Linköping University | Department of Computer and Information Science
Master’s thesis, 30 ECTS | Statistics and Machine Learning
2021 | LIU-IDA/LITH-EX-A--2021/001--SE

Structural Health Monitoring
– deep learning approach

Weng Hang Wong

Supervisor : Amirhossein Ahmadian
Examiner : Johan Alenlöv

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Abstract

The development of the sensor on Structural Health Monitoring (SHM) provides some useful data that indicates the condition of the structures. A data-driven approach as an option for achieving the goal of SHM in predicting structural conditions and obtaining useful information from the numerical data. This thesis introduces deep learning methods to perform supervised learning on damage conditions in SHM.

Deep learning methods such as one-dimensional Convolutional Neural Networks (1D-CNN) and Long Short-term Memory (LSTM) are applied to predicting crack position, crack width and deflection of the concrete beams. A Linear Regression (LR) is also investigated to compare with the deep learning models.

Given multidimensional time-series strain data that simulated from finite element methods and the labeled crack positions, 1D-CNN and LSTM models are proposed to handle the binary classification problem. The result shows that an LSTM model is a more promising model than a 1D-CNN model on crack position prediction while handling multidimensional input and output and time-series classification. LSTM model could be a potential solution to achieve automatic monitoring on structural health with only using strain data obtained from DOFS.

In predicting crack width and deflection, a predictive model as LR is also a promising method for solving the regression problem. While exploring different sets of input variables for the LR model, such as strain and geometry variables as inputs, only training with strain data results in a better performance on prediction. 1D-CNN and LSTM models are also implemented and evaluated for comparison with the LR model, which achieved good performance results.

Acknowledgments

I would like to thank my supervisor Amirhossein Ahmadian for his great support and guidance that provides significant helps during each phrase of this work. I would like to give special thanks to my external supervisors, Rasmus Rempling and Carlos Gil Berrocal, for giving me this opportunity to work with their team in SensIT and providing the data for this thesis.

I would also like to thank my examiner Johan Alenlöv for his great advice, and my opponent Vasileios Karapoulios for reviewing my thesis and giving me valuable suggestions.

Last but not least, I would like to thank my family and friends for giving me huge support during this work.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Structural Health Monitoring (SHM) is a process of implementing a damage detection strategy for civil engineering infrastructure, as well as in aerospace and mechanical industry [12]. The structures will deteriorate under several events throughout their lifetime, so damage detection is essential to put in practice at the earliest stage in order to ensure the safety of their users. The successful of automatizing and digitizing SHM can be widely applied on real-life structures, such as bridge, road or buildings that are concrete-based structures, so that to achieve accurate prediction and improve automation without human intervention in the analysis.

This study will be a step forward to solve the damage detection problem on concrete structure base on reinforce concrete (RC) beam experiments by SensIT project. With applying machine learning (ML) or deep learning (DL) methods in SHM can anticipate the damage condition in a robust and accurate manner. These methods aim to extract useful information and hidden patterns from a large amount of raw data, enabling future predictions and decision-making. Many modern approaches have applied ML and DL to optimize SHM; however, some of them fall into the categories of image-based damage detection such as the study by Dorafshan [11] . To inspect the condition of the structures, robots and unmanned aerial systems (UAS) are used to take photos of the structures. These approaches are often limited by the resolution of the images, as well as the pre-processing time on the image-based data.

Recent application of distributed optical fibre sensors (DOFS) is one of the promising solutions on SHM in civil engineering compared with traditional monitoring systems. To capture the strain data from the structure, the fibres are adhered to the surface of the structure and transfer the modulated signal within the fibre when the external condition changes [4]. DOFS improves the efficiency and accuracy of SHM, and it is capable of monitoring strain and temperature, and the methods have been developed to obtain crack widths and deflection simultaneously on reinforcement concrete (RC) structures[7].

Strain data collected from the DOFS are used in damage prediction. The strain data obtained from the reinforced concrete beam gives sequence measurement on current situations along time. The multi-dimensional strain data is essential for damage detection as well as SHM monitoring. The classical approach to predicting the damage conditions is the finite element model and uses a mathematical calculation to achieve the damage forecasts manually. However, the manual calculations are time-consuming and unable to handle a large dataset, so machine learning solves complex and computation expensive problems.

Neural networks can handle multi-dimensional data, and it is proved to be robust on non-linear classification. Convolutional neural networks (CNNs) is one of the families of neural networks; 1D-CNNs has the advantage for time series forecasting and support for multivariate inputs and outputs.

Multivariate time series data also can be tackled by the Long Short-Term Memory network (LSTM), one of the Recurrent neural networks. When learning the mapping function from input to output, LSTM explicit the order between observations that support input data composed of observation sequences[8].

In order to achieve a highly efficient application with DOFS, robust implementation of machine learning methods on DOFS data will be a big step to further improvement in SHM area. In this study, however, the data we used are simulation data from finite element simulation process, since the experimental data is too costly. We mainly focus on investigate with simulation data to train with the models and apply it to the DOFS technology afterwards.

In this thesis, the problem of accurate prediction on crack widths and crack position on RC will be tackled by applying convolutional neural network (CNN), Long Short-Term Memory (LSTM) and linear regression (LR) method and the potential of using machine learning methods on SHM with simulated DOFS data will be discussed.

Related work

In order to scale up the application of SHM in lower economic and labour costs, the recent research on digitalized monitoring is considerably popular; machine learning (ML) and deep learning (DL) techniques have been extensively experimented with and applied on SHM by the researchers. Avila et al. (2005) [3] discussed the applications of artificial neural networks to non-linear cracking phenomena of reinforced concrete structures. Avila et al. used backpropagation and Genetic algorithms in the training section to estimate the crack widths on the reinforced concrete. The study of Claudia (2017) [26] also shows the applications of support vector machine, artificial neural networks, Gaussian processes, etc; these machine learning methods led to the well-performance on predicting the bridge damage condition. Seventekidis (2020) [29] suggested a combination of optimal finite element model and deep learning is a potential solution for future SHM; the study compared the convolutional neural network model with deep learning model for the structure damage identification.

1.2 Aim

This work aims to assess the condition of reinforced concrete (RC) structures by providing a deep learning approach with statistical analysis. From the strain data obtained by sensors on the RC, the structure's condition can be measured consecutively over time, and necessary evaluation and action can be taken on time. The expected outcome of this study is to enhance the monitoring process on RC with prior prediction and diagnosis by integrating modern data analytic techniques and machine learning. The main objectives of this thesis are:

1. Identifying a deep learning method that could be applied in general cases on RC experiments by conducting an accurate prediction on the positions of the cracks, the width of the cracks, and the deflection on RC along the time.
2. Investigating and pre-process the sensor data for the use of the required operations
3. Evaluating the performance of two neural network models (1D-CNN and LSTM) and LR model with test data by assessing their accuracy and reliability.
4. Comparing the result with a statistical-based machine learning method and discuss the confidence of current approaches.

2 Data

2.1 Data source

This thesis work is based on the simulation data collected by SensIT Project located in Gothenburg, Sweden. SensIT is an innovative research project that aims at bringing advanced technologies to enable a new way to monitor, evaluate and inspect the civil infrastructure. A group of researchers in SensIT experiments on reinforced concrete beams with attaching DOFS and subjected to 3-point loading. The experiment is continuously monitored and collected experimental data at the laboratory of Structural Engineering at the Chalmers University of Technology. However, since the experimental data is too costly to collect, so all the data for this thesis is from the simulation of the structural response of a beam, which is convenient for further analysis on quantifying concrete cracks based on the stain distribution patterns.

2.2 Experiment description

2.2.1 The RC beam set-up

The concrete beam has dimensions 3600mm (L) x 250mm (H) x 200mm (D) with DOFS longitudinally bonded to the surface, where the embedding DOFS accesses the performance indicators in terms of vertical defection, crack position and crack width.

2.2.2 The load set-up

The two load cycles were giving pressures on the beam, and it is performed reaching a maximum total load of 60 kN and unloading down to 5 kN total load[5] the meanwhile strain is measured by the DOFS along the beam as shown in Figure 2.1 and Figure 2.2.

