol. 112, p. 107817, 2021, doi: https://doi.org/10.1016/j.asoc.2021.107817.

[221] Y. Chen, D. Zhang, and W. Zhang, "MSWR-LRCN: A new deep learning approach to remaining useful life estimation of bearings," *Control Eng Pract*, vol. 118, p. 104969, 2022, doi: https://doi.org/10.1016/j.conengprac.2021.104969.

Appendix A: MATLAB Codes for Chapter 3

A-1. The example code for the individual DL model example

```
%Load data
load('B11+B12_XTrain.mat');
load('B11+B12_YTrain.mat');

%Define GRU Network Architecture
numFeatures = 2;
numHiddenUnits = 200;
numClasses = 2;

layers = [ ...
    sequenceInputLayer(numFeatures)
    gruLayer(numHiddenUnits,'OutputMode','sequence')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];

options = trainingOptions('adam', ...
    'MaxEpochs',300, ...
    'InitialLearnRate',0.001, ...
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropPeriod',270, ...
    'LearnRateDropFactor',0.9, ...
    'MiniBatchSize', 64,...
    'GradientThreshold',2, ...
    'Verbose',0, ...
    'Plots','training-progress');

%Train the GRU network
net = trainNetwork(XTrain,YTrain,layers,options);

%Test GRU Network
load('B13_XTest');
load('B13_YTest');

%Classify the test data using classify
YPred = classify(net,XTest);

%Compare the predictions with the test data by using a plot
figure
plot(YPred{1},'.-')
hold on
plot(YTest{1})
hold off

xlabel("Time Step")
ylabel("Activity")
title("Predicted Activities")
legend(["Predicted" "Test Data"])

%Calculate the accuracy of the predictions
accuracy = mean(YPred{1} == YTest{1});
```

A-2. The example code for the combination DL model

```
%Load data
load('B11+B12_XTrain.mat');
load('B11+B12_YTrain.mat');

%View the number of observations in the training data.
numObservations = numel(XTrain);

%View the number of classes in the training data.
classes = categories(YTrain{1});
numClasses = numel(classes);

%View the number of features of the training data.
numFeatures = size(XTrain{1},1);

%Define Deep Learning Model
numFilters = 128;
filterSize = 7;
numHiddenUnits = 250;

    layers = [
        sequenceInputLayer(numFeatures);
        convolution1dLayer(filterSize,numFilters,Padding="same")
        batchNormalizationLayer
        reluLayer
        maxPooling1dLayer(2,Padding="same")
        dropoutLayer(0.9)
        fullyConnectedLayer(250)
        lstmLayer(numHiddenUnits,'OutputMode','sequence')
        fullyConnectedLayer(numClasses)
        softmaxLayer
        classificationLayer];

%Specify Training Options
options = trainingOptions("adam", ...
    MaxEpochs=50, ...
    InitialLearnRate=0.001, ...
    miniBatchSize=32, ...
    Plots="training-progress", ...
    Verbose=0);

%Train model
net = trainNetwork(XTrain,YTrain,layers,options);

%Test CNN_LSTM Network
load('B13_XTest');
load('B13_YTest');

%Classify the test data using classify
YPred = classify(net,XTest);

%Compare the predictions with the test data by using a plot
figure
plot(YPred{1},'.-')
hold on
plot(YTest{1})
hold off

xlabel("Time Step")
ylabel("Activity")
title("Predicted Activities")
legend(["Predicted" "Test Data"])

%Calculate the accuracy of the predictions
accuracy = mean(YPred{1} == YTest{1});
```

Appendix B: MATLAB Codes for Chapter 4

B-1. The example code for the individual DL model example

```
clc;
clear;
close all;
tic
%% Importing Training Data
load('B11+B12_XTrain.mat');
load('B11+B12_YTrain.mat');

%% Bees Algorithm Parameter
n = 3; nep = 6;
recruitment = round(linspace(nep,1,n));
assignment = linspace(0,1,n);
MaxIt = 9;

%Hyperparameters: numHiddenUnits,learning_rate,
%LearnRateDropPeriod,LearnRateDropFactor, batch_size
VarMax = [200 0.01 300 1 512];
VarMin = [10 0.00001 10 0 32];
range = VarMax - VarMin;
max_epochs = 300;

%% Initialization
Bees = Bee.empty(n,0);
bestSol = Bee;

%% Generating Initial Solutions
for i=1:n
    Bees(i).hyperparameters = Hyperparameter_Generator(VarMax, VarMin);
    [net, info] = Model_Generator(XTrain, YTrain, Bees(i).hyperparameters,
max_epochs);
    loss = Latest_Loss(info);
    Bees(i).loss = loss;
    Bees(i).net = net;
    Bees(i).Size = range;
end
    size = linspace(0,1,n);

%% Sites Selection
[~,RankOrder] = sort([Bees.loss]);
Bees = Bees(RankOrder);
bestSol.loss = inf;

%% Bees Algorithm Local and Global Search
checkpointPath = "/rds/projects/p/phamdt-autoreman-deep-learning/Nathinee/Bee
Hyperparameter optimisation/LSTM/B1";

for it=1:MaxIt

     % All Sites (Exploitation and Exploration)
     for i=1:n
         bestnewbee = Bee;
         bestnewbee.loss = inf;
         assignment = D_Triangular_real(0, size(i), 1, 1, recruitment(i));
         for j=1:recruitment(i)
             newbee = Bee;
             newbee.hyperparameters = Foraging(Bees(i).hyperparameters,
assignment(j), VarMax, VarMin, Bees(i).Size);
```

```
            [net, info] = Model_Generator(XTrain, YTrain,
newbee.hyperparameters, max_epochs);
            loss = Latest_Loss(info);
            newbee.loss = loss;
            newbee.net = net;
            newbee.Size = Bees(i).Size;
            disp(["iteration: " num2str(i) " percentage completion:"
num2str((j*100)/recruitment(i)) "%"]);

            if newbee.loss<bestnewbee.loss
                bestnewbee = newbee;
            end
        end
        if bestnewbee.loss<Bees(i).loss
            Bees(i) = bestnewbee;
        end

    end

    %SORTING
    [~, RankOrder] = sort([Bees.loss]);
    Bees = Bees(RankOrder);

    %Updating the best solution ever found
    OptSol = Bees(1);

    if OptSol.loss<bestSol.loss
        bestSol = OptSol;
    end

    OptCost(it) = bestSol.loss;
    Time(it) = toc;
    %Displaying iteration example
    disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(OptCost(it)) ' -
-> Time = ' num2str(Time(it)) ' seconds' ]);

    %save checkpoint of Iteration
   checkpointFilename = fullfile(checkpointPath,
sprintf('beeAlgorithm_checkpoint_iteration_%d.mat', it));
    save(checkpointFilename, 'Bees', 'OptSol', 'OptCost', 'it');

end

%Results
save('best_bee_found.mat', "OptSol");
figure;
semilogy(OptCost, 'LineWidth', 2);
xlabel('Iteration');
ylabel('Best Loss');


function [hyperparameter_value]=Hyperparameter_Generator(VarMax, VarMin)
    max_hiddenunits = VarMax(1); min_hiddenunits = VarMin(1);
    max_learning_rate = VarMax(2); min_learning_rate = VarMin(2);
    max_LearnRateDropPeriod = VarMax(3); min_LearnRateDropPeriod = VarMin(3);
    max_batch_size = VarMax(5); min_batch_size = VarMin(5);
    hyperparameter_value = zeros(5,1);
    hyperparameter_value(1) = pow2(floor(log2(min_hiddenunits +
(max_hiddenunits-min_hiddenunits)*rand())));
    hyperparameter_value(2) = min_learning_rate + (max_learning_rate-
min_learning_rate)*(rand());
    hyperparameter_value(3) = round(min_LearnRateDropPeriod +
(max_LearnRateDropPeriod-min_LearnRateDropPeriod)*rand());
    hyperparameter_value(4) = rand();
    hyperparameter_value(5) = pow2(ceil(log2(min_batch_size +
(max_batch_size-min_batch_size)*(rand()))));
```

```
end
function [net, info]=Model_Generator(XTrain, YTrain, hyperparameters,
max_epochs)

numClasses = 2;
numFeatures = 2;

layers = [
        sequenceInputLayer(numFeatures);
        lstmLayer(hyperparameters(1),'OutputMode','sequence')
        fullyConnectedLayer(numClasses)
        softmaxLayer
        classificationLayer];


%Specify Training Options
options = trainingOptions("adam", ...
