A DEEP LEARNING APPROACH TO COLLISION DETECTION IN NUCLEAR POWER PLANT REACTOR COOLANT SYSTEM USING VIBRATION DATA

CHUA KAI ZER

UNIVERSITI TEKNOLOGI MALAYSIA

PSZ 19:16 (Pind. 1/13)

UNIVERSITI TEKNOLOGI MALAYSIA

DECLARATION OF THESIS / UNDERGRADUATE PROJECT REPORT AND COPYRIGHT

Author's full name : CHUA KAI ZER

Date of Birth : 09 FEB 1998

Title : A DEEP LEARNING APPROACH TO COLLSION DETECTION IN NUCLEAR POWER PLANT COOLANT SYSTEM USING VIBRATION DATA

Academic Session : 2021/2022

I declare that this thesis is classified as:

☐ **CONFIDENTIAL** (Contains confidential information under the Official Secret Act 1972)*

☐ **RESTRICTED** (Contains restricted information as specified by the organization where research was done)*

☑ **OPEN ACCESS** I agree that my thesis to be published as online open access (full text)

1. I acknowledged that Universiti Teknologi Malaysia reserves the right as follows:
2. The thesis is the property of Universiti Teknologi Malaysia
3. The Library of Universiti Teknologi Malaysia has the right to make copies for the purpose of research only.
4. The Library has the right to make copies of the thesis for academic exchange.

Chua

SIGNATURE OF STUDENT

A18KT0047

MATRIX NUMBER

Date: 27 JANUARY 2022

Certified by:

SIGNATURE OF SUPERVISOR

Dr. NAHRUL KHAIR BIN ALANG MD RASHID

NAME OF SUPERVISOR

Date: 27 JANUARY 2022

NOTES : If the thesis is CONFIDENTIAL or RESTRICTED, please attach with the letter from the organization with period and reasons for confidentiality or restriction

“I hereby declare that we have read this thesis and in my opinion this thesis is suffcient in term of scope and quality for the award of the degree of Bachelor of Engineering (Nuclear)”

Signature 

Name of Supervisor : NAHRUL KHAIR BIN ALANG MD RAHSID

Date : 27 JANUARY 2022

A DEEP LEARNING APPROACH TO COLLISION DETECTION IN NUCLEAR POWER PLANT REACTOR COOLANT SYSTEM USING VIBRATION DATA

CHUA KAI ZER

A thesis submitted in fulfilment of the
requirements for the award of the degree of
Bachelor of Engineering (Nuclear)

School of Chemical and Energy Engineering
Faculty of Engineering
Universiti Teknologi Malaysia

JANUARY 2022

DECLARATION

I declare that this thesis entitled *"A Deep Learning Approach to Collision Detection in Nuclear Power Plant Coolant System using Vibration Data"* is the result of my own research except as cited in the references. The thesis has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Signature : 

Name : CHUA KAI ZER

Date : 27 JANUARY 2022

DEDICATION

This thesis is dedicated to my father, who taught me that the best kind of knowledge to have is that which is learned for its own sake. It is also dedicated to my mother, who taught me that even the largest task can be accomplished if it is done one step at a time.

ACKNOWLEDGEMENT

First and foremost, I would like to express my sincere gratitude to Prof. Ir. Dr. Nahrul Khair Bin Alang Md Rashid for giving me an opportunity to work on this project under his supervision. I am deeply indebted to him for his expertise sharing in the field of Artificial Intelligence (AI) and his guidance and patience extended to me throughout the project. I am also grateful to have good health and wellbeing to complete this study.

Finally, I would like to thank my beloved family, firends and coursemates for their supports, encouragement and suggestions which had directly or indirectly lent me a helping hand throughout this project especially during this pandemic

ABSTRACT

In recent years, the adoption of intelligent systems in fault detection and diagnosis (FFD) has expanded into the fields of nuclear engineering especially in structural damage identification, defect tracking of instruments, and other signal processing operations. In this study, the proposed Patch-Based 1D Convolutional Neural Network is subject to two training phases, namely local pre-training and global fine-tuning that allow the model to learn both local and generic features for all-at-once collision parameters prediction in reactor coolant system by using the waveform simulation acceleration data generated by four accelerometers. Besides, a multi-model approach including the data aggregation aided machine learning (ML) algorithms and simplified 1D Convolutional Neural Network for many-to-one prediction of each collision parameter are also implemented as additional model options to achieve the main objective of the study which is to select the best performing model under the percentage mean absolute error metric.

ABSTRAK

Kebelakangani ini, penerapan sistem kecerdasan dalam bidang kejuruteraan nuklear melalui kaedah-kaedah pengesanan dan diagnosis kesalahan semakin berkembang dalam pengesanan dan penjejakan kerosakan struktur, kecacatan komponen dan juga operasi pemprosesan isyarat yang lain. Dalam kajian ini, model Rangkaian Saraf Konvolusi 1D Berpetak telah menjalani dua fasa latihan, iaitu pra latihan secara tempatan dan juga penalaan secara global yang membolehkan pemebelajaran kedua-dua ciri tempatan dan generik dalam peramalan parameter perlanggaran secara keseluruhan dalam sistem penyejukan reactor dengan menggunakan data gelombang yang dihasilkan oleh empat akselerometer. Selain itu, satu pendekatan multi-model yang merangkumi algoritma pembelajaran mesin dibantu agregasi data dan Rangkaian Saraf Konvolusi 1D dipermudahkan demi peramalan setiap parameter perlanggaran secara banyak ke satu juga diimplementasikan sebagai model pilihan untuk mencapai objektif utama kajian dalam memilih model yang berprestasi terbaik bawah metrik peratusan min ralat mutlak.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

1DCNN	-	1-Dimensional Convolutional Neural Network
2DCNN	-	2-Dimensional Convolutional Neural Network
AE	-	Autoencoders
AI	-	Artificial Intelligence
ANFIS	-	Adaptive Neuro-Fuzzy Inference System
ANN	-	Artificial Neural Network
BN	-	Batch Normalization
CART	-	Decision Tree Regressor
CNN	-	Convolutional Neural Network
DBN	-	Deep Belief Network
DFT	-	Discrete Fourier Transform
DNN	-	Deep Neural Network
ELU	-	Exponential Linear Unit
EN	-	Elastic Net
FDD	-	Fault Detection and Diagnosis
FFT	-	Fast Fourier Transform
GAN	-	Generative Adversarial Network
GBM	-	Gradient Boosting Regressor
GRNN	-	General Regression Neural Network
IE	-	Initiating Event
IoT	-	Internet of Things
KAERI	-	Korea Atomic Energy Research Institute

KNN	-	K-Nearest Neighbour
LR	-	Linear Regression
LReLU	-	Leaky Rectified Linear Unit
LSTM	-	Long Short-Term Memory
MAPE	-	Mean Absolute Pecentage Error
ML	-	Machine Learning
MLP	-	Multilayer Perceptron
MSE	-	Mean Squared Error
NPP	-	Nuclear Power Plant
PReLU	-	Parametric Rectified Linear Unit
RCS	-	Reactor Coolant System
ReLU	-	Rectified Linear Unit
RNN	-	Recurrent Neural Network
SELU	-	Scaled Exponential Linear Unit
SSD	-	Silicon Drift Detector
SVM	-	Support Vector Machine
UPN	-	Unsupervised Pretrained Networks

LIST OF SYMBOLS

X	-	x-coordinate of collision point
Y	-	y-coordinate of collision point
M	-	Mass of collision body
V	-	Velocity of collision body
z	-	Weighted sum
y	-	Output
x	-	Input value
w	-	Connection weight
b	-	Bias
φ	-	Activation function
$\hat{y}$	-	Predicted Output
C	-	Loss Function
η	-	Learning rate
a	-	Activation
σ	-	Standard deviation
fan_{in}	-	Layer input
fan_{out}	-	Layer output
fan_{avg}	-	Average input-output layer
α	-	Scalable hyperparameter
λ	-	Scalable hyperparameter

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Research Background

With the rapid advancements in computational performance, computer data storage capacity, big data analytics, and information technologies such as the internet of things (IoT), artificial intelligence (AI), and cloud computation enables the realization of the vision and the potential of Industry 4.0 to achieve a higher level of automation in various industries. The era of Industry 4.0 in the future is expected to accelerate the growth of electricity demand due to the shifts of industries towards more energy-intensive smart systems operated sectors. In association with that, nuclear power has become increasingly important not only as one of the most efficient energy suppliers to counteract the increasing demand for electricity but also a low-emitting source to overcome environmental risks, particularly air pollution due to greenhouse gas and carbon emissions. In 2017, a related study showed that there were 447 nuclear power reactors built in 31 countries with a total installed capacity of more than 390GW additive of 60 units under construction and 160 units that were planned to be built (Yukiya, 2017).

The nuclear power system is a man-machine-network integration system that shows high complexity in its construction, operation, and other aspects. As a result, there exists a variety of risks and major concerns for the current nuclear power plant (NPP). Firstly, the complex system itself creates a relatively high barrier to its operators. According to Chao et al. (2020) in their study, various failures can occur may occur in equipment, instruments, or even system processes in NPP, greatly impacting both the performance and security of the NPP. Due to the insufficient automation level of NPP, system control and management are generally carried out manually by human operators under great pressures because of high control requirements, thus making NPP systems vulnerable to human errors (Chao et al., 2020)

which may lead to fatal transients and accidents. However, the problems above can be addressed by the application of AI technologies as auxiliary systems in the management of NPP systems.

For the last few decades, machine learning and deep learning techniques have proven successful and there has been major interest towards the implementation of artificial neural network (ANN) based fault detection and diagnosis (FDD) methods to improve safety which is of the prime importance of an NPP. More and more resources are directed in optimizing plant operating systems in power generation industries. A typical example is the convolutional neural network (CNN) which has dominated the research field of computer vision, outperforming humans in various areas. Thus, it is believed that deep learning methods can be a promising direction to transform and assist NPP operators in FDD.

1.2 Problem Statement

A nuclear power plant (NPP) is a highly complex machine that is operated and monitored manually by personnel with expertise in the field of nuclear engineering. During the operation of NPP, transients will occur and they may be due to malfunctioning of equipment, external or internal disturbance to the system, or process system failure. When facing a transient under undesired plant condition generally known as the initiating event (IE), where the NPP is departing to abnormal state from its normal state, it is crucially important for the operators to diagnose, identify and carry out necessary corrective actions to prevent it from further developing into a potentially serious and uncontrollable nuclear accident (Chao et al., 2020).

In the reactor coolant system (RCS) of NPP, debris and impurities may build up over time and there is a tendency for the equipment in plant piping system to suffer extensive damage from the propulsion of the collision bodies formed within the system. Depending on the seriousness of the collision event, the readings of an instrument, in this case, the vibration signal generated by the accelerometers might not give a clear reflection and indication of the collision event. This may cause

misidentification of transients and the severity of the collision, resulting in imprecise action by the operators which may lead to an accident. This situation necessitates the development of a system to assist NPP operators to identify and keep track of the collision events gradually from their earliest stage of evolution.

1.3 Objective of Project

The primary objective of the project is to develop an auxiliary neural network-based framework to predict important parameters of a collision event in RCS from accelerometer vibration data. The specific objectives to be carried out in order to accomplish the main objective are:

a) To evaluate and choose the most suitable data processing method.

b) Architectural design and initialization of a deep neural network (DNN) and model selection with the aid of hyperparameter tuning.

c) To experiment and identify the best performing model for collision parameter prediction based on the vibration dataset used.

1.4 Scope of Project

In this project, the simulator-generated vibration data for collision prediction provided by Korea Atomic Energy Research Institute (KAERI) (2020) is chosen as the training dataset. The dataset is comprised of time-series acceleration data from accelerometers and the target features to be predicted for each collision event. The architectures of deep neural networks from previous studies on time-series data analysis are revised and evaluated based on their strengths and weaknesses. Several methods of data processing are proposed. The processed data are split with a train-test ratio of 8:2 and tested on a few deep learning models designed and built entirely in Python using TensorFlow-Keras with the aid of open-source Python packages such as

Pandas, Numpy, Matplotlib, and Scikit-learn. Hyper-parameter tuning technique of Bayesian Optimization is performed using only the train data on each model and the process is kept under supervision to maintain a balance between the train and validation scores for each epoch to prevent overfitting and underfitting of data. The tuned model will be trained and tested on train and test data respectively using the mean squared percentage error (MAPE) regression metrics. The collision parameters to be predicted are collision point (X, Y), the mass (M), and velocity (V) of the body upon collision.