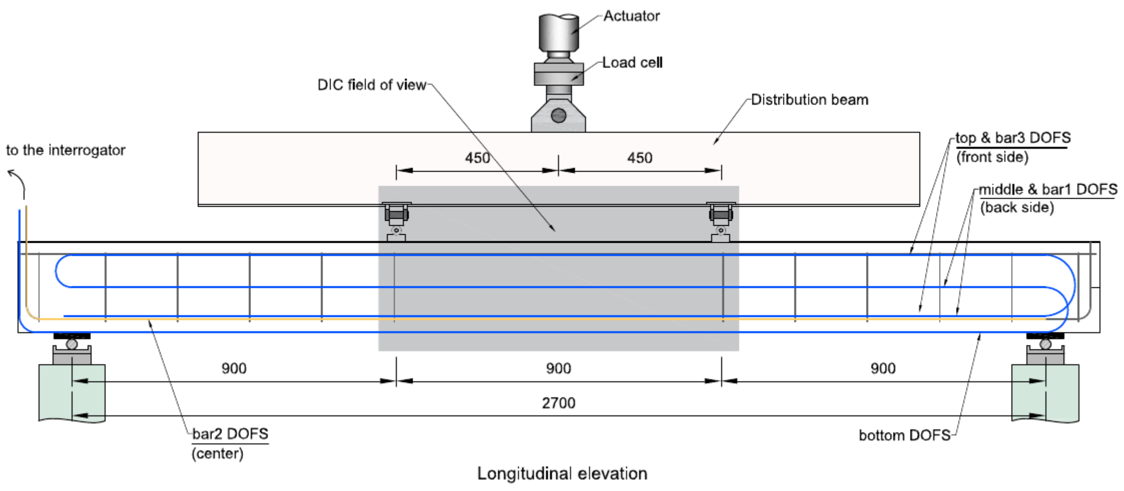

Figure 2.1: Loading setup and DOFS installation configuration for the RC beam specimens. The blue lines are the DOFS that obtains the strain value with names of "middle & bar1", "top & bar3", "bar2" at the front side and back side of the beam.

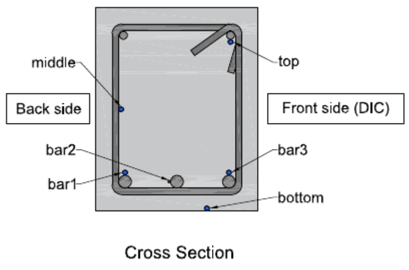

Figure 2.2: Vertical cut of the RC beam with DOFS positions. The small circles are the DOFS.

2.2.3 Simulation process

As aforementioned that the numerical data of this thesis is obtained from finite element simulation. The simulation of the structure is responded of a beam where is modelling with different components (the concrete, the reinforcement and the supports), then some boundary conditions and loads are applied and used to calculate how the model behaves based on different components. After the setup of the the model, a finite element method is applied for numerically approximating the partial differential equations in two or two space domains (i.e., some boundary value problems). The crack (labels) forms in the concrete material using this model, so that we have the ground truths for the crack positions and crack width for our training examples. Similarly, the deflection is extracted from the same model.

By varying the geometry and configuration of the numerical model, it is easy to obtain a large amount of data compared to experimental data. There are total 180 times for the simulation process, and it is also called 180 episodes for the following chapters.

2.3 Data description

The data contains 180 episodes with different geometry settings of the beam simulations. The inputs and outputs come from simulations using different geometry sets and calculated by finite element methods. The variables such as strain values and geometry are helpful to determine the beam damage. The strain data correspond to two load cycles of a beam in each episode, of which recorded the strain variation with different length of time steps. There are observed crack positions, crack width and max deflection variables from the beam experiments, corresponding to the length of strain data. The data has a size of 5 GBs, which in total are 875740 observations. Each observation represents a 1-time step in each beam episode.

Input variables:

1. Geometry: It is the geometrical features of the beam. They are all the same in all episodes because all the data points come from the same beam. The geometry eight variables: [L H W d ns ds ns′ ds′] where these parameters are:
    - L = length of the beam
    - H = height of the beam
    - W = width of the beam
    - d = lever arm of reinforcement
    - ns = number of steel bars in tension
    - ds = diameter of steel bars in tension
    - ns′ = number of steel bars in compression
    - ds′ = diameter of bars in compression
2. Positions: The positions are the length of the beam that divided into 1000 units, which can be seen as a length of 3.6mm of each position along the beam.
3. Bar1: they are the time series strains data measured by the optical fiber position at different locations in the cross-section of the beam. This variable is a matrix where the rows are time steps, and the columns are positions where the strain is measured.

Output variables:

1. Max Deflection: The maximum deflection along the beam for each time step.
2. Cracked positions: A matrix where cracks are already detected (1) or they are still not cracked(0) where the rows are time steps and the columns are the length of positions along the beam.
3. Crack width: A matrix with all the crack widths where the rows are time steps and the columns are the length of positions along the beam.

Beam name	Geometry	Strain (t-n matrix)	Crack position/ Crack width (t-n matrix)	Deflection (1-d array)
	[L H W d ns ds ns′ ds′]	Bar1(t,n)	(t,n)	(time step)
beam_2111	[1800, 150, 75, 117, 1, 10, 2, 8]	(3960, 1000)	(3960, 1000)	(3960, 1)
beam_2112	[1800, 150, 75, 115, 1, 12, 2, 10]	(4360, 1000)	(4360, 1000)	(4360, 1)
beam_2113	[1800, 150, 75, 113, 1, 16, 2, 12]	(5200, 1000)	(5200, 1000)	(5200, 1)
beam_2121	[1800, 150, 90, 117, 1, 10, 2, 8]	(3780, 1000)	(3780, 1000)	(3780, 1)
beam_2122	[1800, 150, 90, 115, 1, 12, 2, 10]	(4180, 1000)	(4180, 1000)	(4180, 1)
beam_2123	[1800, 150, 90, 113, 1, 16, 2, 12]	(4920, 1000)	(4920, 1000)	(4920, 1)
:	:	:	:	:
:	:	:	:	:
beam_3111	[2700, 224, 112, 191, 2, 10, 2, 8]	(5200, 1000)	(5200, 1000)	(5200, 1)
beam_3112	[2700, 224, 112, 190, 2, 12, 2, 10]	(5700, 1000)	(5700, 1000)	(5700, 1)
beam_3113	[2700, 224, 112, 188, 2, 16, 2, 12]	(6560, 1000)	(6560, 1000)	(6560, 1)
beam_3121	[2700, 224, 134, 191, 3, 10, 2, 8]	(5480, 1000)	(5480, 1000)	(5480, 1)
beam_3122	[2700, 224, 134, 190, 3, 12, 2, 10]	(5940, 1000)	(5940, 1000)	(5940, 1)
beam_3123	[2700, 224, 134, 188, 3, 16, 2, 12]	(7080, 1000)	(7080, 1000)	(7080, 1)
:	:	:	:	:
:	:	:	:	:
beam_4111	[3600, 300, 150, 266, 4, 10, 2, 8]	(6840, 1000)	(6840, 1000)	(6840, 1)
beam_4112	[3600, 300, 150, 265, 4, 12, 2, 10]	(7360, 1000)	(7360, 1000)	(7360, 1)
beam_4113	[3600, 300, 150, 263, 3, 16, 2, 12]	(8020, 1000)	(8020, 1000)	(8020, 1)
beam_4121	[3600, 300, 180, 266, 5, 10, 2, 8]	(6900, 1000)	(6900, 1000)	(6900, 1)
beam_4122	[3600, 300, 180, 265, 5, 12, 2, 10]	(7520, 1000)	(7520, 1000)	(7520, 1)
beam_4123	[3600, 300, 180, 263, 4, 16, 2, 12]	(8260, 1000)	(8260, 1000)	(8260, 1)
:	:	:	:	:
:	:	:	:	:
beam_5111	[4500, 375, 187, 341, 6, 10, 2, 8]	(8260, 1000)	(8260, 1000)	(8260, 1)
beam_5112	[4500, 375, 187, 340, 5, 12, 2, 10]	(8600, 1000)	(8600, 1000)	(8600, 1)
beam_5113	[4500, 375, 187, 338, 4, 16, 2, 12]	(9380, 1000)	(9380, 1000)	(9380, 1)
beam_5121	[4500, 375, 224, 341, 7, 10, 2, 8]	(8260, 1000)	(8260, 1000)	(8260, 1)
beam_5122	[4500, 375, 224, 340, 7, 12, 2, 10]	(8880, 1000)	(8880, 1000)	(8880, 1)
beam_5123	[4500, 375, 224, 338, 6, 16, 2, 12]	(9960, 1000)	(9960, 1000)	(9960, 1)
:	:	:	:	:
:	:	:	:	:
beam_6111	[5400, 449, 224, 416, 7, 10, 2, 8]	(9400, 1000)	(9400, 1000)	(9400, 1)
beam_6112	[5400, 449, 224, 415, 7, 12, 2, 10]	(10080, 1000)	(10080, 1000)	(10080, 1)
beam_6113	[5400, 449, 224, 413, 6, 16, 2, 12]	(11140, 1000)	(11140, 1000)	(11140, 1)
beam_6121	[5400, 449, 269, 416, 9, 10, 3, 8]	(9500, 1000)	(9500, 1000)	(9500, 1)
beam_6122	[5400, 449, 269, 415, 8, 12, 2, 10]	(10060, 1000)	(10060, 1000)	(10060, 1)
beam_6123	[5400, 449, 269, 413, 7, 16, 2, 12]	(11140, 1000)	(11140, 1000)	(11140, 1)
:	:	:	:	:
:	:	:	:	:

Table 2.1: The overview of the data. (t,n) represents the matrix size (time steps x position length) of variables in each beam(episode). Each episode has a different set of geometry variables. The length of time-steps are different along with all episodes, and the crack position, crack width, and deflection correspond to the length of strain data.