    MaxEpochs=max_epochs, ...
    InitialLearnRate=hyperparameters(2), ...
    LearnRateSchedule="piecewise",...
    LearnRateDropPeriod=hyperparameters(3), ...
    LearnRateDropFactor=hyperparameters(4), ...
    miniBatchSize=hyperparameters(5), ...
    Verbose=0);

%Train model
[net,info] = trainNetwork(XTrain,YTrain,layers,options);
End


function [loss_val]=Latest_Loss(info)
    training_loss_val = info.TrainingLoss;
    loss_val = training_loss_val(numel(training_loss_val));
end

function [M] = D_Triangular_real(k,t,b,baris,kolom)
     M = zeros(baris,kolom);
     for i=1:baris
         for j=1:kolom
             M(i,j) = D_Tri_real(k,t,b);
             %disp("M(i,j)"); disp(M(i,j));
         end
     end
end


function [angka] = D_Tri_real(k,t,b)
    m=randi([1 10]);
    %disp("m"); disp(m);
    a=(t-k)/10;
    %disp("t- m*a"); disp(t-m*a);
    b=(b-t)/10;
    %disp("t +m*b"); disp(t+m*b);

    angka=unifrnd((t-m*a), (t+m*b), 1);
end


function y=Foraging(x, ass, VarMax, VarMin, size)
y = zeros(5,1);
for i=1:5
    Vmx = VarMax(i); Vmn = VarMin(i); sz=size(i);
    %disp("VarMax"); disp(Vmx); disp("VarMin"); disp(Vmn); disp("Size");
disp(sz);
    r=ass*sz;
    %disp("r"); disp(r);
```

```
    y(i)=x(i);
    y(i)=y(i)+ r*((-1)^randi(2));
    if i==3
        y(i)=round(y(i));
    end

    if i==1 || i==5
        y(i) = pow2(ceil(log2(y(i))));
    end

    if y(i)>Vmx
        y(i) = Vmx;
    end

    if y(i)<Vmn
        y(i) = Vmn;
    end
    disp("y(i)"); disp(y(i));
end


classdef Bee
    properties
        hyperparameters;
        loss;
        net;
        Size;
    end

end


hyperparameters = Hyperparameter_Extractor(OptSol);

%Test LSTM Network
best_bee = load('best_bee_found.mat');
best_bee = best_bee.OptSol;
net = best_bee.net;
load('B13_XTest');
load('B13_YTest');

%Classify the test data using classify
YPred = classify(net,XTest);

%Compare the predictions with the test data by using a plot
figure
plot(YPred{1},'.-')
hold on
plot(YTest{1})
hold off

xlabel("Time Step")
ylabel("Activity")
title("Predicted Activities")
legend(["Predicted" "Test Data"])


%Calculate the accuracy of the predictions
accuracy = mean(YPred{1} == YTest{1});
netly = net.Layers;
```

B-2. The example code for the combination DL model

```
clc;
clear;
close all;
tic
%% Importing Training Data
load('B11+B12_XTrain.mat');
load('B11+B12_YTrain.mat');

%% Bees Algorithm Parameter
n = 10; nep = 20;
recruitment = round(linspace(nep,1,n));
assignment = linspace(0,1,n);
MaxIt = 10;

%Hyperparameters: num_filters, filter_size, num_hidden_units,
%drop_porb,learning_rate, batch_size, fully connected
VarMax = [128 11 200 0.9 0.01 512 200];
VarMin = [4 2 10 0.1 0.00001 32 10];
range = VarMax - VarMin;
max_epochs = 100;

%% Initialization
Bees = Bee.empty(n,0);
bestSol = Bee;

%% Generating Initial Solutions
for i=1:n
    Bees(i).hyperparameters = Hyperparameter_Generator(VarMax, VarMin);
    [net, info] = Model_Generator(XTrain, YTrain, Bees(i).hyperparameters,
max_epochs);
    loss = Latest_Loss(info);
    Bees(i).loss = loss;
    Bees(i).net = net;
    Bees(i).Size = range;
end
    size = linspace(0,1,n);

%% Sites Selection
[~,RankOrder] = sort([Bees.loss]);
Bees = Bees(RankOrder);
bestSol.loss = inf;

%% Bees Algorithm Local and Global Search
checkpointPath = "/rds/projects/p/phamdt-autoreman-deep-
learning/Nathinee/Hyperparameter search/B1_CNN+GRU";

for it=1:MaxIt

     % All Sites (Exploitation and Exploration)
     for i=1:n
         bestnewbee = Bee;
         bestnewbee.loss = inf;
         assignment = D_Triangular_real(0, size(i), 1, 1, recruitment(i));
         for j=1:recruitment(i)
             newbee = Bee;
             newbee.hyperparameters = Foraging(Bees(i).hyperparameters,
assignment(j), VarMax, VarMin, Bees(i).Size);
             [net, info] = Model_Generator(XTrain, YTrain,
newbee.hyperparameters, max_epochs);
             loss = Latest_Loss(info);
             newbee.loss = loss;
             newbee.net = net;
             newbee.Size = Bees(i).Size;
```

```
            disp(["iteration: " num2str(i) " percentage completion:"
num2str((j*100)/recruitment(i)) "%"]);

            if newbee.loss<bestnewbee.loss
                bestnewbee = newbee;
            end
        end
        if bestnewbee.loss<Bees(i).loss
            Bees(i) = bestnewbee;
        end

    end

    %SORTING
    [~, RankOrder] = sort([Bees.loss]);
    Bees = Bees(RankOrder);

    %Updating the best solution ever found
    OptSol = Bees(1);

    if OptSol.loss<bestSol.loss
        bestSol = OptSol;
    end

    OptCost(it) = bestSol.loss;
    Time(it) = toc;
    %Displaying iteration example
    disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(OptCost(it)) ' -
-> Time = ' num2str(Time(it)) ' seconds' ]);

    %save checkpoint of Iteration
   checkpointFilename = fullfile(checkpointPath,
sprintf('beeAlgorithm_checkpoint_iteration_%d.mat', it));
    save(checkpointFilename, 'Bees', 'OptSol', 'OptCost', 'it');

end

%Results
save('best_bee_found.mat', "OptSol");
figure;
semilogy(OptCost, 'LineWidth', 2);
xlabel('Iteration');
ylabel('Best Loss');


function [hyperparameter_value]=Hyperparameter_Generator(VarMax, VarMin)
    max_filters = VarMax(1); min_filters = VarMin(1);
    max_filter_size = VarMax(2); min_filter_size = VarMin(2);
    max_hidden_units = VarMax(3); min_hidden_units = VarMin(3);
    max_fully_connected = VarMax(7); min_fully_connected = VarMin(7);
    max_learning_rate = VarMax(5); min_learning_rate = VarMin(5);
    max_batch_size = VarMax(6); min_batch_size = VarMin(6);
    hyperparameter_value = zeros(6,1);
    hyperparameter_value(1) = pow2(floor(log2(min_filters + (max_filters-
min_filters)*rand())));
    hyperparameter_value(2) = round(min_filter_size + (max_filter_size-
min_filter_size)*rand());
    hyperparameter_value(3) = round(min_hidden_units + (max_hidden_units-
min_hidden_units)*rand());
    hyperparameter_value(4) = rand();
    hyperparameter_value(5) = min_learning_rate + (max_learning_rate-
min_learning_rate)*(rand());
    hyperparameter_value(6) = pow2(ceil(log2(min_batch_size +
(max_batch_size-min_batch_size)*(rand()))));
    hyperparameter_value(7) = round(min_fully_connected +
(max_fully_connected-min_fully_connected)*rand());
```

```
end


function [net, info]=Model_Generator(XTrain, YTrain, hyperparameters,
max_epochs)

numClasses = 2;
numFeatures = 2;

    layers = [
        sequenceInputLayer(numFeatures);
         convolution1dLayer(hyperparameters(2),hyperparameters(1),
         Padding="same")
        batchNormalizationLayer
        reluLayer
        maxPooling1dLayer(2,Padding="same")
        dropoutLayer(hyperparameters(4))
        fullyConnectedLayer(hyperparameters(7))
        gruLayer(hyperparameters(3),'OutputMode','sequence')
        fullyConnectedLayer(numClasses)
        softmaxLayer
        classificationLayer];

%Specify Training Options
options = trainingOptions("adam", ...