1.5 Significance of Project

The aim of this study is to examine the applicability of the proposed ANN-based model to be integrated into the NPP safety system, specifically in RCS. Due to large quantities of numeric, symbolic, and quantitative information during routine NPP operations (Chao et al., 2020), operators may find difficulties in handling and controlling the process parameters and system interactions, especially during abnormal or emergency situations. The main goal of the project is to reduce the reliance of NPP safety systems, particularly in RCS on human intervention so to mitigate the operational errors due to varying degrees of external influence on operator performance in decision making. The proposed model is expected to extract, convert and assimilate the raw vibration data from accelerometers to bring meaningful knowledge to the operators.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

In this chapter, an overview of both the basic working principles and algorithms of Artificial Neural Network (ANN) will be presented. The theory and concept of deep learning along with its types of learning algorithms are also discussed. Based on the previously published study and research, the superiorities of deep learning approaches in vibration-based Fault Detection and Diagnosis (FDD) are also reviewed.

2.2 Biological Neurons

As part of the evolutionary history of deep learning models, the early study of biological neural networks in the human brain had propelled the artificial intelligence research frontier forward. Biologically, neural networks are made up of special biological cells called neurons that are responsible for processing information. Interconnected by neuronal junctions called synapse, a biological neuron (Figure 2.1) has a cell body responsible for processing incoming signals from the dendrite. The processed signals are transmitted to relevant cells by axon through the secretion of neurotransmitters.

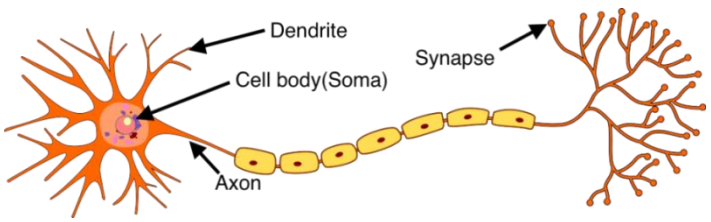

Figure 2.1 A Generic Structure of Biological Neuron

2.3 The Perceptron

A perceptron (neuron or node) is a bio-inspired single mathematical information-processing unit. It takes in one or more input signals from synapses of nodes situated at the previous layer which are characterized by specific weights and bias. The bias is applied externally as an auxiliary node which can promote better data fitting. At the summing junction, as shown in Figure 2.2, the sum of the product of input values and their associated synaptic weights is computed. The calculated weighted sum is then fed into an appropriate activation function to make modifications to the learned pattern (Nwankpa et al., 2018). This is done by squeezing the value to a finite permissible amplitude range. The typical target intervals are [0, 1] and [-1, 1]. The activation function returns a final value that is then passed as an input value to the succeeding layer or as the final output in the case where the perceptron is the output node of the network.

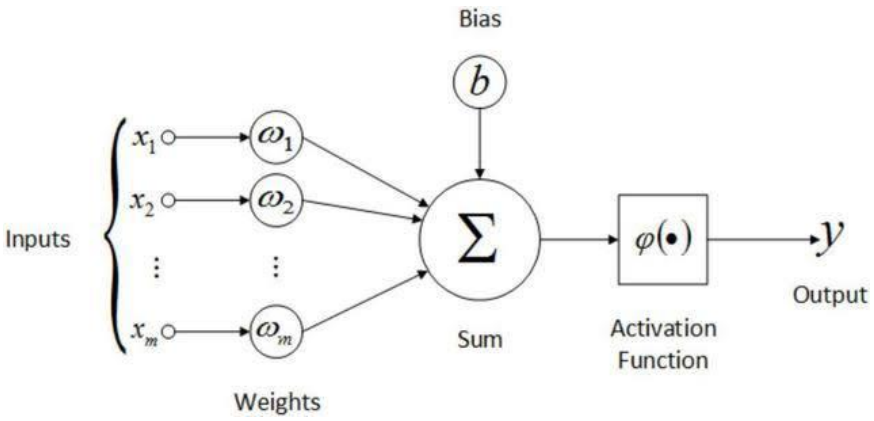

Figure 2.2 Simplified Model of a Perceptron

In mathematical terms, the relationship between the weighted sum, and the output, can be shown in equations (2.1) and (2.2) as:

$$z = w_1x_1 + w_2x_2 + \ldots + w_{m-1}x_{m-1} + w_mx_m + b \quad (2.1)$$

$$y = \varphi(z) \quad (2.2)$$

where,

x_n = the n-th input value of the current instance

w_n = the connection weight between n-th input neuron and the current output neuron

b = the current layer bias

φ = the activation function

2.4 Artificial Neural Network

Artificial Neural Network (ANN) is an interconnection of multiple bio-inspired perceptrons that are ordered into layers in which its process of activation computation is usually done in a single forward direction. ANN comprises processing elements that receive information and produce output based on their learned parameters such as weights, biases, and their pre-defined network architectures and activation functions. ANN is very famous together with backpropagation as its training algorithm (Svozil et al, 1997) and typically consists of hundreds or thousands of interconnecting simple processing units called the artificial neurons (nodes) which are structured and programmed in such a way to mimic the working of the human biological nervous system. Generally, the ANN is organized and arranged in three types of layers:

a) Input layer: A passive layer that receives inputs from explanatory variables and communicates with one or multiple hidden layers without changing the data values.

b) Hidden layer: A layer with weighted connection that processes values entering its nodes as per the pre-assigned activation functions.

c) Output layer: A layer that accepts as input and manipulates the output values from hidden layers to ultimately produce a prediction that corresponds to the response variable.

The nodes in the input layer will supply respective elements from the input vector to nodes in the second layer. The output values of the second layer will be the inputs of third layer and so on for the rest of the network. In typical neural network design as shown in Figure 2.3, more than one hidden layer is used and the network is fully connected in which every node within a layer and its preceding layer is interconnected.

Thus, the activation of each predecessor reflects certain properties from its ancestors in the previous layers which is controlled by the degree of significance for given connections, the weight coefficients (Svozil et al, 1997). The signals produced in the output layer are said to be the result of the overall network activation or response to the input signals supplied at the input layer.

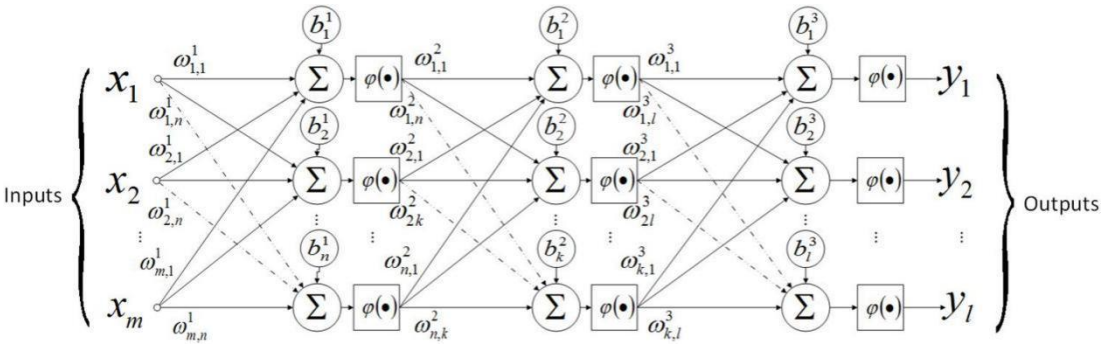

Figure 2.3 A Multilayer Artificial Neural Network

2.4.1 Activation Functions

Activation functions are mathematical functions used to process and appropriate the weighted sum of the input values to a finite range as a way to manipulate how activated the neuron should be and to decide if the neuron can be fired or not. The role of an activation function is highly dependent on its position in the neural network. It performs conversion of linear mappings into suitable non-linear forms for propagation when placed after any hidden layer while in the output layer, it makes a prediction (Nwankpa et al., 2018). Although deep neural networks develop better performances with the aid of activation functions, common problems like vanishing gradients and exploding gradients may arise, thus necessitating thorough activation function selection. Some of the most common activation functions are highlighted in the following subsections.

2.4.1.1 Sigmoid Function

The sigmoid function, also referred to as logistic function in some literature is one of the most popular and widely used activation functions in artificial neural networks (Nwankpa et al., 2018). Fundamentally, the function and its derivative are defined as

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

$$f'(x) = f(x)(1 - f(x)) \tag{2.4}$$

From equations (2.3) and (2.4) shown above, the sigmoid function is noticeably continuous, differentiable, and has a relatively small bounded interval of (0, 1) that may result in small output changes, presenting the problem of vanishing gradients during backpropagation which will be discussed later. While the sigmoid function is well known for its simplicity and has proven successful in problems like logistic regression, binary classification, and other domains in shallow networks, the function is computationally expensive due to its exponential nature (Datta, 2020). Since the sigmoid function is also a non-zero-centered activation function, the output of the function is always positive, greatly affecting the efficiency of weights updating during model training. In order to retain the existing benefits of sigmoid function along with zero-centered nature, the tanh or hyperbolic tangent function was introduced (Datta, 2020). Figure 2.4 (Appendix A) shows the graphical representation of the sigmoid function and its gradient.

2.4.1.2 Tanh Function

The hyperbolic tangent or tanh function though encountering both the same problem of vanishing gradients and expensive computational cost, have slightly higher popularity than sigmoid function due to its zero-centered nature that aids the

backpropagation process. The function and its derivative are defined by equations (2.5) and (2.6) as

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.5}$$

$$f(x) = 1 - f(x)^2 \tag{2.6}$$

Just like sigmoid function, the tanh function is continuous, differentiable but with a bounded range of (-1, 1). Figure 2.5 (Appendix A) shows the graphical representation of the Tanh function and its gradient.

2.4.1.3 ReLU Function

The rectified linear unit function or ReLU is the most widely used state-of-the-art activation function that has proved to outperform other activation functions including sigmoid and tanh function in terms of speed and generalizability. According to Krizhevsky et al. in 2012, the number of epochs required to train a network using ReLU is about six times less than that of sigmoid and tanh function. The ReLU is partially linear and thus preserves some properties of a linear model which allows better optimization especially with optimizer like gradient-descent (Nwankpa et al., 2018). The function is defined as

$$f(x) = max(0, x) \tag{2.7}$$

From equation (2.7), ReLU is represented as a threshold function which rectifies inputs with negative values by setting them to zero which possesses extremely low computational cost. ReLU function is not bounded, non zero-centered and not differentiable at $x = 0$. Since the derivatives of the function are always 0 (left) or 1 (right), the problem of vanishing gradients does not exist in ReLU. However, this makes the function vulnerable to the "dying ReLU" problem due to the multiplication of zero value derivatives during weight. A few variants of ReLU functions were

introduced in counter of the stated problem. Figure 2.6 (Appendix A) shows the graphical representation of the ReLU function and its gradient.

2.5 Deep Learning

Learning is a fundamental trait or ability of intelligence. Unlike humans, learning in artificial neural networks can be viewed as a process of updating the values of connection weights or network architecture in some models so to perform a specific task. During the learning process, the performance of a network is improved over time and its direction of weight updates are supervised and governed by the loss function set by human which maps the deviation of its output from the desired results. The weights and biases of the network are tuned in such a way that the loss function is reduced as much as possible. Given a collection of representative data, artificial neural networks appear to be able to learn underlying patterns connecting the inputs and outputs through deep layers in which some of them do not make sense and are impossible for a human to fully understand. This is one of the major advantages of deep learning over traditional systems that makes them successful in different fields.

2.5.1 Types of Learning Algorithm

In order to design a learning process, the model of the environment and the availability of information to the network must be set up. The model is referred to as the network learning algorithm. Basically, there are 2 main related learning algorithms to be discussed in which a neural network operates: supervised learning and unsupervised learning.

2.5.1.1 Supervised Learning

In supervised learning, the neural network is provided with both inputs and the desired outputs. The inputs are processed and their resulting outputs are directly compared against the given desired output. The error computed by the pre-defined loss function will then determine how the weights should be adjusted as it propagates back through the system. The process will go on over and over depending on the number of training epochs defined, with each step getting the model closer to making precise predictions. However, Maind and Wankar suggested that some neural networks do not learn well and converge as expected. This may be due to insufficient or low quality of the input data which does not contain the specific information from which the given desired output is derived (Maind and Wankar, 2014).

2.5.1.2 Unsupervised Learning

In unsupervised learning, the network learns without a teacher and only the inputs are provided. The desired outputs are absent and the system must explore the underlying correlations or hidden trends in the data structure itself (Maind and Wankar, 2014). From the insights gained, the system will perform feature engineering internally within its neural network to make a decision on classification problems in a self-organization and adaptation manner.