2.4 Data pre-processing

The three different outputs are crack positions, crack width and max deflection of the beam. Crack positions are 1/0 labels of each position of the beams, but the crack width and max deflection are numeric values. The inputs are strain data and geometry, which are the multi-dimensional variables.

Neural network models are sensitive to the range of the data, and the performance is highly affected by large values or small values in the multi-dimensional dataset. Without normalizing the data, the results might not reflect the actual estimations from the training. Also, normalization is a common technique that if the distribution of the data is unknown or the distribution is not Gaussian. Normalizing the data into a range of zero to one is useful that takes all the input and output variables into the same scale.

As well as in multivariate linear regression, the attributed variables that have a large value will contribute a more significant impact to the result. Therefore, the input for training in all models will be normalized in order to have a promising prediction. The output variable such as crack width and deflection will also be normalized, not only because the output values are small but also the normalized outputs can be compared within different models. The estimators will be comparable with normalized data in regression models and neural networks models. The inputs and outputs are scaled into the range of [0, 1], the min-max scalar is calculated as

$$\sigma = \frac{X - X_{min}}{X_{max} - X_{min}}, \tag{2.1}$$

$$X_{Scaled} = \sigma \cdot (X_{max} - X_{min}) + X_{min}, \tag{2.2}$$

where X_{min}, X_{max} are the minimal and maximal value of the features.

3 Theory

3.1 Linear Regression

A linear regression (LR) model is a common statistical technique to the relationship between input variables $x_1, x_2, ...x_k$ and the output response variable Y, which assumes that the regression function $E(Y|X)$ is linear. With an input vector $X^T = (x_1, x_2, ...x_k)$, a simple linear regression model has a form [14],

$$\hat{y} = \beta_0 + \sum_{j=1}^{k} x_j \beta_j, \tag{3.1}$$

where β_j are unknown parameters or coefficients.

The estimation method of simple linear regression is least squares, in which the coefficients $\beta = (\beta_0, \beta_1, ..., \beta_k)^T$ here to minimize the residual sum of squares as

$$RSS(\beta) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{k} x_j \beta_j)^2, \tag{3.2}$$

The loss function to measure the performance of the simple linear model is MSE. MSE is the average squared differences between the predictors and response values as

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2, \tag{3.3}$$

Multivariate linear regression is a statistical methodology for predicting values of one or more response variables from multiple independent variable values. Let $x_1, x_2, ..., x_k$ be K predictor variables that is corresponding to a target variable Y. The first column of the matrix $\mathbf{X}$ is the multiplier of the constant β_0. Each column of $\mathbf{X}$ contains j values of the corresponding features. Classical multivariate linear regression model[19] as

$$\mathbf{Y} = \mathbf{X}\beta + \epsilon, \tag{3.4}$$

where

$$\mathbf{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_j \end{bmatrix}, \mathbf{X} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1k} \\ 1 & x_{21} & \cdots & x_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{j1} & \cdots & x_{jk} \end{bmatrix}, \beta = \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_k \end{bmatrix}, \epsilon = \begin{bmatrix} \epsilon_0 \\ \epsilon_1 \\ \vdots \\ \epsilon_j \end{bmatrix},$$

where β is unknown parameters and the matrix $\mathbf{X}$ has jth row $[x_{j0}, x_{j1}, ..., x_{jk}]$, the errors ϵ are random values.

The aim of multivariate analysis is to predict the response for given predictor variables. That the values for the regression coefficients β and the error variance σ^2 must be determined with the given data. Assume coefficients β here were the "true" parameter, each $x_j = (x_{j1}, x_{j2}, ..., x_{jk})^T$ is a vector of predictors for the jth case. The least squares method selects β to minimize the sum of squares of the differences as

$$S(\beta) = \sum_{j=1}^{N} (y_j - \beta_0 - \sum_{i=1}^{k} \beta_i x_{jk})^2, \tag{3.5}$$

$$S(\beta) = (\mathbf{Y} - \mathbf{X}\beta)^T (\mathbf{Y} - \mathbf{X}\beta). \tag{3.6}$$

The least-square estimate of β in 3.4 is given by

$$\hat{\beta} = (\mathbf{X}^T \, \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}. \tag{3.7}$$

The residuals is

$$\hat{\epsilon} = y - \hat{y} = [\boldsymbol{I} - \boldsymbol{X}(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X})^{-\mathbf{1}}\boldsymbol{X}^{\boldsymbol{T}}]y. \tag{3.8}$$

where $X^T \hat{\epsilon} = 0$ and $\hat{y}^T \hat{\epsilon} = 0$.

The residual sum of squares as

$$RSS(\beta) = \sum_{j=1}^{N} (y_j - \hat{\beta}_0 - \sum_{i=1}^{K} \hat{\beta}_i x_{jk})^2 = \hat{\epsilon}^T \hat{\epsilon}. \tag{3.9}$$

The loss function to measure the performance of the multivariate linear model is

$$MSE = \frac{1}{2N} \sum_{j=1}^{N} (y_j - \hat{\beta}_0 - \sum_{i=1}^{K} \hat{\beta}_i x_{jk})^2. \tag{3.10}$$

3.2 Deep Learning Methods

This chapter will briefly introduce deep learning methods, followed by the application of Convolutional Neural Networks (CNN) and Long Short-term Memory (LSTM). The prediction of the crack position will be solved by using a 1D-CNN model and compare it with an LSTM model. The prediction of crack width and deflection using LR mentioned in the previous chapter and the details on how to apply with the 1D-CNN model and LSTM will be discussed on Chapter 4.

3.2.1 Introduction of Multi-layer Perceptron

Multi-layer perceptron (MLP) can solve the classification or regression problems that are not linearly separable. MLP is a supervised learning algorithm that learns a function $f(\cdot) : \mathbb{R}^p \rightarrow \mathbb{R}^o$, where p is the number of dimensions for input and o is the number of dimensions for output [27].

MLP is a feed-forward artificial neural network model with one or more layers between the input and output layer; it consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. The layers of nodes in a directed graph, with each layer fully connected to the next one. An MLP often includes an additional bias unit feeding into the input layer and hidden layer, of which is a constant '1' as an additional feature to capture the intercepts in the model[6]. See Figure 3.3.

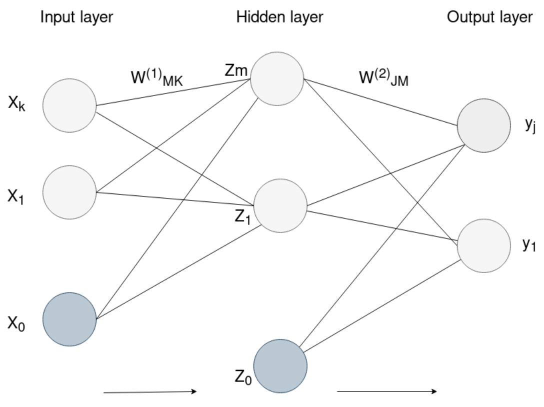

Figure 3.1: The architecture of multi-layer perceptron

A basic neural network model includes a series of functional transformations so that to solve non-linear classification problems. Assumes input variables $x_1, ..., x_k$ and output variables $y_1, ..., y_j$, the weight matrix W_{MK} and the bias units construct a M linear combinations with the input variables. The output is a function of the sum of all inputs transformed by an activation function $h(\cdot)$. The weights and outputs connect the nodes in the network, and the output is a sum of all inputs to a node transformed by a non-linear activation function.
A M linear combinations with the input variables $x_1, ..., x_k$ has a form as

$$a_m = \sum_{k=1}^{K} w_{mk}^{(1)} x_k + w_{m0}^{(1)}, \tag{3.11}$$

where $m = 1, ..., M$ and the superscript (1) represents the first layer of the network, $w_{m0}^{(1)}$ is a bias unit and $w_{mk}^{(1)}$ is the weight parameters of the layer. The quantities a_m is the activations. Each of the activation unit is transformed using a nonlinear activation function $h(\cdot)$ as

$$z_m = h(a_m). \tag{3.12}$$

The transformation of the activation units are corresponding to the outputs, z_m in the neural networks are called hidden units. And these values from 3.12 are again linearly combined to have a output unit activations in the second layer that a form as

$$a_m = \sum_{m=1}^{M} w_{jm}^{(2)} z_m + w_{j0}^{(2)}, \tag{3.13}$$

where $j = 1, ..., J$ and J is the total number of outputs. The superscript (2) represents the second layer of the network, $w_{j0}^{(2)}$ is a bias unit and $w_{jm}^{(2)}$ is the weight parameters of the layer.