    MaxEpochs=max_epochs, ...
    InitialLearnRate=hyperparameters(5), ...
    miniBatchSize=hyperparameters(6), ...
    Verbose=1);

%Train model
[net,info] = trainNetwork(XTrain,YTrain,layers,options);
End


function [loss_val]=Latest_Loss(info)
    training_loss_val = info.TrainingLoss;
    loss_val = training_loss_val(numel(training_loss_val));
end


function [M] = D_Triangular_real(k,t,b,baris,kolom)
     M = zeros(baris,kolom);
     for i=1:baris
         for j=1:kolom
             M(i,j) = D_Tri_real(k,t,b);
             %disp("M(i,j)"); disp(M(i,j));
         end
     end
end


function [angka] = D_Tri_real(k,t,b)
    m=randi([1 10]);
    %disp("m"); disp(m);
    a=(t-k)/10;
    %disp("t- m*a"); disp(t-m*a);
    b=(b-t)/10;
    %disp("t +m*b"); disp(t+m*b);

    angka=unifrnd((t-m*a), (t+m*b), 1);
end
```

```
function y=Foraging(x, ass, VarMax, VarMin, size)
y = zeros(7,1);
for i=1:7
    Vmx = VarMax(i); Vmn = VarMin(i); sz=size(i);
    %disp("VarMax"); disp(Vmx); disp("VarMin"); disp(Vmn); disp("Size");
disp(sz);
    r=ass*sz;
    %disp("r"); disp(r);
    y(i)=x(i);
    y(i)=y(i)+ r*((-1)^randi(2));
    if i==2 || i==3 || i==7
        y(i)=round(y(i));
    end

    if i==1 || i==6
        y(i) = pow2(ceil(log2(y(i))));
    end

    if y(i)>Vmx
        y(i) = Vmx;
    end

    if y(i)<Vmn
        y(i) = Vmn;
    end
    disp("y(i)"); disp(y(i));
end


classdef Bee
    properties
        hyperparameters;
        loss;
        net;
        Size;
    end

end

hyperparameters = Hyperparameter_Extractor(OptSol);

%Test CNN_GRU Network
best_bee = load('best_bee_found.mat');
best_bee = best_bee.OptSol;
net = best_bee.net;
load('B13_XTest');
load('B13_YTest');

%Classify the test data using classify
YPred = classify(net,XTest);

%Compare the predictions with the test data by using a plot
figure
plot(YPred{1},'.-')
hold on
plot(YTest{1})
hold off

xlabel("Time Step")
ylabel("Activity")
title("Predicted Activities")
legend(["Predicted" "Test Data"])

%Calculate the accuracy of the predictions
accuracy = mean(YPred{1} == YTest{1});
netly = net.Layers;
```

Appendix C: MATLAB Codes for Chapter 5

C-1. The example code for the individual DL model example

```
clc;
clear;
close all;
tic

%Load data
load('B11+B12_XTrain.mat');
load('B11+B12_YTrain01.mat');

%View the number of observations in the training data.
numObservations = numel(XTrain);

%View the number of classes in the training data.
%classes = categories(YTrain{1});
numClasses = 2;

%View the number of features of the training data.
numFeatures = size(XTrain{1},1);

%Define Deep Learning Model
numHiddenUnits = 200;

    layers = [
        sequenceInputLayer(numFeatures);
        lstmLayer(numHiddenUnits,'OutputMode','sequence')
        fullyConnectedLayer(numClasses)
        softmaxLayer
        classificationLayer];


% Remove the classification layer.
layersWithoutOutput = layers(1:end-1);

% Convert the modified SeriesNetwork to a dlnetwork.
dlnet = dlnetwork(layersWithoutOutput);

% Get the initial weights and biases from the dlnetwork model
[initWeights,initinputWeights,initrecurrentWeights,initBiases] =
extractWeightsBiases(dlnet);

% Convert the initial weights and biases to a format that can be used by the
Bee Algorithm
initParameters =
weightsBiasesToParam(initWeights,initinputWeights,initrecurrentWeights,initBi
ases);

% Define the custom cost function that evaluates the loss for given weights
and biases
CostFunction = @(x) modelGradients(x, dlnet, XTrain, YTrain); %x=parameters

% Run the Bee Algorithm
nVar = sum(cellfun(@numel, dlnet.Learnables{:, 'Value'}));  % Number of
Decision Variables

VarSize = [1 nVar];        % Decision Variables Matrix Size

VarMin = -0.1;             % Decision Variables Lower Bound
VarMax = 0.1;              % Decision Variables Upper Bound
```

```
%% Bees Algorithm Parameters

MaxIt = 20;         % Maximum Number of Iterations

nScoutBee = 30;     % Number of Scout Bees (n)

nSelectedSite = 5;  % Number of Selected Sites (m)

nEliteSite = 1;     % Number of Selected Elite Sites (e)

nSelectedSiteBee = 5;% Number of Recruited Bees for Selected Sites (nsp)

nEliteSiteBee = 10;        % Number of Recruited Bees for Elite Sites (nep)

r = 0.1*(VarMax-VarMin);   % Neighborhood Radius

rdamp = 0.95;              % Neighborhood Radius Damp Rate

%% Initialization

% Empty Bee Structure
empty_bee.initParameters = [];
empty_bee.Cost = [];

% Initialize Bees Array
bee = repmat(empty_bee, nScoutBee, 1);

% Create New Solutions
for i = 1:nScoutBee
    bee(i).initParameters = unifrnd(VarMin, VarMax, VarSize);
    bee(i).Cost = CostFunction(bee(i).initParameters);
end

%extract data from dlarray
beeCosts = cellfun(@extractdata,{bee.Cost});

% Sort
[~, SortOrder] = sort(beeCosts);
bee = bee(SortOrder);

% Update Best Solution Ever Found
BestSol = bee(1);

% Array to Hold Best Cost Values
BestCost = zeros(MaxIt, 1);

%% Bees Algorithm Main Loop

% Initialize Adam optimizer states
m = zeros(VarSize);
v = zeros(VarSize);

% Initialize Adam optimizer parameters
learnRate = 0.001;
beta1 = 0.9;
beta2 = 0.999;
epsilon = 1e-8;

checkpointPath = "/rds/projects/p/phamdt-autoreman-deep-
learning/Nathinee/Bee_EliteAdam/LSTM∀GRU/B1";
if ~exist(checkpointPath, 'dir')
    mkdir(checkpointPath);
end


for it = 1:MaxIt
```

```
    % Elite Sites
    for i = 1:nEliteSite

        bestnewbee.Cost = inf;

        for j = 1:nEliteSiteBee
            % Perform Bee Dance to get new parameters
            newParameters = PerformBeeDance(bee(i).initParameters, r);

            % Compute gradients and loss using your 'modelGradients' function
            [gradients, loss] = modelGradients(newParameters, dlnet, XTrain,
YTrain);

            % Update parameters using Adam optimization
            [newParameters, m, v] = updateParametersAdam(newParameters,
gradients, m, v, beta1, beta2, epsilon, learnRate, it);