2.5.2 Backpropagation Training Algorithm

The backpropagation algorithm is well-known for its ability to monitor neural network learning and store the intrinsically learned mapping relations without the need to specifically disclose the related mathematical equation in advance. Depending on the types of neural networks involved, the training algorithm typically utilizes gradient descent to regulate and modify the weight and bias values of the network to achieve a minimum sum of squares error (Li et al., 2012). In this algorithm, the weights and biases are initialized by a pre-defined kernel initializer before the process starts. The

algorithm handles one mini-batch of 32 instances by default at a time until the full training set is gone through, indicating the passage of an epoch. According to (Li et al., 2012), the backpropagation learning process can be described by two main operations: the forward propagation of operating signal and the backward propagation of error signal. During the forward propagation, the outputs of all input neurons are computed and passed to the next layer as described before until the output of the last layer is obtained. The algorithm then compares the desired output, *y*, and the predicted output, $\hat{y}$ of the *j*th output neuron of the network using the typical sum of squares loss function, *C* in equation (2.8) as

$$C = \frac{1}{2}\sum_{j}(\hat{y}_j - y_j)^2 \tag{2.8}$$

The vector of loss function values for each instance in the mini-batch is calculated and averaged. According to the minimization method of steepest descent as illustrated in Figure 2.7, the fastest decline in loss function can be achieved by some modifications to the weight and bias values along with their respective negative gradient direction (Li et al., 2012). In association with that, the chain rule is applied to compute the changes of the loss function with respect to the weight, *w* and bias, *b* of the *k*th training step which will be used in equations (2.9) and (2.10) along with the learning rate, η to update the values of weight and bias.

$$w_{k+1} = w_k - \eta\frac{\partial C}{\partial w_k} \tag{2.9}$$

$$b_{k+1} = b_k - \eta\frac{\partial C}{\partial b_k} \tag{2.10}$$

During the process, the error contributions from each connection are measured. The weights and biases remain constant during the loss function gradients computation with regards to each parameter of each instance within the mini-batch. Finally, during the error signals backpropagation, a gradient descent step is performed in order to tweak all the connection weights and biases in the network by using the gradients of

loss functions computed. The next mini-batch repeats the same process but with newly updated weights and biases.

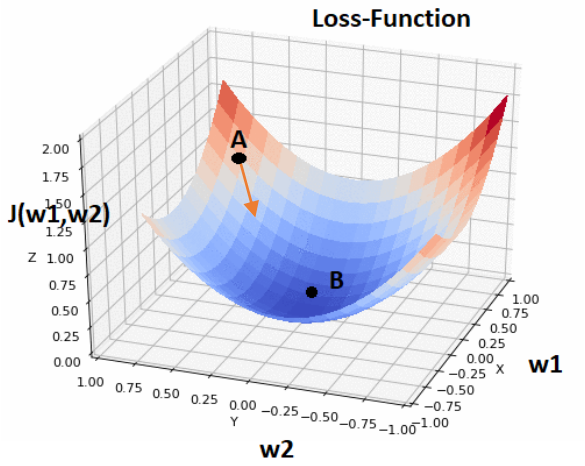

Figure 2.7 The Steepest Descent $(-\nabla J(w1, w2))$ from Point A to B

2.5.3 Avoiding Vanishing & Exploding Gradients, Dead Neurons, and Overfitting

Although the backpropagation algorithm shows promising efficiency in updating connection parameters, the loss function gradients usually get smaller and smaller as the gradients propagate towards the lower layers. As a result, the connection weights especially of the lower layers will take forever to train and converge to a desirably decent solution. The scenario is known as the vanishing gradients problem.

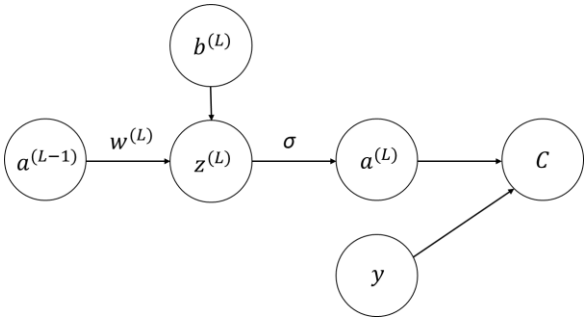

Figure 2.8 Neural Illustration of Output Node and its Preceding Node

For simplicity purposes, assuming the relationship between an output node of layer L and one of its preceding neurons, the output neuron activation, $a^{(L)}$ is determined by equations (2.11) and (2.12), where the transformation of the result, $z^{(L)}$ of weight, $w^{(L)}$, bias, $b^{(L)}$ of layer L and the previous neuron's activation, $a^{(L-1)}$ by the activation function, σ along with the desired output, y is used to compute the sum of squares loss, C as shown in Figure 2.8.

$$z^{(L)} = w^{(L)}a^{(L-1)} + b^{(L)} \tag{2.11}$$

$$a^{(L)} = \sigma(z^{(L)}) \tag{2.12}$$

The error gradient or the change of loss both with respect to weight and bias used for connection parameters update is computed by using the chain rule as shown in equations (2.13) and (2.14).

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial C}{\partial a^{(L)}} \tag{2.13}$$

$$\frac{\partial C}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial C}{\partial a^{(L)}} \tag{2.14}$$

From equations (2.4) and (2.6), the derivative ($\partial a^{(L)}/\partial z^{(L)}$) of sigmoid and tanh function lie in the intervals of (0,1) and (-1,1) respectively. Since the maximum modulus value of the derivatives is always less than 1, repetitive multiplication of smaller numbers will result in a very small error gradient especially at the region of lower layers as the chain rule progresses. In some cases, as opposed to the case of the vanishing gradient problem, the gradient may grow bigger and bigger as the algorithm propagates backward. This is the exploding gradients problem. By further expanding the chain rule applied,

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial a^{(L-1)}}{\partial w^{(L)}}\frac{\partial z^{(L)}}{\partial a^{(L-1)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial C}{\partial a^{(L)}} \tag{2.15}$$

$$\frac{\partial z^{(L)}}{\partial a^{(L-1)}} = \frac{\partial}{\partial a^{(L-1)}}(w^{(L)}a^{(L-1)} + b^{(L)}) = w^{(L)} \tag{2.16}$$

From equation (2.16), the term $\partial z^{(L)}/\partial a^{(L-1)}$ depends solely on the value of the $w^{(L)}$ and if the weight is inappropriately initialized as a big number, the result of the chain rule in equation (2.15) will be a very huge number, thus causing the error gradients to be divergently updated from the desired set of values. In the case of "dying ReLU", the dead neuron problem is caused by the zero gradients on the left side of the ReLU activation function derivative. The zero value of ($\partial a^{(L)}/\partial z^{(L)}$) will result in complete deactivation of certain connection areas where $z^{(L)} < 0$, which in turn brings the network into a state of coma. These problems show the significance of appropriate weight initialization and activation function selection before training an artificial neural network. With tens of thousands or even millions of learnable parameters, deep neural networks can learn highly complicated input-output relationships and its flexibility allows an incredible amount of freedom for parameter tuning. However with the limited size and quality of the dataset, the great flexibility may lead to overfitting where a majority of the so-called complex relationships are the results of sampling noise that exist only in the training set but not in real test set drawn from the actual distribution (Srivastava et al., 2014). Failure of hitting a desired generalizability is a common problem faced by deep neural networks. To reduce the degree of overfitting in the network, multiple regularization techniques can be implemented, including early stopping, the introduction of weights penalty, batch normalization, and dropout as proposed in (Srivastava et al., 2014). The opposite of overfitting is underfitting, which can be solved by simply increasing the number of training epochs in hope of getting a better fit to the data.

2.5.3.1 Weight Initialization

A proper weight initialization is very essential to ensure the convergence of a neural network. Glorot and Bengio (2010) experimented and proposed the very first initialization analysis and it was found that by assuming equal layer inputs (fan_{in}) and outputs (fan_{out}) neurons in the case where the activation function is differentiable at 0,

the variance of input and output gradients are equal both in forward and backward flow of the algorithm. The widely used initialization method has proven to work very well in practice and is known as the Glorot Initialization. In the initialization method, the weights in a neural network are sampled and initialized from a normal distribution N(0, σ^2) where the variance,

$$\sigma^2 = \frac{1}{fan_{avg}} \tag{2.17}$$

$$fan_{avg} = \frac{fan_{in} + fan_{out}}{2} \tag{2.18}$$

With fan_{avg} in equation (2.17) replaced by fan_{in} comes the initialization strategy named LeCun initialization. LeCun initialization is identical to that of Glorot when $fan_{in} = fan_{out}$. In the case where the neural network is built entirely with dense layers and SELU activation function, LeCun initialization works extremely well with the self-normalizing nature of the neural network. However, despite Glorot's promising weight initialization method, it was found that the technique did not perform well with activation functions that are not differentiable at 0 like ReLU and its variants. In association with that, He et al. (2015) proposed an initialization of weights sampled from variance,

$$\sigma^2 = \frac{2}{fan_{in}} \tag{2.19}$$

and justified their method with an experiment on a 30 layers deep ReLU function neural network in which He initialization method surpassed Glorot's method in terms of convergence. The activation functions and their corresponding weight initialization methods are summarized in Table 2.1 (Appendix B).

2.5.3.2 Variants of ReLU

As mentioned earlier in Section 2.4.1.3, the ReLU function has significant limitations and is subject to the problem of dead neurons which hinders model learning. To resolve the issue, multiple variants of ReLU such as Leaky ReLU (LReLU), Parametric ReLU (PReLU), Exponential Linear Unit (ELU), and Scaled Exponential Linear Unit (SELU) were proposed and applied in replacement of ReLU. The functions are defined in equations (2.20), (2.21), and (2.22) as

$$LRELU_{\alpha}(x) = PReLU_{\alpha}(x) = max(\alpha x, x) \tag{2.20}$$

$$ELU_{\alpha}(x) = max(\alpha(e^{x} - 1), x) \tag{2.21}$$

$$SELU_{\alpha,\lambda}(x) = \lambda max(\alpha(e^{x} - 1), x) \tag{2.22}$$

The hyperparameter α is tuneable and it ranges from 0 to 1. Both LReLU and PReLU have an identical function but different nature of α. In LReLU, hyperparameter α is user-defined while in PReLU, α acts as a learnable parameter which will gradually be modified by backpropagation (Nwankpa et al., 2018). For sequential network architecture, SELU would be the best candidate among the variants. If the architecture is non-sequential and restricts the network from self-normalizing, ELU (or SELU with $\lambda = 1$) will generally outperform SELU. Depending on the size of the network architecture, LReLU and PReLU may solve the concerns about runtime latency due to their non-exponential nature for faster training.

2.5.3.3 Batch Normalization

Batch Normalization (BN) is an important technique to address the vanishing and exploding gradients problems by reducing the internal covariate shift of the neural network. Internal covariate shift is defined as the activation distribution shift due to the constant changing of network parameters during training (Ioffe and Szegedy, 2015).

In the BN technique, an operation of zero-centering and normalizing activation is added and performed before or after a hidden layer. By using two new parameter vectors, BN scales and shifts the input result so as to let the model learn optimally and not chasing the moving dot. These actions create the whitening effect of inputs produced by the lower layer, thus making a big step towards achieving non-moving input distribution to mitigate the negative impact of internal covariate shift (Ioffe and Szegedy, 2015).

2.5.3.4 Dropout

The serious overfitting problem of deep neural nets with an extremely large number of parameters can be overcome by the inclusion of dropout layers in model top layers. As detailed in (Srivastava et al., 2014), there were cases where state-of-the-art neural networks achieving as much as 2% of accuracy boost with just the simple aid of the dropout regularization technique. There is a structure-wise resemblance of dropout layer to a dense layer, the only thing different is that at every training step, every neuron of the dropout layer has a probability or dropout rate, p of being "shut down", leaving the weights associated to the neuron untrained and ineffective to the preceding and proceeding layer for the particular step. Applying the dropout layer to a neural network can be thought of as sampling vast amounts of sub-networks that contain survived dropout neurons. Consider just adding a dropout layer with n units of neuron, there are $2^n - 1$ possible sub-networks to be sampled and presented for weights training (Srivastava et al., 2014). Thus, the dropout technique performs model combination and grants neural networks a very powerful generalizability in counter of overfitting problem.