Finally, the output is transformed by using a unit activation as logistic sigmoid function

$$y_j = \sigma(a_j), \tag{3.14}$$

$$\sigma(a) = \frac{1}{1 + exp(-a)}, \tag{3.15}$$

where a_j is the intermediate output transformed by an activation unit , $j = 1, ..., J$ and J is the total number of outputs.

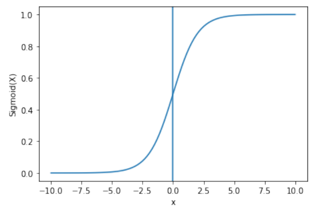

Figure 3.2: Sigmoid function

The overall network function for sigmoidal output unit activation functions have the form as

$$y_j(x, w) = \sigma(\sum_{m=1}^{M} w^{(2)} h(\sum_{i=1}^{K} w^{(1)} x_i + w_{m0}^{(1)}) + w_{j0}^{(2)}). \tag{3.16}$$

All weight and bias parameters are grouped together into a vector **w**. Therefore, the MLP model is an non-linear function combined with input variables **x** and output variables y, where controlled by the weight vector w from first layer and second layer.

3.2.2 Convolutional Neural Networks

Convolutional neural networks, also called CNNs, belong to the family of neural network that for processing a supervised, gird-like topology data[16]. It is first developed by LeCun et al. (1989) [23] as a feed-forward artificial neural network. A one-dimensional (1D) grid CNN commonly takes samples for processing time-series data at certain time intervals, while image data can be processed by a two-dimensional (2D) grid CNN model. The works developed by Wan et al. (2019) [30] and Acquraelli et al. (2017) [1] are successfully applied CNNs to training 1D input on the time series data.

A conventional CNNs is based on a hierarchy architecture of convolutional layers with subsampling layers and followed by (or many) fully connected layers. The basis of the architecture of one-dimensional CNN is similar to a conventional CNN, and it requires 1D input data and 1D filters on the convolution layers[25]. The 1D-CNN performs with both convolution and subsampling layers where are consisted of a number L of layers, and each layer is composed of s feature signals.

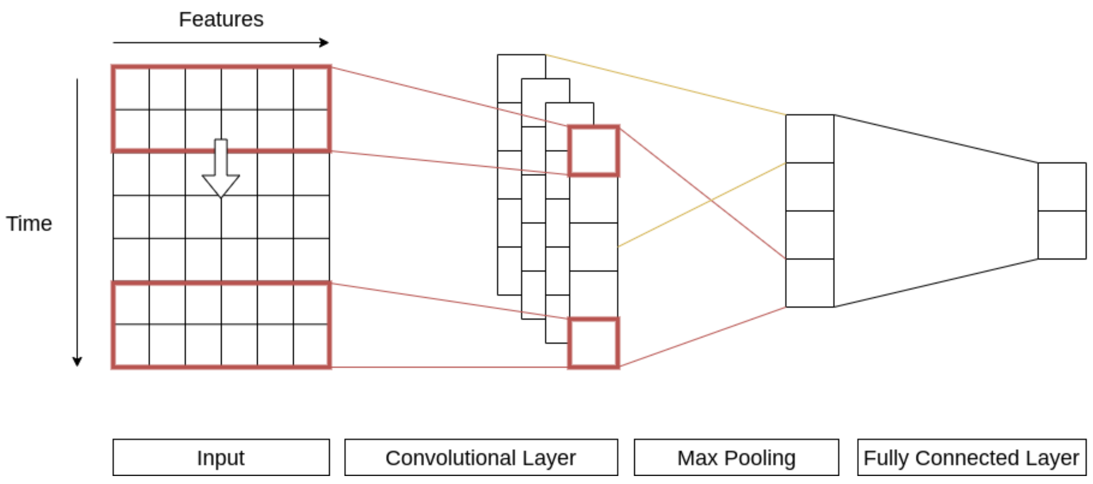

Figure 3.3: 1D Convolutional Network architecture for time series data

1D-CNN layer: The general form of operation in CNN are consisted with two functions of a real-valued argument. A output $x(t)$, where t is position at a time. Both x and t are real-valued. With a weighting (kernel) function w, the convolution operation is denoted as[16]

$$f(t) = (x \cdot w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a), \tag{3.17}$$

where a is the age of a measurement in a weighting function $w(a)$.

Max pooling layer: A max-pooling layer is often seen as a subsampling layer, which selects the maximum output within a vector-like neighbourhood. It reduced the complexity of the output and transformed the input into small amount features; the values of most of the pooled output does not change[16].

Dropout layer: Dropout layer is used to prevent over-fitting in a CNN model. The dropout layer will randomly set an input unit to 0 with a frequency of the dropout rate at each step during training[9]. The model will be less sensitive to slight variations in the training with the dropout layer.

Fully connected dense layer: Fully connected layers connect every neuron in a layer to every neuron in another layer. After the max-pooling layer, the final output is delivered via the last dense layer and transformed by an activation function. A fully connected layer has connections to all layers and activation units to the previous layer; the activation function can be computed with matrix multiplication.

3.2.3 Forward and Back-Propagation in CNN-layers

The current layer is denoted as l, the input $x_1^l, ..., x_j^l$ has the jth neuron at layer, where the input of each neuron at layer is the accumulation of the final output $y_1^{l-1}, .., y_j^{l-1}$of the previous layer $(l-1)$. The 1D forward propagation has a form as

$$x_n^l = b_n^l + \sum_{m=1}^{S_{l-1}} conv1D(w_{n,m}^{(l-1)}, y_m^{(l-1)}), \tag{3.18}$$

where x_n^l is the input of the nth feature signal of the layer l, b_n^l is the bias of the nth neuron at layer l, $w_{n,m}^{l-1}$ is the weight (kernel) from the mth neuron at layer $(l-1)$ to the nth neuron at layer l. $conv1D(\cdot)$ is used to perform 'in-valid' 1D convolution without zero-padding [21].

The intermediate output y_n^l can be transformed by the input x_n^l via the ReLu activation function $\sigma(\cdot)$ [21]:

$$y_n^l = \sigma(x_n^l), \tag{3.19}$$

$$\sigma(x) = max(0, x). \tag{3.20}$$

The ReLU function can help to achieve fast convergence so that to improve the efficiency for training the model.

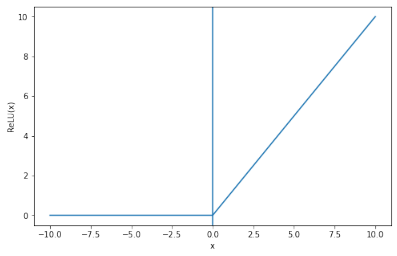

Figure 3.4: ReLU function

Back propagation can evaluates the cost function and minimize the the error by estimate the weights during the process. The back-propagation can be process after computing the first error E and its gradient $\frac{\partial E}{\partial y}$ from the output. The aim of computing the error E is to find out the weights to optimize the model, which means the derivative of the error can be estimated as $\frac{\partial E}{\partial w_{n,m}^l} = \triangle w_{n,m}^l$ [21].

The parameter $\triangle w_{n,m}^l$ can be updated using the chain rules:

$$\triangle w_{n,m}^{l*} = w_{n,m}^l + \eta \frac{\partial E}{\partial w_{n,m}^l}, \tag{3.21}$$

where

$$\frac{\partial E}{\partial w_{n,m}^l} = \frac{\partial E}{\partial y_n^l} \frac{\partial y_n^l}{\partial a_n^l} \frac{\partial a_n^l}{\partial w_{n,m}^l}, \tag{3.22}$$

where $w_{n,m}^{l*}$ is the updated weight for the next iteration, and η is learning rate.

3.3 Long Short-Term Memory (LSTM)

Long and short-term memory networks, often called LSTMs, are a special kind of recurrent neural networks (RNNs) that can learn long-term dependent variables. Hochreiter and Schmidhuber first introduced the idea of LSTMs (1997) [17] , and was improved and promoted by many people afterwards. LSTM performs well on long sequences problems and solves the vanishing error problem during long sequence training. Unlike RNN, which only considers the most recent state, the cell state of LSTM determines which states should be left and which states should be forgotten. The LSTM has been found successful in time series related classification or prediction problems, such as in Karim et al. (2017)[20] and Kwon et al. (2019) [22].

Compared with RNN, which has only one transmission state $y^{(t)}$, where t denoted as the current state and at each state t, the output variables are $y_1, ..., y_j$. (See Figure 3.5). While LSTM has two transmission states, one $C^{(t)}$ as cell state, and one $y^(t)$ as a hidden state. Among them, the $C^{(t)}$ is passed down changes very slowly, and the output $C^{(t)}$ is passed from the previous state $C^{(t-1)}$ plus some values. The $y^{(t)}$ often has a big difference under different nodes. A common LSTM unit contains four interacting layers, a cell, an input gate, an output gate, and a forget gate (See figure 3.6).