            % Evaluate new parameters
            newbee.initParameters = newParameters;
            newbee.Cost = CostFunction(newbee.initParameters);  % You might
use 'loss' directly here if it's equivalent to your CostFunction

            if newbee.Cost < bestnewbee.Cost
                bestnewbee = newbee;
            end
        end
    end

    % Selected Non-Elite Sites
    for i = nEliteSite+1:nSelectedSite

        bestnewbee.Cost = inf;

        for j = 1:nSelectedSiteBee
            newbee.initParameters = PerformBeeDance(bee(i).initParameters, r);
            newbee.Cost = CostFunction(newbee.initParameters);
            if newbee.Cost<bestnewbee.Cost
                bestnewbee = newbee;
            end
        end

        if bestnewbee.Cost<bee(i).Cost
            bee(i) = bestnewbee;
        end

    end

    % Non-Selected Sites
    for i = nSelectedSite+1:nScoutBee
        bee(i).initParameters = unifrnd(VarMin, VarMax, VarSize);
        bee(i).Cost = CostFunction(bee(i).initParameters);
    end

    %extract data from dlarray
    beeCosts = cellfun(@extractdata,{bee.Cost});

    % Sort
    [~, SortOrder] = sort(beeCosts);
    bee = bee(SortOrder);


    % Update Best Solution Ever Found
    BestSol = bee(1);
```

```
    % Store Best Cost Ever Found
    BestCost(it) = BestSol.Cost;

   Time(it) = toc;

    % Display Iteration Information
    disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCost(it)) ' -
-> Time = ' num2str(Time(it))' seconds']);

    % Damp Neighborhood Radius
    r = r*rdamp;

     %save checkpoint of Iteration
    checkpoint = fullfile(checkpointPath,
sprintf('beeAlgorithm_checkpointLSTM_iteration_%d.mat', it));
    save(checkpoint, 'bee', 'BestSol', 'BestCost', 'it');

end
%Results
save('best_bee_found_LSTM.mat', "BestSol");
figure;
semilogy(BestCost, 'LineWidth', 2);
xlabel('Iteration');
ylabel('Best Loss');

% After the Bee Algorithm has finished, extract the best solution
bestParameters = BestSol.initParameters;

% Convert the best solution to weights and biases
[bestweights, bestinputWeights, bestrecurrentWeights, bestbiases] =
paramToWeightsBiases(bestParameters, dlnet.Learnables);

% Assign the best weights and biases to the deep learning model
dlnet = assignLearnables(dlnet, bestweights, bestinputWeights,
bestrecurrentWeights, bestbiases);

layers = replaceWeightsBiases(layers, dlnet.Learnables);

% Train the model with the best weights and biases using 'trainNetwork'
options = trainingOptions("adam", ... % Use "adam" optimizer here
    MaxEpochs=300, ...
    InitialLearnRate=0.0005, ...
    LearnRateSchedule="piecewise",...
    LearnRateDropPeriod= 190, ...
    LearnRateDropFactor= 0.1669, ...
    miniBatchSize=64, ...
    Verbose=0,...
    Plots="training-progress");


%convert double data to categorical YTrain
classes = [0,1];
for i= 1:length(YTrain)
    YTrain{i} = categorical(YTrain{i},classes);
end


% Train the model with the best weights and biases using 'trainNetwork'
net = trainNetwork(XTrain, YTrain, layers, options);
```

```
function [weights,inputWeights, recurrentWeights, biases] =
extractWeightsBiases(dlnet)
    learnables = dlnet.Learnables;
    weights = {};
    inputWeights ={};
    recurrentWeights = {};
    biases = {};

    for i = 1:height(learnables)
        if strcmp(learnables.Parameter(i), 'Weights')
            weights{end+1} = gather(extractdata(learnables.Value{i}));
        elseif strcmp(learnables.Parameter(i), 'InputWeights')
            inputWeights{end+1} = gather(extractdata(learnables.Value{i}));
        elseif strcmp(learnables.Parameter(i), 'RecurrentWeights')
            recurrentWeights{end+1} =
gather(extractdata(learnables.Value{i}));
        elseif strcmp(learnables.Parameter(i), 'Bias')
            biases{end+1} = gather(extractdata(learnables.Value{i}));

        end
    end
end


function parameters =
weightsBiasesToParam(weights,inputWeights,recurrentWeights,biases)
    parameters = [];
    for i = 1:numel(weights)
        parameters = [parameters; weights{i}(:)];
    end
    for i = 1:numel(inputWeights)
        parameters = [parameters; inputWeights{i}(:)];
    end
    for i = 1:numel(recurrentWeights)
        parameters = [parameters; recurrentWeights{i}(:)];
    end
    for i = 1:numel(biases)
        parameters = [parameters; biases{i}(:)];
    end
end


function [gradients, loss] = modelGradients(parameters, dlnet, XTrain,
YTrain)
    % Reshape parameters into weights and biases
    [weights, inputWeights, recurrentWeights, biases] =
paramToWeightsBiases(parameters, dlnet.Learnables);

    % Assign weights and biases to the dlnet
    dlnet = assignLearnables(dlnet, weights, inputWeights, recurrentWeights,
biases);

    % Perform forward and backward passes and compute gradients
    [gradients, loss] = dlfeval(@modelGradientsLoss, dlnet, XTrain, YTrain);
end


function [loss,gradients] = modelGradientsLoss(dlnet, XTrain, YTrain)

% Arrange your data into Sequence Time Batch (STB) format and then convert to
dlarray.
numObservations = numel(XTrain);
sequenceLength = cellfun(@(x) size(x,2), XTrain);  % Get sequence lengths for
each observation
XTrain = dlarray([XTrain{:}], 'CTB');  % concatenate along the third
dimension
```

```
% Do the same for YTrain
numObservationsY = numel(YTrain);
sequenceLengthY = cellfun(@(x) size(x,2), YTrain);  % Get sequence lengths
for each observation

% Adjust YTrain values so they start from 1
YTrain = cellfun(@(x) x+1, YTrain, 'UniformOutput', false);

% Convert YTrain to one-hot encoding
YTrainOneHot = cell(numObservationsY, 1);
for i = 1:numObservationsY
    YTrainOneHot{i} = full(ind2vec(YTrain{i}));
end

YTrain = dlarray([YTrainOneHot{:}], 'CTB');

% Forward data through network.
dlYPred = forward(dlnet,XTrain);

% Calculate cross-entropy loss.
loss = crossentropy(dlYPred,YTrain);

% Calculate gradients of loss with respect to learnable parameters.