2.6 Convolutional Neural Network (CNN)

As opposed to its 2D counterparts, the deep 1DCNN is exclusively designed to operate on 1D signals. In 1DCNN, instead of accepting a 2D data array like what a conventional CNN does, the passive input layer receives a 1D data array. Alongside

the input layer is the most important building block of a CNN, which is the convolutional block that usually comprises a convolutional layer, a pooling layer, and the choice of activation functions and batch normalization (Avci et al., 2021). The neurons in a convolutional layer are not dense, they are connected to the 'pixels' of their respective receptive fields. Each convolutional layer is associated with one filter which decides the dimension of the receptive field. Depending on user-defined filter size and the number of feature maps to be extracted from the input by sliding the filter across the data array, the stacked feature maps decide the dimension of a convolutional layer. Next up, the feature maps are downsampled by a pooling layer to summarize the activation or the presence of a feature. The block process is repeated before reaching the top layer where the final layer of the final block is obtained and flattened to an array of neural nodes. The flattened layer is followed by a sequence of dense layers as discussed in Chapter 2. The whole process resulted from the fusion of both feature extraction and regression operation can be optimized by both forward and backpropagation to maximize prediction performance.

2.6.1 CNN in Fault Detection and Diagnosis (FDD)

CNN has recently been applied in the development of state-of-the-art methods and proven to be efficient in vibration-based problems by recent studies. In a recent study. In 2018, Abdeljaber et al. designed and trained a 1-dimensional CNN (1DCNN) based model by using Silicon Drift Detector (SSD) vibration data to detect and predict the structural damage of a civil structure. The vibration response of 12 sensors used in a total of 31 scenarios was recorded and used to train a 1DCNN model for each sensor which is only responsible for predicting the local average damage value PoD_{avg}at the corresponding location. It was found that the result of the approach was a success and had unprecedentedly achieved successful predictions on completely unseen damage scenarios (Abdeljaber et al., 2018). In another study, Yu et al. (2019) conducted an experiment on damage identification of smart buildings. Instead of using 1DCNN, the waveform acceleration data from 14 accelerometers were concatenated into a 2-dimensional matrix for 2DCNN model training. Among three models used in the regression problem, the proposed 2DCNN based model showed the least root mean

square error (RMSE) percentage of predicted damage severity in both validating (0.98%) and testing (1.6%) phase as compared to the other models namely General Regression Neural Network (GRNN) and Adaptive Neuro-Fuzzy Inference System (ANFIS). In studies conducted by Abdeljaber et al. (2018) and Yu et al. (2019), both 1-dimensional and 2-dimensional CNN have shown superior results and performance over other architectures in the field of vibration-based FDD. Besides, the studies conducted by Chae et al. in 2020 showed a detection accuracy of 99.1% with their proposed CNN model in NPP coolant system pipe damage detection due to fast accelerated corrosion (FAC) (Chae et al., 2020). The wide usage of CNN as FDD methods in the field of nuclear engineering can also be exemplified by Chen et al. (2017) in their study on crack detection of underwater metallic surfaces for NPP components inspection. While 1DCNN has a relatively low computational cost for training and is preferable for isolated applications (Avci et al., 2021), 2DCNN outperforms 1DCNN in cases involving a massive number of detectors through data concatenation.

2.7 Proposed Work

In this study, a variant of 1DCNN called the Patch-Based 1DCNN is proposed. While being able to retain the advantage of low computational cost as a 1DCNN, the patch networks design in the main structure allows the model to focus on the importance of deep regional or local features and hold them together for prediction. The proposed model is expected to be as precise as 2DCNN with data concatenation in vibration analysis and outperform conventional CNNs in terms of learnable parameters' tuneability and flexibility under an appropriate regularization framework. In association with that, a few alternatives of different modelling apporach which includes machine learning algorithms and also CNN of different architecture are also proposed as additional options for model selection. The techniques used, architecture, training, validating, and the evaluation processes of the Patch-Based 1DCNN and other models will be further discussed in the coming chapters.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

This chapter focuses on the handling of experimental dataset and the outline of research methodology to achieve the study objectives. In this section, the architecture of the proposed Patch-Based 1D Convolutional Neural Network model is discussed and designed based on multiple factors or pre-settings mentioned in Chapter 2. Throughout the training and testing methodology, TensorFlow-Keras along with open-source Python packages such as Pandas, Numpy, Matplotlib, and Scikit-Learn will be utilized.

3.2 Experimental Flowchart

Figure 3.1 (Appendix C) shows the experimental flowchart to achieve the objectives of the study. First, data cleaning is performed to eliminate any corrupted or incomplete data within the dataset obtained. The cleaned data are then processed into suitable formats and split into train and test subsets. Based on the theoretical considerations discussed in Chapter 2, the main architecture of the proposed model is designed and its patch networks are initialized appropriately in their respective signal region for pre-training. The bottom layers of the pre-trained submodels are used in constructing the main model for the next training phase, global fine-tuning. The main model is evaluated using the unexplored test dataset and depending on its result, the multi-model approach might be needed to further increase the general flexibility of the fitting process, with each parameter being predicted by its specifically tuned predictive model. In the following section, each process in the flow diagram will be discussed in more detail.

3.3 Dataset

In this study, the dataset used consists of simulation time-domain vibration data of 2800 collision events generated by 4 accelerometers. Each collision event is associated with collision parameters such as collision point, (X, Y) in mm, the mass, M (g), and velocity, V (m/s) of the body upon collision to be trained and predicted. The locations of the accelerometers in the simulator are enclosed to avoid any intended selection bias. Figure 3.2 (Appendix C) shows an example of a simulated collision event used in generating the dataset.

3.3.1 Data Preprocessing

Since the acceleration data of all 2800 impact events have the same uniform distribution and number of time instances which is 375, the specific time instances are of no importance and can be represented as a simple unit. For each impact event, the 4 one-dimensional arrays of acceleration wave data from 4 accelerometers are concatenated to form a 4 x 375 two-dimensional array representing a single training data as illustrated in Figure 3.3 (a). Alternatively, instead of using the raw waveform of the data, fast Fourier transform (FFT) is performed to generate another set of experimental data by computing discrete Fourier transform (DFT) of the data sequences as shown in Figure 3.3 (b).

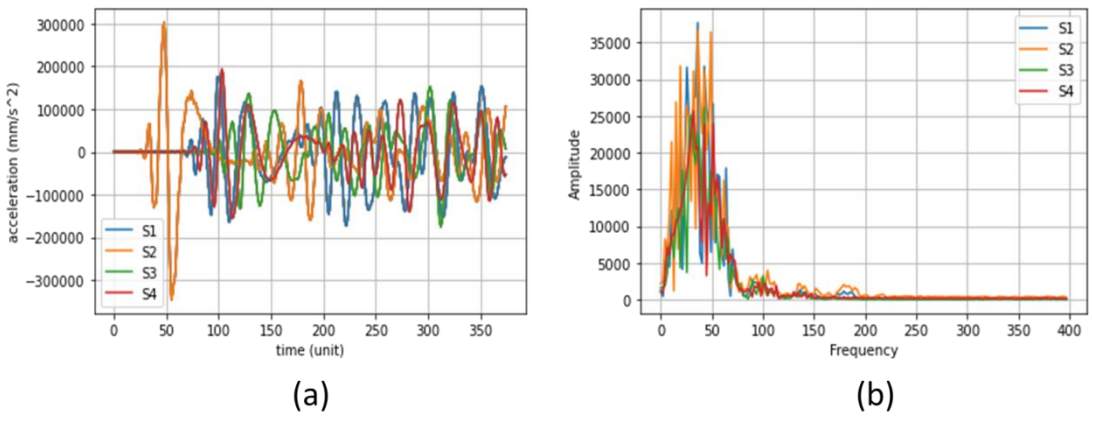

Figure 3.3 (a) Raw Waveform and (b) DFT plot of Waveform

3.3.2 Train-Test Split

By using the Scikit-Learn library, specifically the train_test_split method, the data arrays are split into random train and test subsets. In this experiment, the test_size is set to 0.2, giving a train-test ratio of 8:2, equivalent to 2240 training and 560 testing instances respectively.

3.4 Patch-Based 1D Convolutional Neural Network

The Patch-Based 1DCNN is composed of two learning processes, namely local pre-training, and global fine-tuning. Since there are four sets of accelerometer vibration signals for each data instance, the 1DCNN architecture is split into four smaller CNNs, called PatchNets for simplicity. During the local pre-training phase, each PatchNet is initialized with a similar top layer as the main CNN and trained in their respective signal region. In the global fine-tuning phase, the tuned weights and biases are used to initialize the four CNN patches of the whole model and their final convolutional block outputs are flattened and concatenated as the input to a shared main top layer for training. Since the deep convolutional layers are locally specialized, the model is expected to be able to separately extract useful information from each set of vibration data and effectively perform generic summarization.

3.4.1 Model Architecture

The fundamental Patch-Based 1DCNN presents four convolutional blocks at the bottom, with each block comprising a one-dimensional convolutional layer immediately followed by a batch normalization, a SELU activation layer, and a max pooling layer as shown in Figure 3.4 (a) (Appendix D). The batch normalization layer serves as a normalizer to the output feature maps from the convolutional layer. Since the network is to be trained sequentially, the SELU activation function is preferred for better training performance. At the top of the network, output feature maps of the last convolutional block are flattened and connected to a dense layer of 375 nodes which

is also followed by a batch normalization, a SELU activation, as well as a dropout layer with a dropout rate of 0.5. The final layer of the model consists of four nodes that return four continuous numerical values, representing the collision point, (X, Y) in mm, the mass, M (g), and velocity, V (m/s).

3.4.2 Local Pre-training

In order to effectively initialize the whole Patch-Based 1DCNN, the model is made specialized in deep features from each signal region by splitting the two-dimensional vibration data and Patch-Based 1DCNN architecture into four patches and PatchNets respectively. The architecture of a PatchNet is shown in Figure 3.4 (b) (Appendix D). Each PatchNet expects a patch or one-dimensional array of 375 elements from a respective vibration signal of the accelerometer. Before proceeding to the pre-training phase, hyperparameters such as filter number of one-dimensional convolutional layer, momenta of batch normalization layer, and the learning rate of PatchNet are fine-tuned. While maintaining a 2:3:4:5 ratio for the filter number from the first to the last convolutional layer, an integer scaler ranges from 5 to 10 is introduced. The scaler introduced is applied by direct multiplication with the ratio, resulting in the number of filters as shown in Table 3.1 (Appendix D). Meanwhile, the batch normalization momenta and model learning rate are assigned with two sets of values respectively, [0.9, 0.99, 0.999] and [0.1, 0.01, 0.001]. By using the training set and a suitable tuning method, the three sets of hyperparameter values are implemented into a defined HyperModel and run for 20 epochs to select the best performing hyperparameter combination to achieve the objective function, in this case, the combination with the lowest validation error. In this experiment, the Bayesian Optimization method is preferred as it is a probability model that keeps track and learns from its past evaluations in contrast to other methods like random and grid search. In correspondence to the SELU activation used, the LeCun Normal method together with the best hyperparameter combination is then used to initialize each PatchNet for pre-training.

3.4.3 Global Fine Tuning

After training the four PatchNets in their respective signal region, their final weights and biases, excluding that of top layers are used to construct the bottom layers of the whole Patch-Based 1DCNN with shared top layers as shown in Figure 3.5 for global training on full vibration data training. The top layers of the main model are randomly initialized so to further improve the generalizability of the model. After the initialization, the same training data are split into patches and fed into their respective channels to fine-tune the weights of the model as a whole, allowing it to explore more global features which were not locally learned by each PatchNet.