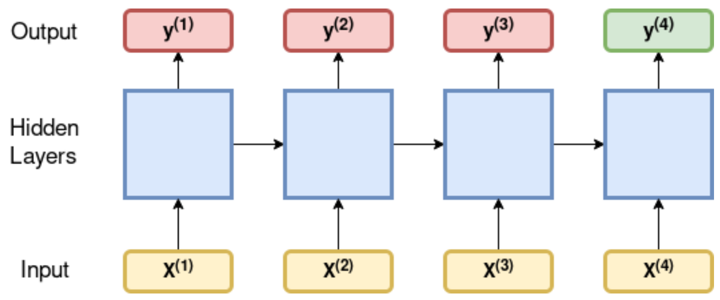

Figure 3.5: An unrolled RNN classification model. The target (green) at the final step.

In order to remember the long-term state, LSTM adds one input and one output based on RNN. The added path is the cell state, which is the top path on the way. In fact, the entire LSTM is divided into three parts:
1. Which cell states should be forgotten,
2. Which new states should be added,
3. According to the current state and current input, what should be the output.
These three parts are corresponding to the work of forget gate, input gate and output gate, respectively.

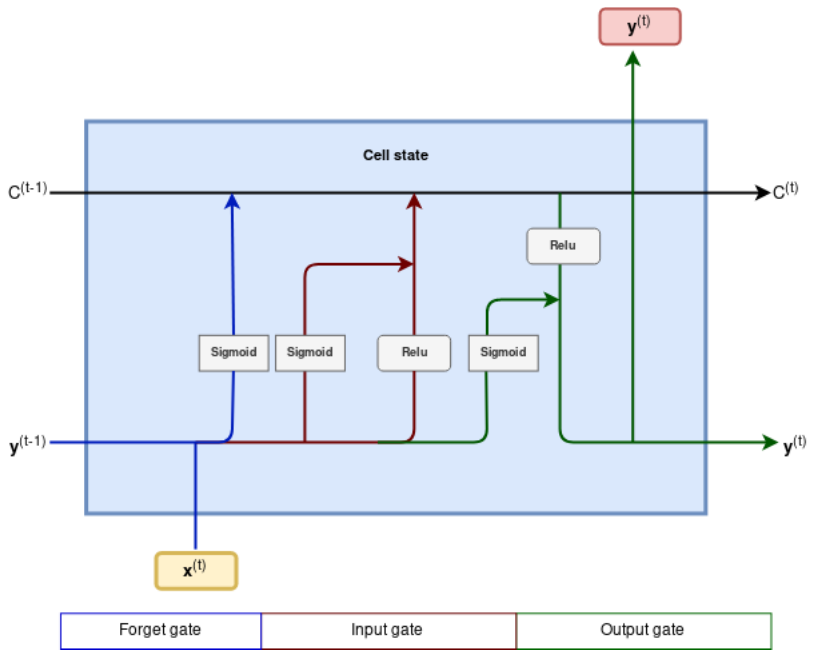

Figure 3.6: One of the repeating module in an LSTM

The cell state crosses straight down the entire chain and transfer the information throughout the processing of the sequence. Each cell has the same inputs and outputs as an RNN, the state units $C^{(t)}$that has a linear self-loop that is controlled by a forget gate unit $f^{(t)}$. The forget gate removes the information that is less important or no longer needed for training LSTM. The sigmoid unit helps the gate to determine which to forget after the given inputs are multiplied by the weight and added a bias unit. The forget gate $f^{(t)}$ defines the weight through a sigmoid unit in a range [0,1] as

$$f_i^{(t)} = \sigma(b_i^f + \sum_j V_{i,j}^f x_j^{(t)} + \sum_j W_{(i,j)}^f y_j^{(t-1)}), \tag{3.23}$$

where f_i^t is for time step t and cell i, $\sigma(\cdot)$ is a sigmoid function, $x^{(t)}$ is the current input vector, $y^{(t)}$ is the current hidden layer vector of which containing the output of all the cells, and b, V, W represent biases, input weights and recurrent weights for the forget gates[16].

The input gate unit $g^{(t)}$ decides the additional information added to the cell state. It is computed similarly to the forget gate and filter the information through a sigmoid unit as

$$g_i^{(t)} = \sigma(b_i^g + \sum_j V_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g y_j^{(t-1)}), \tag{3.24}$$

where the b, V, W represent biases, input weights and recurrent weights for the input gates respectively.

The output gate unit $q^{(t)}$ selects the useful information to the next hidden state from the current cell state and the sigmoid unit helps to decide what should be the output. First, the previous hidden state is put into the ReLu function (instead of tanh function in our case), and multiply it by the output of the sigmoid unit. The output gate has a form similar to the forget gate and input gate as

$$q_i^{(t)} = \sigma(b_i^q + \sum_j V_{i,j}^q x_j^{(t)} + \sum_j W_{i,j}^q y_j^{(t-1)}), \tag{3.25}$$

where the b, V, W represent biases, input weights and recurrent weights for the output gates respectively and the output $y^{(t)}$ of the LSTM cell can be passed via a ReLu function with [16]

$$y_i^{(t)} = ReLU(C_i^{(t)}) q_i^{(t)}. \tag{3.26}$$

Thus, the LSTM cell state $C^{(t)}$ is updated with the forget gate and input gate as follow :

$$C_i^{(t)} = f_i^{(t)} C_i^{(t-1)} + g_i^{(t)} \sigma(b_i + \sum_j V_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}). \tag{3.27}$$

3.4 Evaluation Methods

3.4.1 ROC-AUC

The area under the ROC (Receiver Operating Characteristics) curve, or AUC, has been widely used for evaluating the performance of binary classifiers[18]. The performance measurement of machine learning algorithms is critical; the ROC-AUC curve is a common method to visualize the performance for classification models. It especially offers an alternative measurement to the imbalanced dataset. A ROC-AUC curve is a probability curve that AUC indicates the capacity of the model on distinguishing classes. The higher the AUC, the better the model performs on classification.

In the case of training with imbalanced data, only taking accuracy as a performance indicator will mislead the result of the classifier, where the accuracy is calculated as $\frac{TP+TN}{TP+FP+FN+TN}$. See Table 3.1. In this thesis, there are 980 negative labels and 20 positive labels in each sample, the overall accuracy will be 98%, but it missed classified all the positive data, which should be cared about. Consider the data structure in this thesis, the ROC-AUC method will be used for evaluating the performance of the classifiers.

Assumes P as positive instance classes and $\hat{P}$ as predicted positive classes, N and $\hat{N}$ as negative instance classes and predicted negative classes, respectively. The confusion matrix as follows[13]:

	P	N
$\hat{P}$	TP	FP
$\hat{N}$	FN	TN

Table 3.1: Confusion matrix for classification

In Table 3.1, TP, FP, FN, TN represent the number of True Positives, False Positives, False Negatives and True Negatives.

Additional terms associate with ROC curves are: precision and recall, they are two evaluation metrics that shows how the classifier performs in classifying for the positive class and negative class. Recall metric indicates the rate of relevant items are correctly classified. Precision metric indicates the rate of the classified items are relevant. [13]

True Positive Rate (TPR) / Recall /Sensitivity:

$$TPR = \frac{TP}{TP + FN}. \tag{3.28}$$

Precision:

$$Precision = \frac{TP}{TP + FP}. \tag{3.29}$$

False Positive Rate (FPR) defines the rate of incorrect positive result among all negative samples during the test:

$$FPR = \frac{FP}{TN + FP}. \tag{3.30}$$

F1- score represents the balance between precision and recall, which is calculated by the harmonic mean of the precision and recall:

$$F1score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}. \tag{3.31}$$

Geometric mean (G-mean) is a metric takes the negative rate and positive rate at one specific threshold that balanced both errors. The highest G-mean indicates the location of best threshold between TPR and FPR. (See Figure 3.7). It can be calculated as

$$Gmean = \sqrt{TPR \cdot (1 - FPR)}, \tag{3.32}$$

where the G-mean can also be seen as $\sqrt{Sensitivity \cdot Specifity}$.

ROC curve is a curve that has FPR as x-axis vs. the TPR as y-axis, where the range is between zero to one. The best threshold of probability can be found on the ROC curve, where it is the trade-off between the TPR and FPR. The AUC is equivalent to the probability of a classifier to distinguish classes correctly. If the AUC = 1, means the classifier is able to classify all the positive classes and negative classes correctly; it is opposite if AUC=0. If 0.5<AUC<1, means the classifier is able to classifier the true positive and negative classes than false positive and negative classes. See Figure 3.7.