gradients = dlgradient(loss,dlnet.Learnables);

end


function dlnet = assignLearnables(dlnet, weights, inputWeights,
recurrentWeights, biases)
    numLayers = numel(dlnet.Layers);
    newLearnables = table('Size',[0 3],
'VariableTypes',{'string','string','cell'},'VariableNames',{'Layer','Parameter',
'Value'});
    weightsIndex = 1;
    biasesIndex = 1;
    inputWeightsIndex = 1;
    recurrentWeightsIndex = 1;
    for i = 1:numLayers
        layer = dlnet.Layers(i);
        if isprop(layer, 'Weights') && weightsIndex <= numel(weights)
            %disp(['Setting Weights for layer: ', layer.Name]);
            %disp(['Expected size: ', mat2str(size(layer.Weights))]);
            %disp(['Received size: ', mat2str(size(weights{weightsIndex}))]);
            newRow = {layer.Name, 'Weights',
{dlarray(single(weights{weightsIndex}))}};
            newLearnables = [newLearnables; newRow];
            weightsIndex = weightsIndex + 1;
        end
        if isprop(layer, 'InputWeights') && inputWeightsIndex <=
numel(inputWeights)
            %disp(['Setting InputWeights for layer: ', layer.Name]);
            %disp(['Expected size: ', mat2str(size(layer.InputWeights))]);
            %disp(['Received size: ',
mat2str(size(inputWeights{inputWeightsIndex}))]);
            newRow = {layer.Name, 'InputWeights',
{dlarray(single(inputWeights{inputWeightsIndex}))}};
            newLearnables = [newLearnables; newRow];
            inputWeightsIndex = inputWeightsIndex + 1;
        end
        if isprop(layer, 'RecurrentWeights') && recurrentWeightsIndex <=
numel(recurrentWeights)
            %disp(['Setting RecurrentWeights for layer: ', layer.Name]);
            %disp(['Expected size: ',mat2str(size(layer.RecurrentWeights))]);
            %disp(['Received size: ',
mat2str(size(recurrentWeights{recurrentWeightsIndex}))]);
```

```
            newRow = {layer.Name, 'RecurrentWeights',
{dlarray(single(recurrentWeights{recurrentWeightsIndex}))}};
            newLearnables = [newLearnables; newRow];
            recurrentWeightsIndex = recurrentWeightsIndex + 1;
        end
        if isprop(layer, 'Bias') && biasesIndex <= numel(biases)
            %disp(['Setting Biases for layer: ', layer.Name]);
            %disp(['Expected size: ', mat2str(size(layer.Bias))]);
            %disp(['Received size: ', mat2str(size(biases{biasesIndex}))]);
            newRow = {layer.Name, 'Bias',
{dlarray(single(biases{biasesIndex}))}};
            newLearnables = [newLearnables; newRow];
            biasesIndex = biasesIndex + 1;
        end
    end
    dlnet.Learnables = newLearnables;
end


function newPosition = PerformBeeDance(currentPosition, r)
    % Generate a random number within the range [-r, r]
    randomShift = 2 * r * rand(size(currentPosition)) - r;

    % Calculate newPosition based on currentPosition and randomShift
    newPosition = currentPosition + randomShift;
end


function [updatedParameters, m, v] = updateParametersAdam(parameters,
gradients, m, v, beta1, beta2, epsilon, learningRate, t)
    % Adam optimization for updating parameters

    % Calculate the moving averages of the gradients
    m = beta1 .* m + (1 - beta1) .* gradients;

    % Calculate the moving averages of the squared gradients
    v = beta2 .* v + (1 - beta2) .* gradients.^2;

    % Compute bias-corrected estimates of the moving averages
    mHat = m ./ (1 - beta1.^t);
    vHat = v ./ (1 - beta2.^t);

    % Update the parameters
    updatedParameters = parameters - learningRate .* mHat ./ (sqrt(vHat) +
epsilon);
end


function [weights,inputWeights,recurrentWeights,biases] =
paramToWeightsBiases(parameters, learnables)

    weights = {};
    inputWeights = {};
    recurrentWeights = {};
    biases = {};
    currentParamIdx = 1;
    currentWeightIdx = 1;
    currentInputWeightsIdx = 1;
    currentRecurrentWeightsIdx = 1;
    currentBiasIdx = 1;


    for i = 1:size(learnables, 1)
        if string(learnables.Layer(i)) ~= "batchnorm" &&
string(learnables.Parameter(i)) == "Weights"
            weightSize = size(learnables.Value{i});
```

```
            numWeightElements = prod(weightSize);
            if currentParamIdx + numWeightElements - 1 <= numel(parameters)
                weight = parameters(currentParamIdx : currentParamIdx +
numWeightElements - 1);
                weight = reshape(weight, weightSize);
                weights{currentWeightIdx} = weight;
                currentWeightIdx = currentWeightIdx + 1;
                currentParamIdx = currentParamIdx + numWeightElements;
            else
                error('Parameters do not have enough elements for the weights
of layer %d.', i);
            end
        elseif string(learnables.Layer(i)) ~= "batchnorm" &&
string(learnables.Parameter(i)) == "InputWeights"
            weightSize = size(learnables.Value{i});
            numInputWeightElements = prod(weightSize);
            if currentParamIdx + numInputWeightElements - 1 <=
numel(parameters)
                inputWeight = parameters(currentParamIdx : currentParamIdx +
numInputWeightElements - 1);
                inputWeight = reshape(inputWeight, weightSize);
                inputWeights{currentInputWeightsIdx} = inputWeight;
                currentInputWeightsIdx = currentInputWeightsIdx + 1;
                currentParamIdx = currentParamIdx + numInputWeightElements;
            else
                error('Parameters do not have enough elements for the input
weights of layer %d.', i);
            end
        elseif string(learnables.Layer(i)) ~= "batchnorm" &&
string(learnables.Parameter(i)) == "RecurrentWeights"
            weightSize = size(learnables.Value{i});
            numRecurrentWeightElements = prod(weightSize);
            if currentParamIdx + numRecurrentWeightElements - 1 <=
numel(parameters)
                recurrentWeight = parameters(currentParamIdx :
currentParamIdx + numRecurrentWeightElements - 1);
                recurrentWeight = reshape(recurrentWeight, weightSize);
                recurrentWeights{currentRecurrentWeightsIdx} =
recurrentWeight;
                currentRecurrentWeightsIdx = currentRecurrentWeightsIdx + 1;
                currentParamIdx = currentParamIdx +
numRecurrentWeightElements;
            else
                error('Parameters do not have enough elements for the
recurrent weights of layer %d.', i);
            end
        elseif string(learnables.Layer(i)) ~= "batchnorm" &&
string(learnables.Parameter(i)) == "Bias"
            biasSize = size(learnables.Value{i});
            numBiasElements = prod(biasSize);
            if currentParamIdx + numBiasElements - 1 <= numel(parameters)
                bias = parameters(currentParamIdx : currentParamIdx +
numBiasElements - 1);
                bias = reshape(bias, biasSize);
                biases{currentBiasIdx} = bias;
                currentBiasIdx = currentBiasIdx + 1;
                currentParamIdx = currentParamIdx + numBiasElements;
            else
                error('Parameters do not have enough elements for the bias of
layer %d.', i);
            end
        end
    end
end
```

```
function layers = replaceWeightsBiases(layers, learnables)
    for i = 1:numel(layers)
        layer = layers(i);
        if isprop(layer, 'Name') % Check if layer has a 'Name' property
            name = layer.Name;
            idxWeights = find(learnables.Layer == name & learnables.Parameter
== "Weights");
            idxBias = find(learnables.Layer == name & learnables.Parameter ==
"Bias");
            idxInputWeights = find(learnables.Layer == name &
learnables.Parameter == "InputWeights");
            idxRecurrentWeights = find(learnables.Layer == name &
learnables.Parameter == "RecurrentWeights");

            if ~isempty(idxWeights) && isprop(layer, 'Weights')
                layer.Weights = learnables.Value{idxWeights};
            end

            if ~isempty(idxBias) && isprop(layer, 'Bias')
                layer.Bias = learnables.Value{idxBias};
            end

            if ~isempty(idxInputWeights) && isprop(layer, 'InputWeights')
                layer.InputWeights = learnables.Value{idxInputWeights};
            end

            if ~isempty(idxRecurrentWeights) && isprop(layer,
'RecurrentWeights')
                layer.RecurrentWeights =
learnables.Value{idxRecurrentWeights};
            end

            layers(i) = layer;
        end
    end
end


%Test LSTM Network
load('B13_XTest');
load('B13_YTest01');

%Classify the test data using classify
YPred = classify(net,XTest);


%Compare the predictions with the test data by using a plot
figure
plot(YPred{1},'.-')
hold on
plot(YTest{1})
hold off

xlabel("Time Step")
ylabel("Activity")
title("Predicted Activities")
legend(["Predicted" "Test Data"])

%Calculate the accuracy of the predictions
accuracy = mean(YPred{1} == YTest{1});
```

C-2. The example code for the combination DL model

```
clc;
clear;
close all;
tic

%Load data
load('B11+B12_XTrain.mat');
load('B11+B12_YTrain01.mat');

%View the number of observations in the training data.
numObservations = numel(XTrain);

%View the number of classes in the training data.