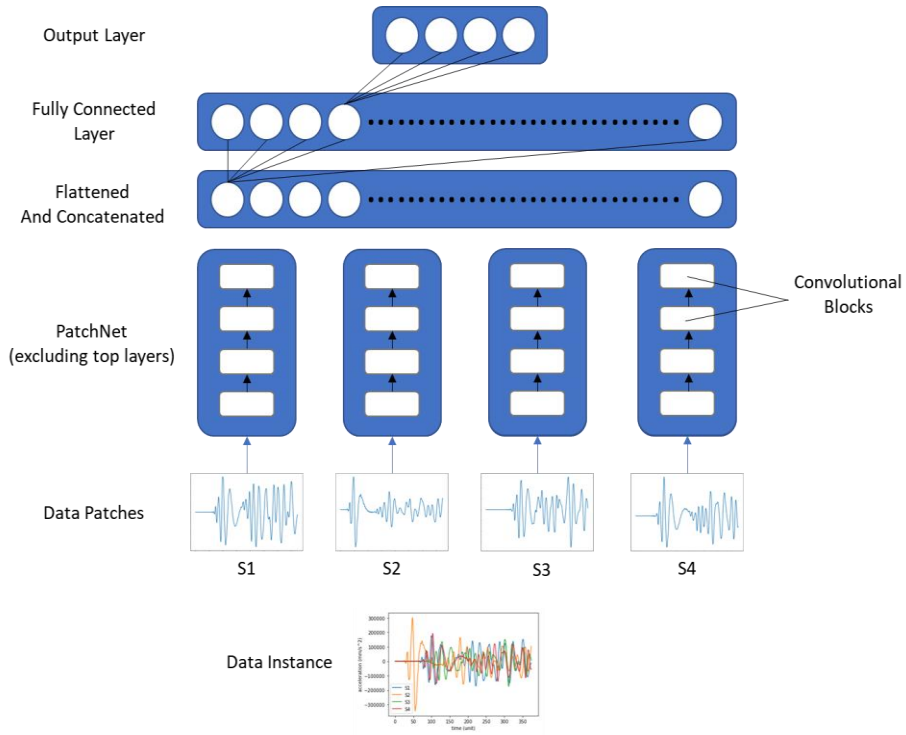

Figure 3.5 Patch-Based 1DCNN Architecture

3.5 Testing

Since the testing dataset has yet to be explored and remains unseen to the model throughout the training process, it is used as the gold standard to provide an unbiased evaluation of the final model. Once the model is completely trained, the tuned weights

and biases are frozen and remain uninfluenced during the testing phase. In this experiment, the model is evaluated based on the Mean Absolute Percentage Error (MAPE) of the 560 instances to be tested.

3.6 Multi-Model Approach

Depending on the result of the proposed approach, a multi-model approach may necessarily be implemented where each target parameter is associated with only one predictive model. In this approach, both machine learning algorithms and deep learning models will be considered to look for the best model for each parameter. In contrast to the nature of the proposed model, there will be 3 separate models used to individually predict the impact point (X, Y), the velocity (V), and the mass (M) of the collision body. Depending on the collision parameters' time-frequency dependencies, a suitable type of dataset generated is used in their respective training, validating, and testing processes.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

In this chapter, the proposed model Patch-Based 1D Convolutional Neural Network will be contructed and evaluated as planned. In the multi-model approach, the model setting process will be carried out with different appropriate combinations of model frameworks, architectures, weight initialization methods, regularization methods, and activation functions based on their suitabilities. The statistical performance of each model will be further discussed in the following section.

4.2 Patch-Based 1D Convolutional Neural Network

By using Python 3.9, the PatchNet architecture are constructed and defined using TensorFlow-Keras in Jupyter Notebook, an interactive web-based computing platform that is suitable for model training. A total of 4 PatchNet are defined, each representing an input channel for wave data of each sensor. As described in Chapter 3, hyperparameter tuning will be performed, followed by local pre-training of PatchNet, global fine tuning, and model testing on bth raw waveform and FFT waveform data.

During hyperparameter tuning, Bayesian Optimization algorithms is used and the best working combination of parameters is found to be [‘scaler’ : 9, ‘momentum’ : 0.99, ‘learning_rate’ : 0.001]. The technique of early stopping is applied and its parameters are set to halt the model training once 50 non-progressive epochs based on validation mean absolute error are reached, returning the best performing model weights throughout the training history. By using the best parameters from the tuner, each PatchNet is initialized and trained for 500 epochs. The training history of all PatchNet is shown in Figure 4.1.

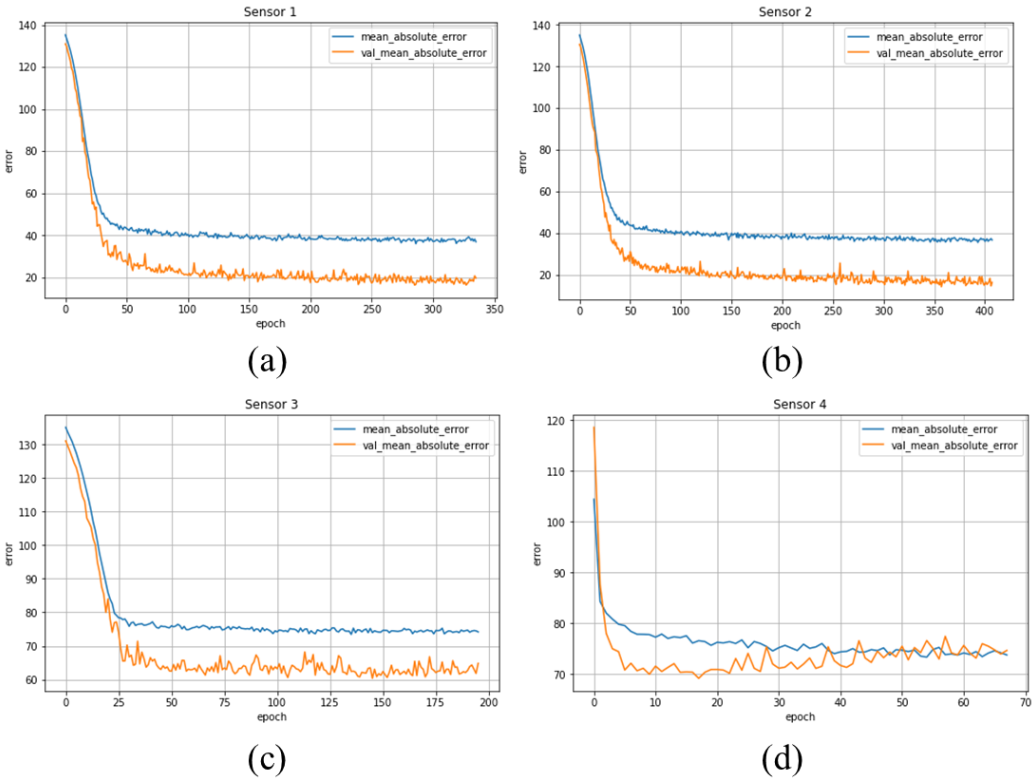

Figure 4.1 Training History of Model for (a) Sensor 1, (b) Sensor 2, (c) Sensor 3, and (d) Sensor 4

From the figure above, it can be noticed that both training and validation mean absolute error dropped significantly in the first 20 epochs. According to Figure 4.1 (a) and (b), Model 1 and 2 managed to make their long ways to more than 350 epochs, indicating a good training with fewer encounter of non-progressive epochs. On the other hand, Model 3 and 4 shown in Figure 4.1 (c) and (d) were called back and halted earlier due to their fluctuating validation mean absolute error, inferring the lack of useful information from the wave data of Sensor 3 and 4 for generalization purpose and all-at-once prediction of collision parameters. In spite of the considerably promising performance by Model 1 and 2, almost all models have their training mean absolute error higher than that of validation. In most cases, training error would always underestimate validation error. While the essence of deep learning is to learn from the training and predict the unknown, almost all PatchNets in this experiment performed better at predicting the unknown than what they have learned. This is a bad sign of unknown fits as the models are not predicting based on what they are trained for.

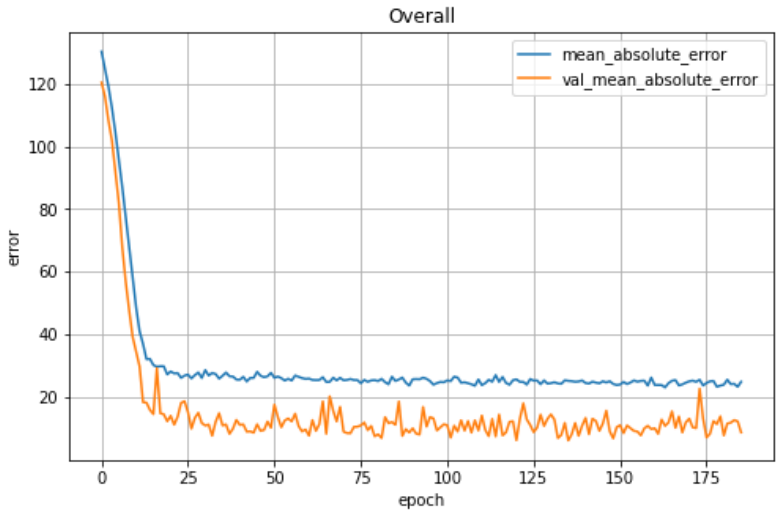

Figure 4.2 The Overall Model Training History

After the pre-training process, the 4 PatchNets are concatenated and the resulting model architecture plot as shown in Figure 4.3 (Appendix E) is then subject to global tuning. Having the exact same characteristic of unknown fit as illustrated in Figure 4.2, the final model is halted at epoch 186, recording the best overall validation mean absolute error of 6.2713 (0.6596%). As expected, the final model did not perform well on testing sets, marking a mean absolute error of 110.5015 (11.6219%) which is also a clear sign of overfitting. Before proceeding to the multi-model approach, another attempt was made by switching to the dataset after FFT. However, the dataset was found to be very poor and lack of meaningful information for collision parameters prediction both in single and multi-model cases. Thus, the FFT dataset is eliminated from the experimental procedures afterwards.

4.3 Multi-model Approach

In this approach, a sets of regression machine learning algorithms are assembled and validated using a pipeline structure to select the best performing models for each collision parameter for tuning and testing. In prior to that, the raw wave data of each sensors are arranged and processed into features to feed the machines. Additionally, a simplied version of 1D convolutional neural network is also

conctructed and tested separately on each collision parameter with the aid of scalers and appropriate final output activation functions based on the range of target values.

4.3.1 Machine Learning (ML) Algorithms

In this method, machine learning algorithms are used for modelling based on the sample data with a different feature engineering approach. Instead of using the raw waveform as the input data, the amplitude maximum, minimum, mean, median, standard deviation and the distribution skewness of each sensor data array for data instances are computed and made into a data frame as shown in Figure 4.4. The resulting table has a dimension of 2800 x 24, which consists of 2800 instances of 1D arrays to be fed to the machines.

id	S1_max	S2_max	S3_max	S4_max	S1_min	S2_min	S3_min	S4_min	S1_mean	S2_mean	...
0	235080.9	235080.90	231773.4	286557.2	-315471.5	-315471.50	-168789.1	-293865.7	-308.304553	-308.304553	...
1	1069688.0	1244737.00	951349.6	1095499.0	-826482.9	-2390317.00	-1137225.0	-1306526.0	12780.176893	981.778780	...
2	383092.2	150770.40	148724.0	168649.2	-364591.8	-161006.00	-150458.2	-201313.7	-674.289628	-1575.681938	...
3	379650.3	378475.30	394448.7	487880.4	-329680.7	-726961.00	-333712.8	-463918.9	2164.216981	-11274.728589	...
4	754833.0	321731.20	373526.8	397334.0	-592148.9	-378091.60	-423593.5	-472361.4	-980.480585	-6966.912566	...
...	...	...	...	...	...	...	...	...	...	...	...
2795	698079.1	1027622.00	1031316.0	1037711.0	-793812.4	-720545.40	-1155664.0	-1703724.0	5600.614397	5885.824925	...
2796	290787.9	144759.00	119504.7	157180.4	-493104.6	-179902.70	-100318.6	-180438.7	-1215.328099	2702.879683	...
2797	109273.7	98441.96	109713.6	162800.1	-127440.0	-90686.41	-101889.3	-123130.7	436.239993	276.327514	...
2798	576629.1	550360.40	803728.8	878377.7	-587942.2	-742554.80	-811196.2	-1196096.0	-2394.515047	6230.349355	...
2799	648335.8	965234.70	499031.7	645815.6	-571597.8	-627908.20	-492273.2	-621268.9	1059.619352	28.166706	...

2800 rows x 24 columns

Figure 4.4 Transformed Sensors Wave Data

From the Scikit-learn machine learning library, regression models like Linear Regression (LR), Elastic Net (EN) (representing Lasso and Ridge regression), K-Nearest Neighbours Regressor (KNN), Decision Tree Regressor (CART), and Gradient Boosting Regressor (GBM) are imported and appended into a pipeline for k-fold cross validation to predict the collision parameters using 10 splits within the training set. During the validation process, the dataset will be split into 10 sets (9 for training and 1 for validating) and validated for 10 trials, with each split acting as the

validation set per trials. The resulting mean absolute errors of each model after the 10 trials are averaged with their respective trial standard deviation computed and tabulated as shown in Table 4.1.