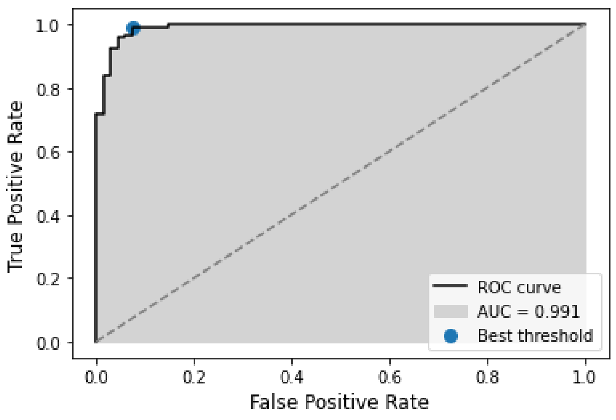

Figure 3.7: ROC curve and a best threshold that selected by the highest G-mean

3.4.2 Error evaluation

To evaluate a regression model, three evaluation metrics are used in this thesis, mean squared error(MSE), root mean squared error (RMSE), and mean absolute error (MAE).

Mean squared error (MSE) is to measure how the model fitted the original data points. It takes the absolute deviation for the target outputs y_i and the predicted outputs $\hat{y}_i$:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{3.33}$$

The RMSE is the square root of MSE :

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}. \tag{3.34}$$

Mean absolute error (MAE) computed the mean absolute error of the target outputs and predicted outputs can be calculated as:

$$MAE = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n}. \tag{3.35}$$

3.4.3 Monte Carlo dropout method

Monte Carlo dropout (MC dropout) is introduced by Gal and Ghahramani (2016)[15], they discovered that the NNs could obtain model uncertainty from dropouts, where dropouts are typically used for preventing over-fitting. The uncertainty on ML is fall on the categories that defines by Der Kiureghian and Ditlevsen[10], which are epistemic uncertainty and aleatoric uncertainty. In this thesis, the MC dropout will be used to capture the epistemic uncertainty of 1D-CNN and LSTM model on the crack prediction.

Epistemic uncertainty can be called as model uncertainty, the uncertainty comes with the unknown weights in all training layers. Theoretically, the uncertainty can be reduced when fitting with more training data [28]. The idea of obtaining epistemic uncertainty based on

Bayesian neural network(BNN). The basic principle of BNN is fo find a posterior distribution $P(W|D)$, where D is the given dataset $\{X, y\}$ and W is the layer weights of NN model parameters. If we use Bayes's formula

$$P(W|D) = \frac{P(D|W)P(W)}{P(D)}, \tag{3.36}$$

where $P(D)$ is the distribution of real data and it is very difficult to get. Hence, here the Monte Carlo method is used as the alternative to approximate $P(D)$.

The essence of Monte Carlo method is to generate an estimation of posterior distribution $P(W|D)$ through a finite number of samples, so that the posterior distribution can be used to obtain epistemic uncertainty. The details of the inference can be found on Gal and Ghahramani (2016)[15]. The MC dropout obtains uncertainty by feeding the test data into the network by T times, the expected mean of MC dropout distribution is the output prediction and the variance is the uncertainty. It has a form as

$$E(y) \approx \frac{1}{T} \sum_{j=1}^{J} x_j, \tag{3.37}$$

$$Var(y) \approx \sigma^2 + \frac{1}{T} \sum_{j=1}^{J} f^W(x_j)^T f^W(x_j) - E(y)^T E(y), \tag{3.38}$$

where $Var(y)$ is the epistemic uncertainty, $f^W(x_j)$ is the predicted output and σ^2 is the data noise. In this thesis, the σ^2 is assumed as 0.

4 Methods

This chapter will elaborate the details on how to apply the models on predicting crack positions, crack width and deflection.

4.1 Sampling Method

In order to reduce the time cost for training the data, a downsampling method is used to reduce the number of the samples to make it faster to process and obtain high quality information. The downsampling method selects the samples in an ordered manner. It selects samples from every *k*th member of the populations.
In this thesis, the data in crack position prediction, crack width prediction, and deflection prediction will be sampling by *k*th step. There are 875740 observations in inputs and outputs, where every *k*th observation will be sampled for the mentioned predictions. Since each episode has different lengths of time steps, efficiently collecting useful information from all populations, a downsampling method can retain the proportional samples according to the length of each episode. Moreover, it is efficient to capture the time-series relationship between the data points.

When using a sampling method, 60% of the subset data are training data, 20% of the subset are test data, and the remaining 20% of the subset are validation data.

4.2 Crack position prediction

4.2.1 1D Convolutional Neural Networks

The 1D-CNN model achieved excellent performance in handling 1D data, such as signal processing mentioned in the previous chapter. To apply on classifying crack position in this case, however, the training labels are imbalanced data containing only a few positive labels. The strain as input data consisted of downsampled time-steps and 1000 positions at each time-step. The output labels are two-dimensional data that corresponds to the input with 1000 positions at each time step. It consists of 1/0 as classes labels where 1 represents crack and 0 represents non-cracks.

In order to handle the imbalanced data with the 1D-CNN model, the training process is specified the class weights on the labelled data. The class weights are calculated for each class is computed as [2]:

$$Class\ weight = \frac{Number\ of\ samples}{Number\ of\ classes \cdot Number\ of\ samples\ in\ current\ class} \tag{4.1}$$

The 1D-CNN model in classification consists of two one-dimensional convolutional layers using ReLU (See Figure 3.4) as activation function, and two max-pooling layers and dropout layers. (See Figure 4.1).

Considering a matrix of strain data as input samples $x_1, x_2, ..., x_j$, where j is the total number of time-steps, and each vector x has k positions that represents the length of the position. The output variables $y_1, ..., y_j$ are two-dimensional targets that each vector y has also 1000 positions that corresponding to the input matrix X. The 1D-CNN classification model performs with both convolution and subsampling layers where are consisted of a number L of layers, and each layer is composed of s feature signals. The last dense layer is a fully connected layer that connects with all the previous layer and transforms each units using a sigmoid function into final outputs (See 3.15)

$$\sigma(x) = \frac{1}{1 + exp(-x)}.$$

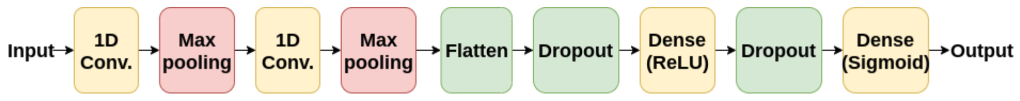

Figure 4.1: The architecture of the 1D-CNN for crack position classification

In order to achieve high performance on classification using the 1D-CNN model, the hyper-parameters needed to tune for the model in the case are: 1) Batch size. 2) Learning rate of the optimizer. 3) Dropout rate in the dropout layers.

The batch size defines the number of observations that train into the network. The network will take less time and require less memory to train with mini-batch size; an optimal mini-batch size can result in sound performance with increasing convergence rate [24]. The parameters of the learning rate can determine the speed of updating weights in the back-propagation operation. A smaller rate results in a smaller amount of the updated weights, which means it slows the network processing time. However, a higher learning rate might fail to find the global minimum since the updating step is too significant.

A hold-out method is applied to the selection of optimal hyper-parameters. Different sets of hyper-parameters result in different performance measured by AUC, and the best threshold is selected according to the highest AUC. The model is first training with 32 batch size with 0.5 dropout rate as a baseline model, and compare the performance within a range of learning rate from 0.001 to 0.1. From the highest AUC value measurement, the result leads to take the best three learning rate compared with a range of batch size from 16 to 128. The selected hyper-parameters from the best performance are compared with a range of dropout rates from 0.5 to 0.9. The best hyper-parameters are tuned by using the hold-out method in this task.

4.2.2 Long Short-Term Memory

In the aforementioned crack position classification using 1D-CNNs in 3.3.2, the weighted classes' imbalanced data is taken care. The class weight is specified on the imbalance data at each time-step, where the calculation is presented as (4.1). While LSTMs possess the ability to handle long term sequences data from the iterations in the networks. In this case, the input and outputs are two-dimensional, which makes the case a sequence-to-sequence model.

Considering a matrix of strain data as input samples $x^{(1)}, x^{(2)}, ..., x^{(t)}$, where t is the total number of time-steps an each vector $x^{(t)}$ has k positions at each time steps. The target variables $y^{(1)}, ..., y^{(t)}$ are matrix outputs that each vector $y^{(t)}$ has k positions at each time-steps where is corresponding to the input matrix X. The LSTM model performs with one LSTM layer, one dropout layer and one dense layer. The output of the hidden state at the last time step of LSTM is using a sigmoid layer.(See Figure 4.2.)