%classes = categories(YTrain{1});
numClasses = 2;

%View the number of features of the training data.
numFeatures = size(XTrain{1},1);

%Define Deep Learning Model
numFilters = 128;
filterSize = 9;
numHiddenUnits = 10;

    layers = [
        sequenceInputLayer(numFeatures);
        convolution1dLayer(filterSize,numFilters,Padding="same")
        batchNormalizationLayer
        reluLayer
        maxPooling1dLayer(2,Padding="same")
        dropoutLayer(0.9)
        fullyConnectedLayer(10)
        lstmLayer(numHiddenUnits,'OutputMode','sequence')
        fullyConnectedLayer(numClasses)
        softmaxLayer
        classificationLayer];

% Remove the classification layer.
layersWithoutOutput = layers(1:end-1);

% Convert the modified SeriesNetwork to a dlnetwork.
dlnet = dlnetwork(layersWithoutOutput);

% Get the initial weights and biases from the dlnetwork model
[initWeights,initOffsets,
initScales,initinputWeights,initrecurrentWeights,initBiases] =
extractWeightsBiases(dlnet);

% Convert the initial weights and biases to a format that can be used by the
Bee Algorithm
initParameters = weightsBiasesToParam(initWeights,initOffsets,
initScales,initinputWeights,initrecurrentWeights,initBiases);

% Define the custom cost function that evaluates the loss for given weights
and biases
CostFunction = @(x) modelGradients(x, dlnet, XTrain, YTrain); %x=parameters

% Run the Bee Algorithm
nVar = sum(cellfun(@numel, dlnet.Learnables{:, 'Value'}));  % Number of
Decision Variables

VarSize = [1 nVar];   % Decision Variables Matrix Size

VarMin = -0.1;        % Decision Variables Lower Bound
```

```
VarMax = 0.1;         % Decision Variables Upper Bound

%% Bees Algorithm Parameters

MaxIt = 20;              % Maximum Number of Iterations

nScoutBee = 30;          % Number of Scout Bees (n)

nSelectedSite = 5;       % Number of Selected Sites (m)

nEliteSite = 1;          % Number of Selected Elite Sites (e)

nSelectedSiteBee = 5;    % Number of Recruited Bees for Selected Sites (nsp)

nEliteSiteBee = 10;      % Number of Recruited Bees for Elite Sites (nep)

r = 0.1*(VarMax-VarMin);          % Neighborhood Radius

rdamp = 0.95;                     % Neighborhood Radius Damp Rate

%% Initialization

% Empty Bee Structure
empty_bee.initParameters = [];
empty_bee.Cost = [];

% Initialize Bees Array
bee = repmat(empty_bee, nScoutBee, 1);

% Create New Solutions
for i = 1:nScoutBee
    bee(i).initParameters = unifrnd(VarMin, VarMax, VarSize);
    bee(i).Cost = CostFunction(bee(i).initParameters);
end

%extract data from dlarray
beeCosts = cellfun(@extractdata,{bee.Cost});

% Sort
[~, SortOrder] = sort(beeCosts);
bee = bee(SortOrder);

% Update Best Solution Ever Found
BestSol = bee(1);

% Array to Hold Best Cost Values
BestCost = zeros(MaxIt, 1);

%% Bees Algorithm Main Loop

% Initialize Adam optimizer states
m = zeros(VarSize);
v = zeros(VarSize);

% Initialize Adam optimizer parameters
learnRate = 0.001;
beta1 = 0.9;
beta2 = 0.999;
epsilon = 1e-8;

checkpointPath = "/rds/projects/p/phamdt-autoreman-deep-learning";

for it = 1:MaxIt

    % Elite Sites
    for i = 1:nEliteSite
```

```
        bestnewbee.Cost = inf;

        for j = 1:nEliteSiteBee
            % Perform Bee Dance to get new parameters
            newParameters = PerformBeeDance(bee(i).initParameters, r);

            % Compute gradients and loss using your 'modelGradients' function
            [gradients, loss] = modelGradients(newParameters, dlnet, XTrain,
YTrain);

            % Update parameters using Adam optimization
            [newParameters, m, v] = updateParametersAdam(newParameters,
gradients, m, v, beta1, beta2, epsilon, learnRate, it);

            % Evaluate new parameters
            newbee.initParameters = newParameters;
            newbee.Cost = CostFunction(newbee.initParameters);  % You might
use 'loss' directly here if it's equivalent to your CostFunction

            if newbee.Cost < bestnewbee.Cost
                bestnewbee = newbee;
            end
        end
    end

   % Selected Non-Elite Sites
    for i = nEliteSite+1:nSelectedSite

        bestnewbee.Cost = inf;

        for j = 1:nSelectedSiteBee
            newbee.initParameters = PerformBeeDance(bee(i).initParameters, r);
            newbee.Cost = CostFunction(newbee.initParameters);
            if newbee.Cost<bestnewbee.Cost
                bestnewbee = newbee;
            end
        end

        if bestnewbee.Cost<bee(i).Cost
            bee(i) = bestnewbee;
        end

    end

    % Non-Selected Sites
    for i = nSelectedSite+1:nScoutBee
        bee(i).initParameters = unifrnd(VarMin, VarMax, VarSize);
        bee(i).Cost = CostFunction(bee(i).initParameters);
    end

    %extract data from dlarray
    beeCosts = cellfun(@extractdata,{bee.Cost});

    % Sort
    [~, SortOrder] = sort(beeCosts);
    bee = bee(SortOrder);


    % Update Best Solution Ever Found
    BestSol = bee(1);

    % Store Best Cost Ever Found
    BestCost(it) = BestSol.Cost;

     Time(it) = toc;
```

```
    % Display Iteration Information
    disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCost(it)) ' -
-> Time = ' num2str(Time(it)) ' seconds']);

    % Damp Neighborhood Radius
    r = r*rdamp;

    %save checkpoint of Iteration
    checkpointFilename = fullfile(checkpointPath,
sprintf('beeAlgorithm_checkpoint_iteration_%d.mat', it));
    save(checkpointFilename, 'bee', 'BestSol', 'BestCost', 'it');

end

%Results
save('best_bee_found_CNN_LSTM.mat', "BestSol");
figure;
semilogy(BestCost, 'LineWidth', 2);
xlabel('Iteration');
ylabel('Best Loss');

% After the Bee Algorithm has finished, extract the best solution
bestParameters = BestSol.initParameters;

% Convert the best solution to weights and biases
[bestweights, bestoffsets, bestscales, bestinputWeights,
bestrecurrentWeights, bestbiases] = paramToWeightsBiases(bestParameters,
dlnet.Learnables);

% Assign the best weights and biases to the deep learning model
dlnet = assignLearnables(dlnet, bestweights, bestoffsets, bestscales,
bestinputWeights, bestrecurrentWeights, bestbiases);

layers = replaceWeightsBiases(layers, dlnet.Learnables);

% Train the model with the best weights and biases using 'trainNetwork'
options = trainingOptions('adam', ...
    'MaxEpochs', 100, ...
    'InitialLearnRate', 0.0018, ...
    'MiniBatchSize', 32, ...
    'Verbose', 0, ...