Table 4.1 Performance of ML Models on Each Collsion Parameters

Model	X, Y [-1,1]		M [0,1]		V [0,1]	
	Mean	Std	Mean	Std	Mean	Std
LR	0.328200	0.013658	0.207814	0.006412	0.109594	0.004585
EN	0.556603	0.008279	0.290931	0.009985	0.301392	0.009794
KNN	0.176049	0.011449	0.181503	0.006838	0.080335	0.005614
CART	0.128181	0.011692	0.153274	0.021667	0.070647	0.009285
GBM	nan	nan	0.166211	0.005929	0.075040	0.003554

In cases of M and V prediction, CART achieved the lowest mean absolute error and highest standard deviation among all the models. Due to its relatively unsteady results throughout the trials, the second performing GBM will also be brought to the next tuning phase because of its steady performance. In X and Y prediction, the nan values indicate the incompatibility of 2D array outputs with the nature of GBM which combines “weak learners” sequentially to improve from its error. Thus, the cross-validations of X and Y are excluded from the experimental procedure due to the concerns of information impairments if parameter X and Y carrying the same locational information are to be treated separately. (e.g. an earlier discovery during the exploratory data analysis phase detected a few significantly distinct wave data with same value of X but totally different value of Y). Thus, only CART will be tuned and tested in the next phase.

Figure 1 and 2 illustrates the codes used to perform grid search and build a model for every possible combination of hyperparameter for evaluation. After the tuning process, the best performing combination will be used to initialize the model to be tested on the testing sets. For parameter X and Y, CART records a mean absolute error in percentage of 25.558% while for parameter M and V, giving {“CART” : 24.9399%, “GBM” : 15.6316%} and {“CART” : 11.0310%, “GBM” : 6.8796%}

respectively. The overall performance of using only CART model is 20.5096%, while for CART+GBM, giving a moderately lower mean absolute error of 15.8564%.

4.3.2 Simplified 1D Convolutional Neural Network

Instead of separating the wave data into patches based on their sensor origin, the data are transposed and arranged in such a way to directly feed the model with 4 channels of 1D arrays. A very simple and direct 1DCNN architecture as illustrated in Figure 4.5 (Appendix E) is used for training. In order to ease the training process, XY and M target values, originally ranged at [-400,400] and [25,175] are scaled to [-1,1] and [0,1] using a min-max scaler, governed by tanh and sigmoid respectively as the final output activation function of their respective model. A sigmoid final output activation function is also directly applied to Model (V) without the need of scaling as its target values at range [0.2,1] are already within the sigmoid "sweet spot". By applying the same number of training epochs and callback, each model is trained individually.

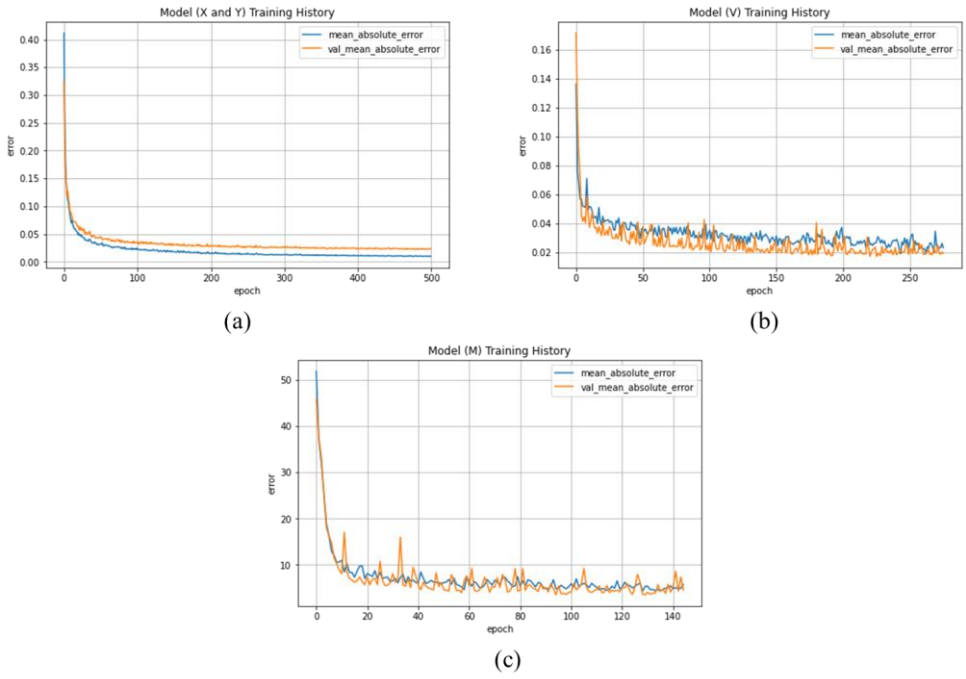

Figure 4.6 Training History of Model (a) X and Y, (b) V, and (c) M

From Figure 4.6, it can be noticed that the neural network training history of parameter X and Y showed a good and healthy trendline with its training mean absolute value slightly lower than that of validation set. Besides, Model (X and Y) also made its way through 500 epochs without hitting 50 non-progressive epochs and with significantly less fluctuations as compared to the previous approach. Since, there is relatively high progressiveness and low fluctuation magnitude in the training curve of Model (X and Y), it is believed that the mean absolute error can further be converged and reduced if the training continues. Although there is still an insignificant sign of unknown fit, Model (V) and Model (M) has considerably small gap between the training and validation mean absolute error around lower-error region. As expected, all models perform well on test set and produce a very good overall result with Model (X and Y) at percentage mean absolute error 1.1147%, Model (M) at 1.9967%, and Model (V) at 1.4943%. The overall mean absolute error for this multi-model solution is 1.5352%.

4.4 Summary

The proposed Patch-Based 1DCNN is stated to have its limitation in dealing with unknown and overfitting and this may be the result of excessively deep layers in the model architecture. FFT waveform data is also found to be unfruitful in bringing out distinctive information to characterize each data instance in both cases of single and multi-model approach. The data in the frequency domain might have been subject to distortion by reflections of wave after the collision which is not appropriately treated. For multi-model approach, the lack of real-time incremental and interactive learning in ML algorithms is the main cause of inefficient hyperparameter tuning. In this experiment, the need of ML algorithms for narrow and specific training necessitates further feature engineering to transform the raw wave data from four sensors into an 1D array, which might not promise a good result. On the other hand, a simplified 1DCNN with a self-defined loss function and the most basic architecture of 3 1D convolution blocks and 2 hidden layers yields the best result with the suitable use of target data scaler and activation functions in the final output layer of each model. It can be summarized that a very deep and complex neural network architecture does not

necessarily pledge a promising performance especially in regression problems. The overall experimental results of each model from both approahces are tabulated in Table 4.2.

Table 4.2 Overall Performance of Experimental Models

			Mean Absolute Error				Overall
			X	Y	M	V	
Patch-Based 1DCNN			11.6219%				11.6219%
Multi-model Approach	ML Algorithms	CART	25.5580%		24.9399%	11.0310%	20.5096%
		CART+GBM			15.1316%	6.8796%	15.8564%
	Simplified 1DCNN		1.1147%		1.9967%	1.4943%	1.5353%

CHAPTER 5

CONCLUSION

In conclusion, the main objectives of this study, namely the data processing method development, model training, selection, tuning, evaluation and selection are achieved. Among the 3 experimented models, the best performing model on mean absolute error (percentage) metric is the simplified 1D convolutional (1.5353%), followed by the proposed Patch-Based 1D Convlutional Neural Network (11.6219%), CART+GBM (15.8564%) and CART (20.5096%). While being the only many-to-many (multi-output) model in this experiment, Patch-Based 1DCNN still performed better than all ML algorithms running on one-to-one (single-output) modelling, which has once again proven the advantages of deep learning in terms of tuneability, flexibility. With a switch to multi-model approach and the simplification of proposed model in counter of its overfitting problem, the model has shown massive improvements in its stability, generalizability and predictability which hits the main goal of the experiment that is to achieve a mean absolute error of less than 1.6% benchmarked to the previous similar FDD experiment done by Yu et al. (2019) on damage identification.

REFERENCES

Abdeljaber O, Avci O, Kiranyaz MS, Boashash B, Sodano H, Inman DJ. 1-D CNNs for structural damage detection: Verification on a structural health monitoring benchmark data. Neurocomputing. 2018;275:1308-17.

Avci O, Abdeljaber O, Kiranyaz S, Hussein M, Gabbouj M, Inman DJ. A review of vibration-based damage detection in civil structures: From traditional methods to Machine Learning and Deep Learning applications. Mechanical Systems and Signal Processing. 2021;147:107077.

Chae, Y. H., Kim, S. G., Kim, H., Kim, J. T., & Seong, P. H. (2020). A methodology for diagnosing FAC induced pipe thinning using accelerometers and deep learning models. Annals of Nuclear Energy, 143, 107501. doi:10.1016/j.anucene.2020.107501.

Chao L, Jiafei L, Liming Z, Aicheng G, Yipeng F, Jiangpeng Y, et al. Nuclear Power Plants with Artificial Intelligence in Industry 4.0 Era: Top-level Design and Current Applications—A Systemic Review. IEEE Access. 2020.

Chen C-C, Liu Z, Yang G, Wu C-C, Ye Q. An Improved Fault Diagnosis Using 1D-Convolutional Neural Network Model. Electronics. 2021;10(1):59.

Chen, F.-C., & Jahanshahi, M. R. (2018). NB-CNN: Deep Learning-Based Crack Detection Using Convolutional Neural Network and Naïve Bayes Data Fusion. IEEE Transactions on Industrial Electronics, 65(5), 4392–4400. doi:10.1109/tie.2017.2764844.

Datta L. A survey on activation functions and their relation with xavier and he normal initialization. arXiv preprint arXiv:200406632. 2020.

Glorot X, Bengio Y, editors. Understanding the difficulty of training deep feedforward neural networks. Proceedings of the thirteenth international conference on artificial intelligence and statistics; 2010: JMLR Workshop and Conference Proceedings.

He K, Zhang X, Ren S, Sun J, editors. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. Proceedings of the IEEE international conference on computer vision; 2015.

Ioffe S, Szegedy C, editors. Batch normalization: Accelerating deep network training by reducing internal covariate shift. International conference on machine learning; 2015: PMLR.

Kiranyaz S, Avci O, Abdeljaber O, Ince T, Gabbouj M, Inman DJ. 1D convolutional neural networks and applications: A survey. Mechanical Systems and Signal Processing. 2021;151:107398.

Korean Atomic Energy Research Institute (KAERI). (2020). Collision Detection using Vibration Data (Version 1). Retrieved from https://dacon.io/competitions/official/235614/data.

Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems. 2012;25:1097-105.

Li J, Cheng J-h, Shi J-y, Huang F. Brief introduction of back propagation (BP) neural network algorithm and its improvement. Advances in computer science and information engineering: Springer; 2012. p. 553-8.

Maind SB, Wankar P. Research paper on basic of artificial neural network. International Journal on Recent and Innovation Trends in Computing and Communication. 2014;2(1):96-100.

Nwankpa C, Ijomah W, Gachagan A, Marshall S. Activation functions: Comparison of trends in practice and research for deep learning. arXiv preprint arXiv:181103378. 2018.

Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res. 2014;15(1):1929–58.

Svozil D, Kvasnicka V, Pospichal J. Introduction to multi-layer feed-forward neural networks. Chemometrics and intelligent laboratory systems. 1997;39(1):43-62.

Yu Y, Wang C, Gu X, Li J. A novel deep learning-based method for damage identification of smart building structures. Structural Health Monitoring. 2019;18(1):143-63.

Yukiya A, editor International Status and Prospects for Nuclear Power 2017. IAEA Board of Governors General Conference, Vienna; 2017.