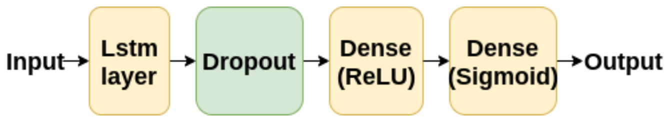

Figure 4.2: The architecture of LSTM for crack position classification

The predicted outputs are the probability of the crack along the length of 1000 positions; in order to find the threshold probability that determined the crack positions, we need to optimize the model and its parameters. To turn the LSTM, the hold-out method is used similar to the classification process of using 1D-CNN in 3.3.2. The model is tuned with the following parameters: 1) Batch size. 2) Learning rate of the optimizer. 3) Dropout rate in the dropout layers.

The model is first trained with 32 batch size, 0.5 dropout rate with adam optimizer, and then it compares with a range of different learning rate within 0.001 to 0.1. Different sets of hyper-parameters result in different performance measured by AUC, and the best threshold is selected according to the highest AUC. Given the results of the different learning rate, it will compare with different batch size from 16 to 128. The hyper-parameter of learning rate and batch size will be selected from the best result and compares with the dropout rate from 0.3 to 0.7. After comparing the final performance, the hyper-parameters will come from the best result of AUC.

4.3 Crack width and deflection prediction

Noticed that not all the strain data along the beam length are useful for the prediction on the crack width, hence, a subset matrix is windowed out by 20 positions from each strain matrix, where each windows is corresponding to the positions of cracks. On the other hand, all the positions are used as the input in the case of predicting the deflection.

4.3.1 Linear Regression

The idea behind the linear regression model is simple, but it is valid and fits well with predicting the crack width and deflection in this thesis. Here the time relationship of the data is eliminated, each time-step of input are assumed to be independent.

To explore how the strain data input and geometry input is useful to the prediction, the linear regression model will be trained with a different set of input variables. First, Let $x_1, ..., x_j$ denoted as the strain data, k as the length of positions of each vector$x$,

$$\hat{y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_j \end{bmatrix}, X = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1k} \\ 1 & x_{21} & \cdots & x_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{j1} & \cdots & x_{jk} \end{bmatrix}, \beta = \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_k \end{bmatrix}, \epsilon = \begin{bmatrix} \epsilon_0 \\ \epsilon_1 \\ \vdots \\ \epsilon_j \end{bmatrix}. \tag{4.2}$$

where X is the strain data with using all features.

All the features of the strain data are sequences value at each time steps, the mean and maximum value of the strain data at each time step will be taken as predictors. The input sets will be compared with Let μ_j represents the mean of the strain input as

$$\hat{y} = \beta_0 + \sum_{j=1}^{k} \mu_j \beta_j, \tag{4.3}$$

and $max(x_j)$ represents the maximum of strain input for each jth time-steps as

$$\hat{y} = \beta_0 + \sum_{j=1}^{k} \max(x_j) \beta_j, \tag{4.4}$$

Also, the mean of the strain data and geometry are taken as input. Let the X represent the strain data and G represent the geometry data, where has p features of geometry data at each time step j, in which the width or deflection output can be represented as $y_1, ..., y_j$, which in time-steps j the output of width and deflection are one-dimensional output. Assumes all the input variables are independent and fit into the model 3.4,

$$\hat{y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_j \end{bmatrix}, \mathbf{X} = \begin{bmatrix} 1 & \mu_1 & G_{11} & \cdots & G_{1p} \\ 1 & \mu_2 & G_{21} & \cdots & G_{2p} \\ \vdots & \vdots & \vdots & \vdots & \\ 1 & \mu_j & G_{j1} & \cdots & G_{jp} \end{bmatrix}, \beta = \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_k \end{bmatrix}, \epsilon = \begin{bmatrix} \epsilon_0 \\ \epsilon_1 \\ \vdots \\ \epsilon_j \end{bmatrix}, \tag{4.5}$$

where X is the strain and geometry combination.

Thus, the linear regression model can be trained with different set of inputs for the prediction of crack width and prediction, with the loss function MSE (See 3.16) as evaluation measurement.

4.3.2 1D Convolutional Neural Networks

In this thesis, the 1D-CNNs will be used to forecast crack width and deflection using the strain data from all 180 episodes. See the architecture in Figure 4.3.

Considering a matrix of strain data as input samples $x_1, x_2, ..., x_j$, where j is the total number of time-steps, and each vector x_j has k lengths of positions. The output variables $y_1, ..., y_j$ are the one-dimensional target that corresponding to the input matrix X. The 1D-CNN performs with both convolution and subsampling layers where are consisted of a number L of layers, and each layer is composed of s feature signals. The only difference between the prediction of crack width and deflection is that deflection prediction uses $k = 1000$ positions for inputs at each time-step, and the input in crack width prediction is windowing by $k = 20$ positions where corresponds with true crack positions.

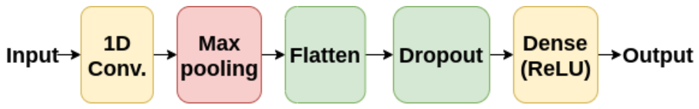

Figure 4.3: The architecture of the 1D-CNN for crack width and deflection prediction

In the aforementioned architecture of the 1D-CNN model, an optimal 1D-CNN model is able to achieve high performance on time-series forecast. A 1D-CNN model is developed from the MLP model; in order to find the optimal 1D-CNN model, hyper-parameters that needed to tune when training the 1D-CNN model for regression are [21]:
1) Number of hidden CNN and MLP layers/neurons,
2) Filter (kernel) size in each CNN layer,
3) Subsampling factor in each CNN layer,
4) Learning rate of the optimizer,
5) Dropout rate in the dropout layers.

The three hyper-parameters out of the above five hyper-parameters are selected to tune an optimal 1D-CNN model. Traditionally, the parameters are tuned using a grid search method with a 3 fold cross-validation, and they are selected by finding the minimum mean squared (MSE). The selected hyper-parameters are the learning rate of the optimizer, dropout rate and the number of hidden layers. The major advantage to have an optimal model that can result in a better performance on the prediction is that a linear operation in a 1D-CNN model has a lower computational cost during the forward and back-propagation operation. See chapter 3.3.3

4.3.3 Crack width prediction using LSTM

The LSTM model for forecasting crack width is similar to 1D-CNN model. The prediction on crack width uses strain data as input samples $x_1, ..., x_j$, where $j$is the total number of time-steps, and each vector x_j has k length of the position. The output variables $y_1, ..., y_j$ are the one-dimensional target that corresponding to the input matrix X. The input and output size is the same as using LR and 1D-CNN, in which the strain data is windowing by 20 positions. The LSTM model consists of one LSTM layer, one dropout layer and one dense layer. The last layer produces the final output that transformed by a ReLU function. See Figure 4.4

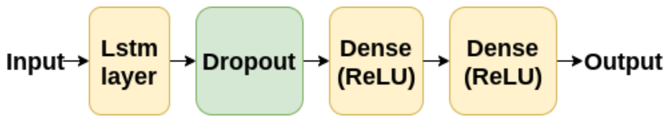

Figure 4.4: The architecture of LSTM for crack width prediction

To tune the LSTM for predicting crack width training with multivariate strain input, a grid search method with three-fold cross-validation is used for finding the best hyper-parameters. The hyper-parameters are selected that achieved the minimum mean squared error (MSE). The candidates of hyper-parameters are the learning rate of the optimizer, dropout rate and the number of hidden layers.

5 Results

5.1 Prediction of crack positions

All the input for training the crack positions is the strain data downsampled from every 20-time steps of 180 episodes of the dataset. The sample size of each episode is the time steps x the length of the positions. Results from the parameter tuning on the test data and validation data are shown with using a downsampling method.

	Sample size
beam_2111	(198, 1000)
beam_2112	(218, 1000)
beam_2113	(260, 1000)
beam_2121	(189, 1000)
beam_2122	(209, 1000)
beam_2123	(246, 1000)
beam_2131	(217, 1000)
beam_2132	(240, 1000)
beam_2133	(284, 1000)
beam_2211	(135, 1000)
beam_2212	(149, 1000)
beam_2213	(144, 1000)
beam_2221	(144, 1000)
:	:
:	:
Total	(43787, 1000)

Table 5.1: Sample size from downsampling method with total 43787 samples

4.1.1 Classification results using CNN

	Training data	Test data	Validation data
Strain data	26272 x 1000	8758 x 1000	8757 x 1000

Table 5.2: Distribution of observations of the strain data obtained using downsampling method

The 1D-CNN model is trained with different learning rate from [0.001, 0.005, 0.01, 0.05, 0.1] using Adam optimizer with 32 batch size and 0.5 dropout rate, where the comparison of AUC, G-mean, F1-score, recall and precision is chosen as the evaluating measurements. The best threshold is selected according to the highest G-mean from the test data set to evaluate the best set of hyper-parameters. Since our goal is to measure how many positive labels we successfully found out in the imbalanced data, the comparison of AUC and recall needs to be taken into account. The results as following:

Learning rate	Best threshold	AUC	G-mean	F1-score	Recall	Precision
0.001	0.00039	0.702	0.654	0.107	0.696	0.058
0.005	0.001564	0.701	0.658	0.107	0.718	0.058
0.01	0.002112	0.695	0.655	0.104	0.792	0.055
0.05	0.00497	0.676	0.638	0.098	0.807	0.052
0.1	0.006049	0.678	0.637	0.098	0.779	0.052

Table 5.3: Performance of 1D-CNN model using different learning rate. The evaluation metrics are included AUC, G-mean, F1-score, Recall and Precision.