    'Plots', 'training-progress');

%convert double data to categorical YTrain
classes = [0,1];
for i= 1:length(YTrain)
    YTrain{i} = categorical(YTrain{i},classes);
end

% Train the model with the best weights and biases using 'trainNetwork'
net = trainNetwork(XTrain, YTrain, layers, options);


function [weights,offsets,scales,inputWeights, recurrentWeights, biases] =
extractWeightsBiases(dlnet)
    learnables = dlnet.Learnables;
    weights = {};
    offsets = {};
    scales = {};
    inputWeights ={};
    recurrentWeights = {};
    biases = {};

    for i = 1:height(learnables)
        if strcmp(learnables.Parameter(i), 'Weights')
            weights{end+1} = gather(extractdata(learnables.Value{i}));
```

```
        elseif strcmp(learnables.Parameter(i), 'offset')
            offsets{end+1} = gather(extractdata(learnables.Value{i}));
        elseif strcmp(learnables.Parameter(i), 'scale')
            scales{end+1} = gather(extractdata(learnables.Value{i}));
        elseif strcmp(learnables.Parameter(i), 'InputWeights')
            inputWeights{end+1} = gather(extractdata(learnables.Value{i}));
        elseif strcmp(learnables.Parameter(i), 'RecurrentWeights')
            recurrentWeights{end+1} =
gather(extractdata(learnables.Value{i}));
        elseif strcmp(learnables.Parameter(i), 'Bias')
            biases{end+1} = gather(extractdata(learnables.Value{i}));

        end
    end
end


function parameters = weightsBiasesToParam(weights,Offsets,
Scales,inputWeights,recurrentWeights,biases)
    parameters = [];
    for i = 1:numel(weights)
        parameters = [parameters; weights{i}(:)];
    end
    for i = 1:numel(Offsets)
        parameters = [parameters; Offsets{i}(:)];
    end
    for i = 1:numel(Scales)
        parameters = [parameters; Scales{i}(:)];
    end
    for i = 1:numel(inputWeights)
        parameters = [parameters; inputWeights{i}(:)];
    end
    for i = 1:numel(recurrentWeights)
        parameters = [parameters; recurrentWeights{i}(:)];
    end
    for i = 1:numel(biases)
        parameters = [parameters; biases{i}(:)];
    end
end


function [gradients, loss] = modelGradients(parameters, dlnet, XTrain,
YTrain)
    % Reshape parameters into weights and biases
    [weights, offsets, scales, inputWeights, recurrentWeights, biases] =
paramToWeightsBiases(parameters, dlnet.Learnables);


    % Assign weights and biases to the dlnet
    dlnet = assignLearnables(dlnet, weights, offsets, scales, inputWeights,
recurrentWeights, biases);

    % Perform forward and backward passes and compute gradients
    [gradients, loss] = dlfeval(@modelGradientsLoss, dlnet, XTrain, YTrain);
end


function [loss,gradients] = modelGradientsLoss(dlnet, XTrain, YTrain)

% Arrange your data into Sequence Time Batch (STB) format and then convert to
dlarray.
numObservations = numel(XTrain);
sequenceLength = cellfun(@(x) size(x,2), XTrain);  % Get sequence lengths for
each observation
XTrain = dlarray([XTrain{:}], 'CTB');  % concatenate along the third
dimension
```

```
% Do the same for YTrain
numObservationsY = numel(YTrain);
sequenceLengthY = cellfun(@(x) size(x,2), YTrain);  % Get sequence lengths
for each observation

% Adjust YTrain values so they start from 1
YTrain = cellfun(@(x) x+1, YTrain, 'UniformOutput', false);

% Convert YTrain to one-hot encoding
YTrainOneHot = cell(numObservationsY, 1);
for i = 1:numObservationsY
    YTrainOneHot{i} = full(ind2vec(YTrain{i}));
end

YTrain = dlarray([YTrainOneHot{:}], 'CTB');

% Forward data through network.
dlYPred = forward(dlnet,XTrain);

% Check the size of dlYPred and YTrain
disp("Size of dlYPred: ");
disp(size(dlYPred));
disp("Size of YTrain: ");
disp(size(YTrain));

% Calculate cross-entropy loss.
loss = crossentropy(dlYPred,YTrain);

% Calculate gradients of loss with respect to learnable parameters.
gradients = dlgradient(loss,dlnet.Learnables);

end


function dlnet = assignLearnables(dlnet, weights, offsets, scales,
inputWeights, recurrentWeights, biases)
    numLayers = numel(dlnet.Layers);
    newLearnables = table('Size',[0
3],'VariableTypes',{'string','string','cell'},'VariableNames',{'Layer','Param
eter','Value'});
    weightsIndex = 1;
    biasesIndex = 1;
    scalesIndex = 1;
    offsetsIndex = 1;
    inputWeightsIndex = 1;
    recurrentWeightsIndex = 1;
    for i = 1:numLayers
        layer = dlnet.Layers(i);
        if isa(layer,'nnet.cnn.layer.BatchNormalizationLayer')
            if scalesIndex <= numel(scales)
                newRow = {layer.Name, 'Scale',
{dlarray(single(scales{scalesIndex}))}};
                newLearnables = [newLearnables; newRow];
                scalesIndex = scalesIndex + 1;
            end
            if offsetsIndex <= numel(offsets)
                newRow = {layer.Name, 'Offset',
{dlarray(single(offsets{offsetsIndex}))}};
                newLearnables = [newLearnables; newRow];
                offsetsIndex = offsetsIndex + 1;
            end
        end
        if isprop(layer, 'Weights') && weightsIndex <= numel(weights)
            %disp(['Setting Weights for layer: ', layer.Name]);
            %disp(['Expected size: ', mat2str(size(layer.Weights))]);
            %disp(['Received size: ', mat2str(size(weights{weightsIndex}))]);
```

```
            newRow = {layer.Name, 'Weights',
{dlarray(single(weights{weightsIndex}))}};
            newLearnables = [newLearnables; newRow];
            weightsIndex = weightsIndex + 1;
        end
        if isprop(layer, 'InputWeights') && inputWeightsIndex <=
numel(inputWeights)
            %disp(['Setting InputWeights for layer: ', layer.Name]);
            %disp(['Expected size: ', mat2str(size(layer.InputWeights))]);
            %disp(['Received size: ',
mat2str(size(inputWeights{inputWeightsIndex}))]);
            newRow = {layer.Name, 'InputWeights',
{dlarray(single(inputWeights{inputWeightsIndex}))}};
            newLearnables = [newLearnables; newRow];
            inputWeightsIndex = inputWeightsIndex + 1;
        end
        if isprop(layer, 'RecurrentWeights') && recurrentWeightsIndex <=
numel(recurrentWeights)
            %disp(['Setting RecurrentWeights for layer: ', layer.Name]);
            %disp(['Expected size: ',
mat2str(size(layer.RecurrentWeights))]);
            %disp(['Received size: ',
mat2str(size(recurrentWeights{recurrentWeightsIndex}))]);
            newRow = {layer.Name, 'RecurrentWeights',
{dlarray(single(recurrentWeights{recurrentWeightsIndex}))}};
            newLearnables = [newLearnables; newRow];
            recurrentWeightsIndex = recurrentWeightsIndex + 1;
        end
        if isprop(layer, 'Bias') && biasesIndex <= numel(biases)
            %disp(['Setting Biases for layer: ', layer.Name]);
            %disp(['Expected size: ', mat2str(size(layer.Bias))]);
            %disp(['Received size: ', mat2str(size(biases{biasesIndex}))]);
            newRow = {layer.Name, 'Bias',
{dlarray(single(biases{biasesIndex}))}};
            newLearnables = [newLearnables; newRow];
            biasesIndex = biasesIndex + 1;
        end
    end
    dlnet.Learnables = newLearnables;
end


function newPosition = PerformBeeDance(currentPosition, r)
    % Generate a random number within the range [-r, r]
    randomShift = 2 * r * rand(size(currentPosition)) - r;

    % Calculate newPosition based on currentPosition and randomShift
    newPosition = currentPosition + randomShift;
end


function [updatedParameters, m, v] = updateParametersAdam(parameters,
gradients, m, v, beta1, beta2, epsilon, learningRate, t)
    % Adam optimization for updating parameters

    % Calculate the moving averages of the gradients
    m = beta1 .* m + (1 - beta1) .* gradients;

    % Calculate the moving averages of the squared gradients