Appendix A Graphical Representation of Activation Functions

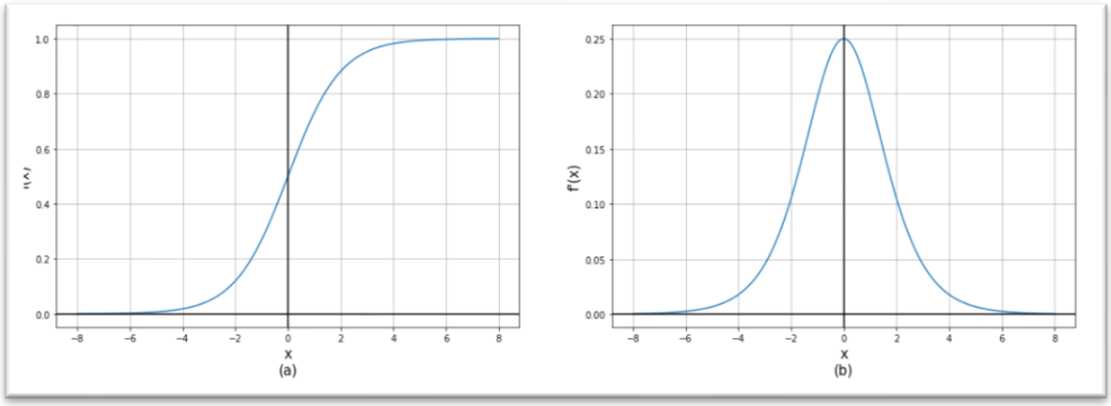

Figure 2.4 (a) Sigmoid Function and (b) Gradient of Sigmoid Function

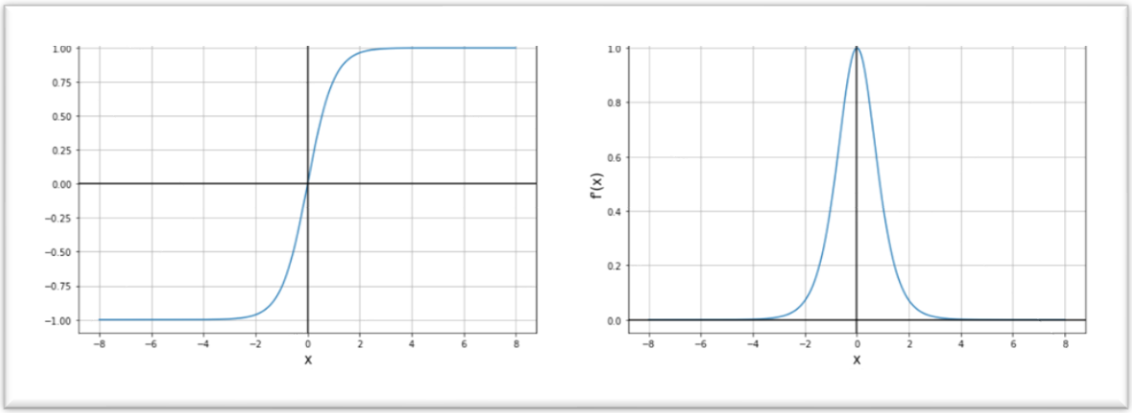

Figure 2.5 (a) Tanh Function and (b) Gradient of Tanh Function

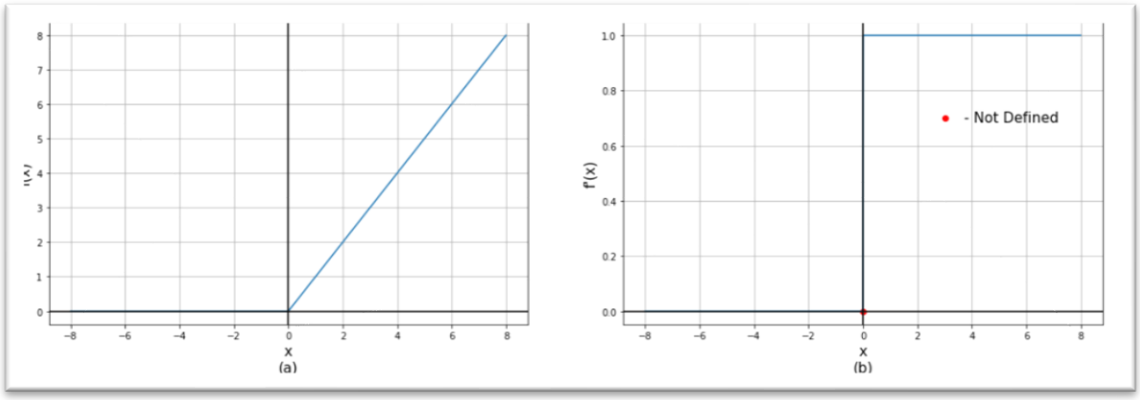

Figure 2.6 (a) ReLU Function and (b) Gradient of ReLU Function

Appendix B Weight Initialization Summary

Table 2.1: Initialization Methods for Different Activation Functions

Initialization Method	Activation Functions	σ^2 (Normal)
Glorot	Tanh, Logistic, Softmax	$\frac{1}{fan_{avg}}$
He	ReLU and variants	$\frac{2}{fan_{in}}$
LeCun	SELU	$\frac{1}{fan_{in}}$

Appendix C Experimental Flowchart and Simulation Illustration

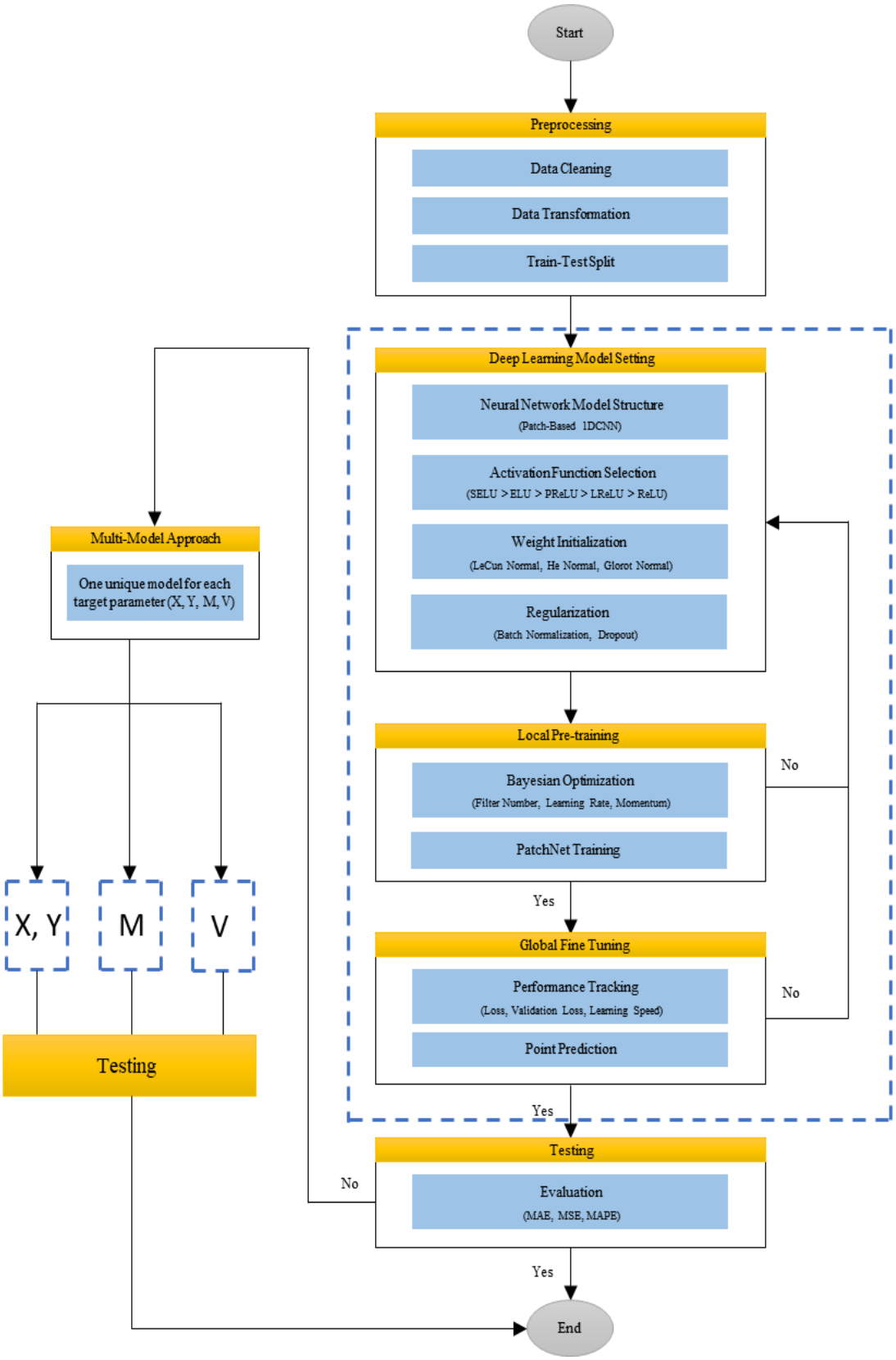

Figure 3.1 Experimental Flowchart

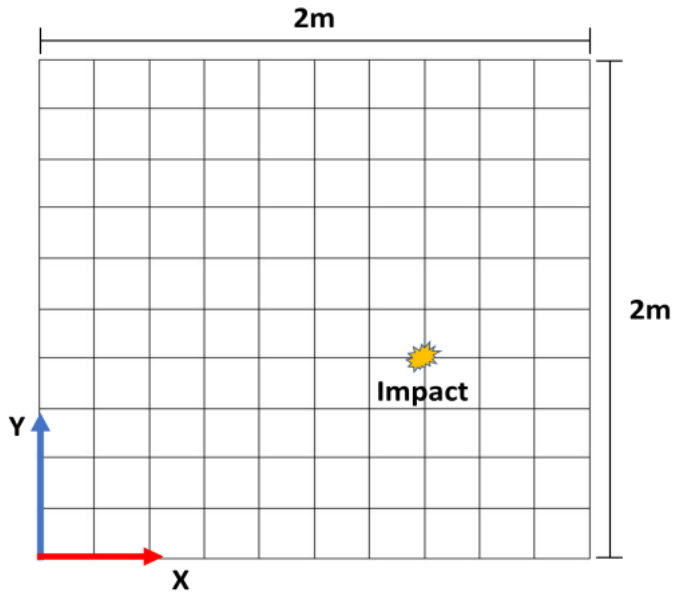

Figure 3.2 Illustration of Simulated Collision Event

Appendix D Model Layer Specifications and Basic Architecture

Table 3.1 Model Layer Specifications

Layer	Kernel Size	Stride	Input Maps	Output Maps
1D Convolutional 1	3 × 3	1	1 × (375)	Scaler*2 × (373)
Max Pooling 1	2 × 2	2	Scaler*2 × (373)	Scaler*2 × (187)
1D Convolutional 2	3 × 3	1	Scaler*2 × (187)	Scaler*3 × (185)
Max Pooling 2	2 × 2	2	Scaler*3 × (185)	Scaler*3 × (93)
1D Convolutional 3	3 × 3	1	Scaler*3 × (93)	Scaler*4 × (91)
Max Pooling 3	2 × 2	2	Scaler*4 × (91)	Scaler*4 × (46)
1D Convolutional 4	3 × 3	1	Scaler*4 × (46)	Scaler*5 × (44)
Max Pooling 4	2 × 2	2	Scaler*5 × (44)	Scaler*5 × (22)
Fully Connected	-	-	Scaler*5 × (22)	1 × (375)
Dropout	-	-	-	-
Output	-	-	1 × (375)	1 × (4)