    v = beta2 .* v + (1 - beta2) .* gradients.^2;

    % Compute bias-corrected estimates of the moving averages
    mHat = m ./ (1 - beta1.^t);
    vHat = v ./ (1 - beta2.^t);
```

```
    % Update the parameters
    updatedParameters = parameters - learningRate .* mHat ./ (sqrt(vHat) +
epsilon);
end


function [weights,offsets,scales,inputWeights,recurrentWeights,biases] =
paramToWeightsBiases(parameters, learnables)

    weights = {};
    scales = {};
    offsets = {};
    inputWeights = {};
    recurrentWeights = {};
    biases = {};
    currentParamIdx = 1;
    currentWeightIdx = 1;
    currentScaleIdx = 1;
    currentOffsetIdx = 1;
    currentInputWeightsIdx = 1;
    currentRecurrentWeightsIdx = 1;
    currentBiasIdx = 1;

    for i = 1:size(learnables, 1)
        if string(learnables.Layer(i)) ~= "batchnorm" &&
string(learnables.Parameter(i)) == "Weights"
            weightSize = size(learnables.Value{i});
            numWeightElements = prod(weightSize);
            if currentParamIdx + numWeightElements - 1 <= numel(parameters)
                weight = parameters(currentParamIdx : currentParamIdx +
numWeightElements - 1);
                weight = reshape(weight, weightSize);
                weights{currentWeightIdx} = weight;
                currentWeightIdx = currentWeightIdx + 1;
                currentParamIdx = currentParamIdx + numWeightElements;
            else
                error('Parameters do not have enough elements for the weights
of layer %d.', i);
            end
        elseif string(learnables.Layer(i)) ~= "batchnorm" &&
string(learnables.Parameter(i)) == "InputWeights"
            weightSize = size(learnables.Value{i});
            numInputWeightElements = prod(weightSize);
            if currentParamIdx + numInputWeightElements - 1 <=
numel(parameters)
                inputWeight = parameters(currentParamIdx : currentParamIdx +
numInputWeightElements - 1);
                inputWeight = reshape(inputWeight, weightSize);
                inputWeights{currentInputWeightsIdx} = inputWeight;
                currentInputWeightsIdx = currentInputWeightsIdx + 1;
                currentParamIdx = currentParamIdx + numInputWeightElements;
            else
                error('Parameters do not have enough elements for the input
weights of layer %d.', i);
            end
        elseif string(learnables.Layer(i)) ~= "batchnorm" &&
string(learnables.Parameter(i)) == "RecurrentWeights"
            weightSize = size(learnables.Value{i});
            numRecurrentWeightElements = prod(weightSize);
            if currentParamIdx + numRecurrentWeightElements - 1 <=
numel(parameters)
                recurrentWeight = parameters(currentParamIdx :
currentParamIdx + numRecurrentWeightElements - 1);
                recurrentWeight = reshape(recurrentWeight, weightSize);
                recurrentWeights{currentRecurrentWeightsIdx} =
recurrentWeight;
```

```
                currentRecurrentWeightsIdx = currentRecurrentWeightsIdx + 1;
                currentParamIdx = currentParamIdx +
numRecurrentWeightElements;
            else
                error('Parameters do not have enough elements for the
recurrent weights of layer %d.', i);
            end
        elseif string(learnables.Layer(i)) ~= "batchnorm" &&
string(learnables.Parameter(i)) == "Bias"
            biasSize = size(learnables.Value{i});
            numBiasElements = prod(biasSize);
            if currentParamIdx + numBiasElements - 1 <= numel(parameters)
                bias = parameters(currentParamIdx : currentParamIdx +
numBiasElements - 1);
                bias = reshape(bias, biasSize);
                biases{currentBiasIdx} = bias;
                currentBiasIdx = currentBiasIdx + 1;
                currentParamIdx = currentParamIdx + numBiasElements;
            else
                error('Parameters do not have enough elements for the bias of
layer %d.', i);
            end
        elseif string(learnables.Layer(i)) == "batchnorm"
            if string(learnables.Parameter(i)) == "Scale"
                scaleSize = size(learnables.Value{i});
                numScaleElements = prod(scaleSize);
                if currentParamIdx + numScaleElements - 1 <=
numel(parameters)
                    scale = parameters(currentParamIdx : currentParamIdx +
numScaleElements - 1);
                    scale = reshape(scale, scaleSize);
                    scales{currentScaleIdx} = scale;
                    currentScaleIdx = currentScaleIdx + 1;
                    currentParamIdx = currentParamIdx + numScaleElements;
                else
                    error('Parameters do not have enough elements for the
scale of layer %d.', i);
                end
            elseif string(learnables.Parameter(i)) == "Offset"
                offsetSize = size(learnables.Value{i});
                numOffsetElements = prod(offsetSize);
                if currentParamIdx + numOffsetElements - 1 <=
numel(parameters)
                    offset = parameters(currentParamIdx : currentParamIdx +
numOffsetElements - 1);
                    offset = reshape(offset, offsetSize);
                    offsets{currentOffsetIdx} = offset;
                    currentOffsetIdx = currentOffsetIdx + 1;
                    currentParamIdx = currentParamIdx + numOffsetElements;
                else
                    error('Parameters do not have enough elements for the
offset of layer %d.', i);
                end
            end
        end
    end
end


function layers = replaceWeightsBiases(layers, learnables)
    for i = 1:numel(layers)
        layer = layers(i);
        if isprop(layer, 'Name') % Check if layer has a 'Name' property
            name = layer.Name;
            idxWeights = find(learnables.Layer == name & learnables.Parameter
== "Weights");
```

```
            idxBias = find(learnables.Layer == name & learnables.Parameter ==
"Bias");
            idxOffset = find(learnables.Layer == name & learnables.Parameter
== "Offset");
            idxScale = find(learnables.Layer == name & learnables.Parameter
== "Scale");
            idxInputWeights = find(learnables.Layer == name &
learnables.Parameter == "InputWeights");
            idxRecurrentWeights = find(learnables.Layer == name &
learnables.Parameter == "RecurrentWeights");

            if ~isempty(idxWeights) && isprop(layer, 'Weights')
                layer.Weights = learnables.Value{idxWeights};
            end

            if ~isempty(idxBias) && isprop(layer, 'Bias')
                layer.Bias = learnables.Value{idxBias};
            end

            if ~isempty(idxOffset) && isprop(layer, 'Offset')
                layer.Offset = learnables.Value{idxOffset};
            end

            if ~isempty(idxScale) && isprop(layer, 'Scale')
                layer.Scale = learnables.Value{idxScale};
            end

            if ~isempty(idxInputWeights) && isprop(layer, 'InputWeights')
                layer.InputWeights = learnables.Value{idxInputWeights};
            end

            if ~isempty(idxRecurrentWeights) && isprop(layer,
'RecurrentWeights')
                layer.RecurrentWeights =
learnables.Value{idxRecurrentWeights};
            end

            layers(i) = layer;
        end
    end
end

%Test CNN_LSTM Network
load('B13_XTest');
load('B13_YTest01');

%Classify the test data using classify
YPred = classify(net,XTest);

%Compare the predictions with the test data by using a plot
figure
plot(YPred{1},'.-')
hold on
plot(YTest{1})
hold off

xlabel("Time Step")
ylabel("Activity")
title("Predicted Activities")
legend(["Predicted" "Test Data"])

%Calculate the accuracy of the predictions
accuracy = mean(YPred{1} == YTest{1});
```