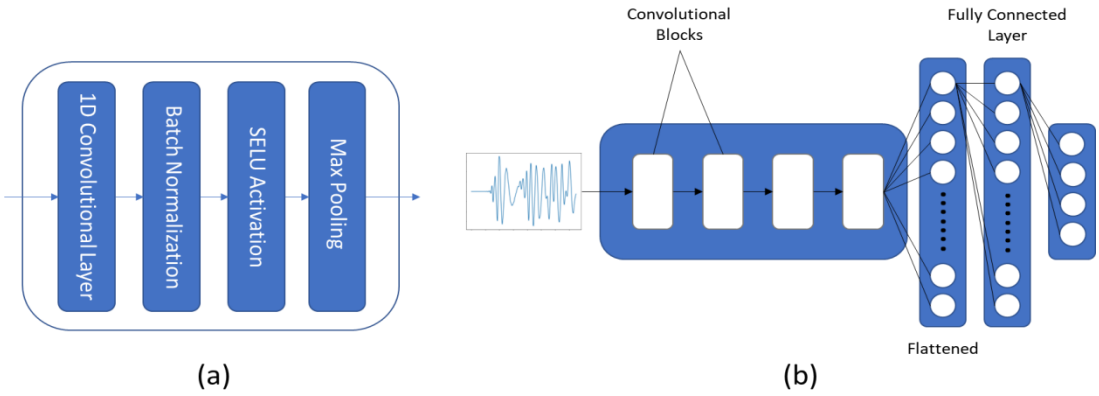

Figure 3.4 (a) Convolutional Block and (b) PatchNet Architecture

Appendix E Model Plotting and Source Code

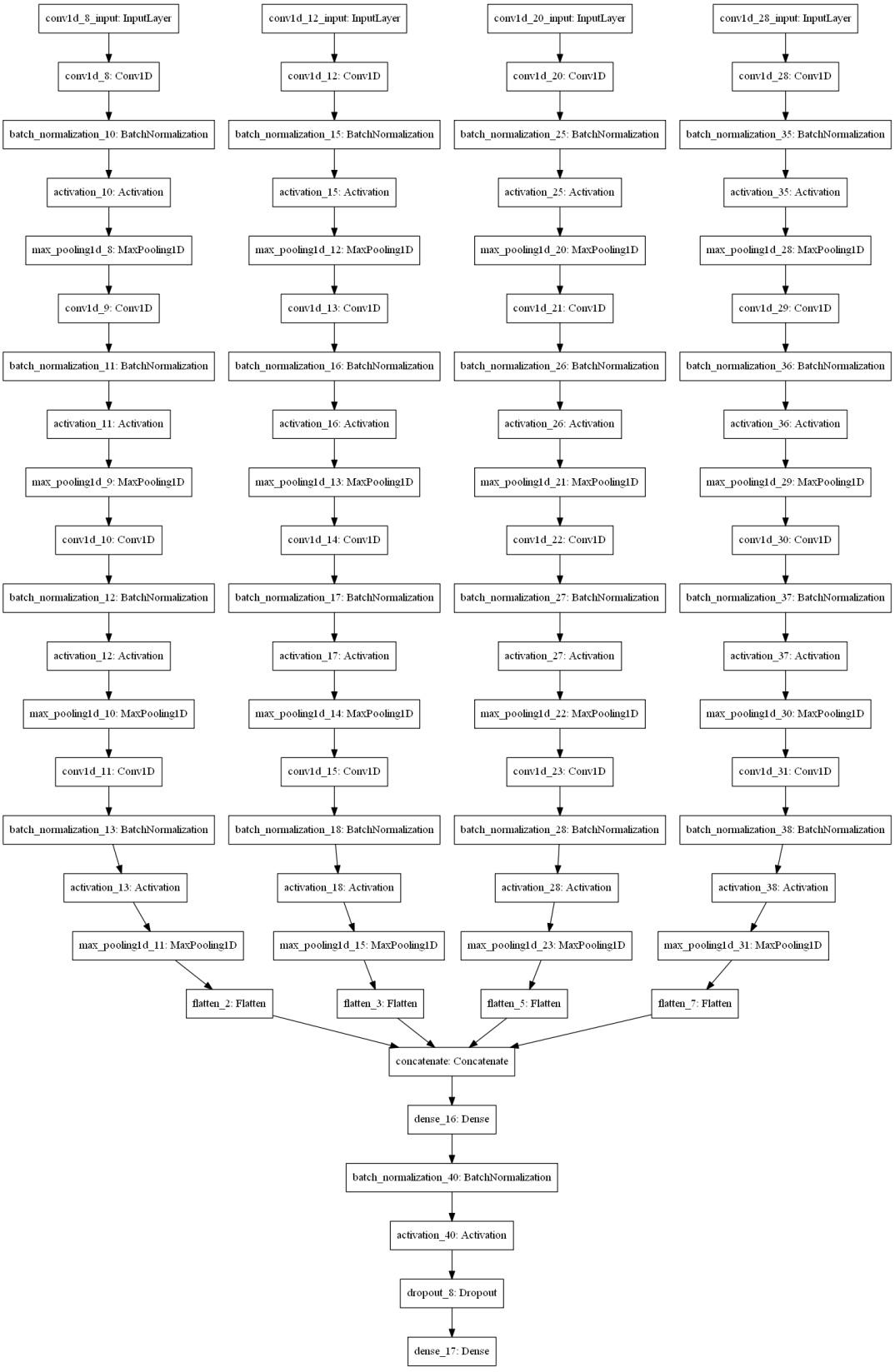

Figure 4.3 Model Plot of Patch-Based 1D Convolutional Neural Network

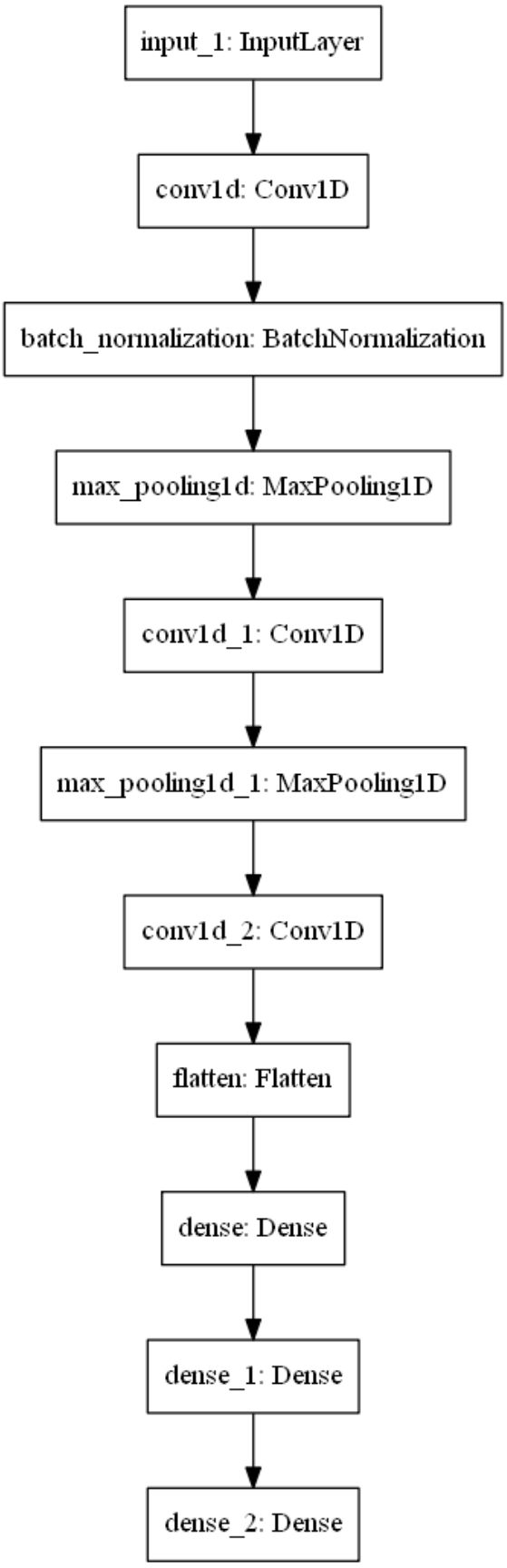

Figure 4.5 Model Plot of Simplified 1D Convolutional Neural Netwok

```
defaultConv1D = partial(tf.keras.layers.Conv1D,kernel_size=3,
                                            kernel_initializer=tf.keras.initializers.lecun_normal)
scaler = 9
momentum = 0.99
learning_rate = 0.001

model1 = tf.keras.Sequential()
model1.add(defaultConv1D(filters=2*scaler))
model1.add(tf.keras.layers.BatchNormalization(momentum=momentum))
model1.add(tf.keras.layers.Activation('selu'))
model1.add(tf.keras.layers.MaxPooling1D(pool_size=2))
model1.add(defaultConv1D(filters=3*scaler))
model1.add(tf.keras.layers.BatchNormalization(momentum=momentum))
model1.add(tf.keras.layers.Activation('selu'))
model1.add(tf.keras.layers.MaxPooling1D(pool_size=2))
model1.add(defaultConv1D(filters=4*scaler))
model1.add(tf.keras.layers.BatchNormalization(momentum=momentum))
model1.add(tf.keras.layers.Activation('selu'))
model1.add(tf.keras.layers.MaxPooling1D(pool_size=2))
model1.add(defaultConv1D(filters=5*scaler))
model1.add(tf.keras.layers.BatchNormalization(momentum=momentum))
model1.add(tf.keras.layers.Activation('selu'))
model1.add(tf.keras.layers.MaxPooling1D(pool_size=2))
model1.add(tf.keras.layers.Flatten())
model1.add(tf.keras.layers.Dense(units=50,kernel_initializer=tf.keras.initializers.lecun_normal))
model1.add(tf.keras.layers.BatchNormalization(momentum=momentum))
model1.add(tf.keras.layers.Activation('selu'))
model1.add(tf.keras.layers.Dropout(0.5))
model1.add(tf.keras.layers.Dense(units=4))

model1.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
              loss='mean_absolute_error',
              metrics=['mean_absolute_error'])
```

Figure 4.7 PatchNet Training Source Code

```
momentum = 0.99
learning_rate = 0.001
conc_list = list()
input_list = list()
for i in range(4):
    model = vars()[f'model{i+1}']
    model = Model(inputs=model.input,outputs=model.layers[-6].output)
    input_list.append(model.input)
    conc_list.append(model.output)
concatenate = Concatenate()(conc_list)
dense = tf.keras.layers.Dense(units=200,
                              kernel_initializer=tf.keras.initializers.lecun_normal)(concatenate)
batchnorm = tf.keras.layers.BatchNormalization(momentum=momentum)(dense)
activation = tf.keras.layers.Activation('selu')(batchnorm)
dropout = tf.keras.layers.Dropout(0.5)(activation)
output = tf.keras.layers.Dense(units=4)(dropout)

final_model = Model(inputs=input_list,outputs=output)
final_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
                    loss='mean_absolute_error',
                    metrics=['mean_absolute_error'])
```

Figure 4.8 Patch-Based 1DCNN Global Training Source Code

```
pipelines = []
pipelines.append(('LR', Pipeline([('LR',LinearRegression())])))
pipelines.append(('LASSO', Pipeline([('LASSO', Lasso())])))
pipelines.append(('ENet', Pipeline([('EN', ElasticNet())])))
pipelines.append(('KNN', Pipeline([('KNN', KNeighborsRegressor())])))
pipelines.append(('CART', Pipeline([('CART', DecisionTreeRegressor())])))
pipelines.append(('GBM', Pipeline([('GBM', GradientBoostingRegressor())])))

results = []
names = []
for name, model in pipelines:
    kfold = KFold(n_splits=10)
    cv_results = cross_val_score(model, x_train_scaled, y_train_scaled,
                                 cv=kfold, scoring='neg_mean_absolute_error')
    results.append(-cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, np.array(results).mean(), np.array(results).std())
    print(msg)
    results = []
```

Figure 4.9 ML Algorithms Cross Validation Pipeline Source Code

```
from sklearn.model_selection import GridSearchCV

#scaler = StandardScaler().fit(X_train)
#rescaledX = scaler.transform(X_train)
param_grid = dict(n_estimators=np.array([50,75,100,150,200,300,400]),
                  learning_rate = np.array([0.01,0.05,0.1,0.2,0.3,0.5]),
                  max_features = np.array([3,4,5,6,7]))
model = GradientBoostingRegressor()
kfold = KFold(n_splits=5)
grid = GridSearchCV(model, param_grid, scoring='neg_mean_absolute_error', cv=kfold)
grid_result = grid.fit(x_train_scaled, y_train_scaled)

means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

Figure 4.10 GBM Hyperparamter Tuning Source Code

```
from sklearn.model_selection import GridSearchCV

#scaler = StandardScaler().fit(X_train)
#rescaledX = scaler.transform(X_train)
param_grid={"splitter":["best","random"],
            "max_depth" : [1,9,12],
           "min_samples_leaf":[1,3,5,7,9],
           "min_weight_fraction_leaf":[0.1,0.2,0.3,0.4,0.5],
           "max_features":["auto","log2","sqrt",None],
           "max_leaf_nodes":[None,10,20,30,40,50,60,70,80,90] }
#param_grid = dict(n_estimators=np.array([50,75,100,150,200,300,400]),max_features = np.array([3,4,5,6,7]))
model = DecisionTreeRegressor()
kfold = KFold(n_splits=5)
grid = GridSearchCV(model, param_grid, scoring='neg_mean_absolute_error', cv=kfold)
grid_result = grid.fit(x_train_scaled, y_train_scaled)

means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

Figure 4.11 CART Hyperparamter Tuning Source Code

```
model= Sequential(
    [
        Input(shape=(x_train.shape[1],x_train.shape[2])),
        Conv1D(32, 2, activation="relu"),
        BatchNormalization(),
        MaxPooling1D(3),
        Conv1D(32,2, activation="relu"),
        MaxPooling1D(2),
        Conv1D(32,2, activation="relu"),
        Flatten(),
        Dense(64, activation="relu"),
        Dense(16, activation="relu"),
        Dense(1)
    ]
)
def my_loss(y_true, y_pred):
    return K.mean(K.square(y_true-y_pred))/2e+04

model.compile(loss=my_loss,optimizer='adam',metrics=['mean_absolute_error'])
from keras import callbacks
early_stopping = callbacks.EarlyStopping(monitor ="val_mean_absolute_error",
                                         mode ="min", patience = 50,
                                         restore_best_weights = True)
history = model.fit(x_train,y_train,validation_split=0.2,epochs=500,callbacks=[early_stopping])
```

Figure 4.12 Simplified 1DCNN Training Source Code