From Table 5.3, the result from the test set using the 1D-CNN model with systemic sampling input data is shown. The performance of using a learning rate of 0.001 has the highest AUC 0.702, and the second-best performance is using learning rate 0.005, where achieved 0.701 AUC

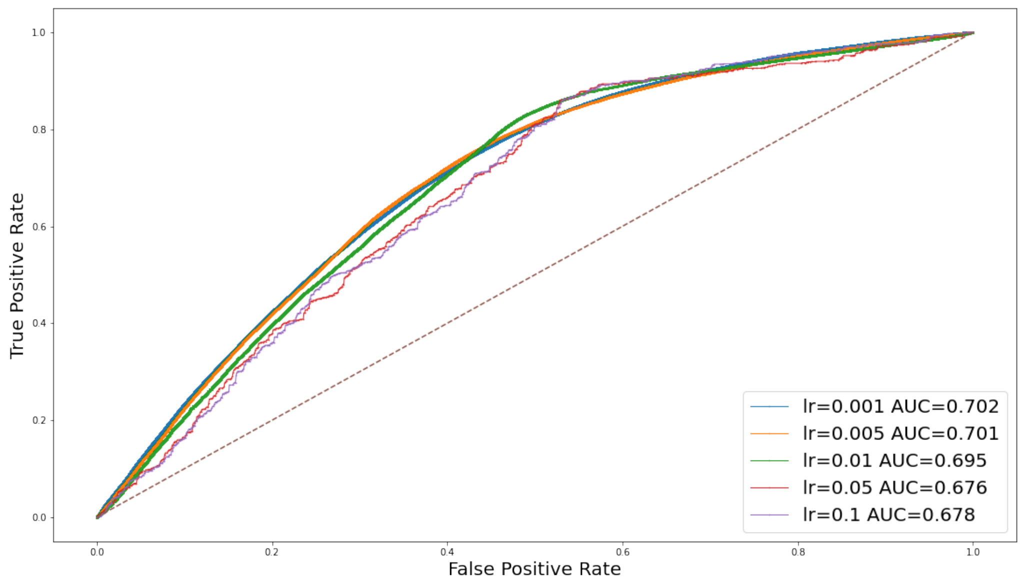

Figure 5.1: Comparison of ROC curve of different learning rate (lr) from 1D-CNN model. Learning rate is compared from [0.001,0.005,0.01,0.05,0,1] with corresponding AUC.

Figure 5.5 visualized the ROC curve with different learning rate from [0.001,0.005,0.01,0.05,0,1] using 1D-CNN model. The ROC curves are calculated from the true positive and false positive rate obtained from the selected best threshold. The ROC curve of learning rate 0.05 has the highest true positive rate, and the ROC curve of 0.001 learning has the highest AUC value.

	Learning rate		
Batch size	0.001	0.005	0.5
16	0.687	0.697	0.677
32	0.702	0.68	0.676
64	0.687	0.685	0.677
128	0.701	0.684	0.678

Table 5.4: Comparison of AUC from different learning rate under different batch size of 1D-CNN model

Table 5.4 shows the AUC performance from the learning rate in [0.001, 0.005, 0.5] and batch size in [16, 32, 64, 128]. AUC is selected as the evaluation measurement to tune 1D-CNN model.

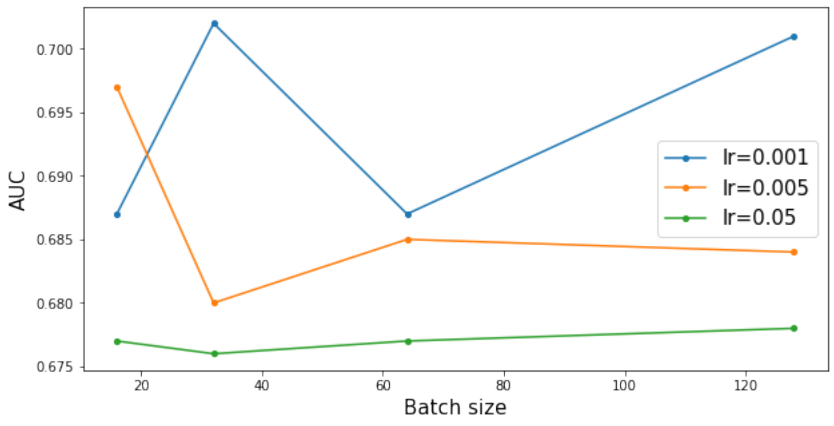

Figure 5.2: Visualization of AUC with different learning rate and batch size of 1D-CNN model

Figure 5.2 visualized the result from the Table 5.4. The figures show an obvious result that the learning rate of 0.001 with batch size 32 achieved the best AUC. The lower the learning rate, the worst the AUC performance achieved from the 1D-CNN model.

	Learning rate
Dropout rate	0.001
0.5	0.690
0.7	0.704
0.9	0.705

Table 5.5: Comparison of AUC from different dropout rate with 0.001 learning rate

Table 5.5, the performance based on the AUC is shown under different dropout rates. Learning rate of 0.001 and batch size of 32 are selected because it achieved the best result from the above Figure 5.2. The dropout rate in the 1D-CNN model can prevent over-fitting. In this case, the best result is from 0.9 dropout rate, where obtained 0.705 AUC from learning rate

0.001.

Best model

The optimal hyper-parameters are selected from tuning with hold-out methods as above. The models are trained on the training set and tested on the test set. The selected hyper-parameters are 0.001 learning rate, 32 batch size and 0.9 dropout rate.

	AUC	G-mean	Recall	Best threshold
Training	0.870	0.789	0.862	0.022855
Test	0.703	0.652	0.777	0.00581
Validation	0.752	0.693	0.748	0.014585

Table 5.6: Performance of crack position prediction from the optimal 1D-CNN model with evaluation metrics of AUC, G-mean and Recall. Best threshold is selected by the highest G-mean.

Table 5.6 shows the result from the selected hyper-parameters for the optimal 1D-CNN model. The selected parameters are obtained from the hold-out method tuning, which are 0.001 learning rate, 32 batch size and 0.9 dropout rate. The model is trained with the training set and predicted on the training set, test set and validation set separately. AUC, G-mean and Recall are chosen as the evaluating measurements for the optimal model performance. The optimal 1D-CNN model achieves good results from the optimal parameters where the AUC reached 0.751 at the validation data.

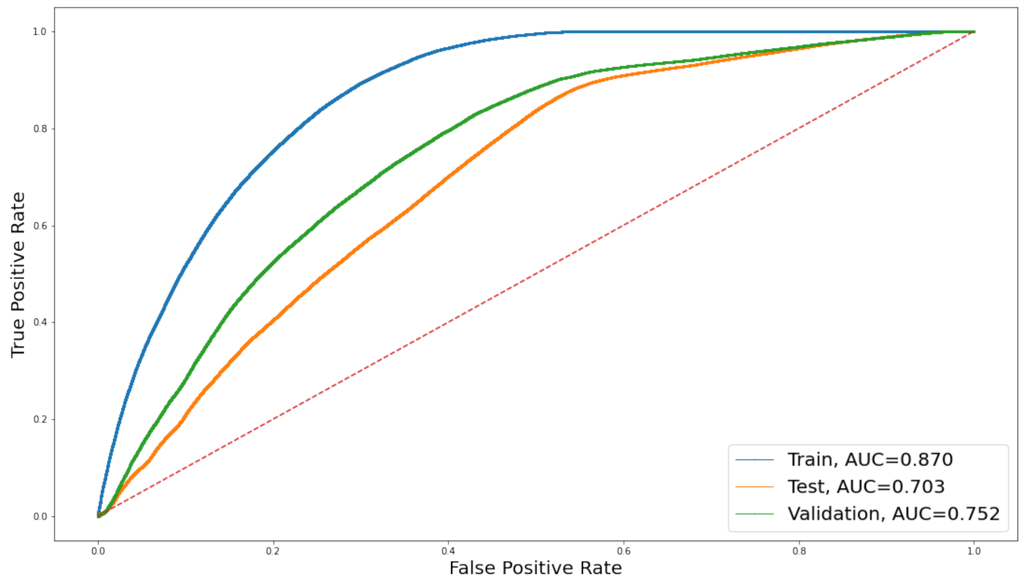

Figure 5.3: Comparison of ROC curve of different datasets of CNN model.

Visualization for the crack position using 1D-CNN model

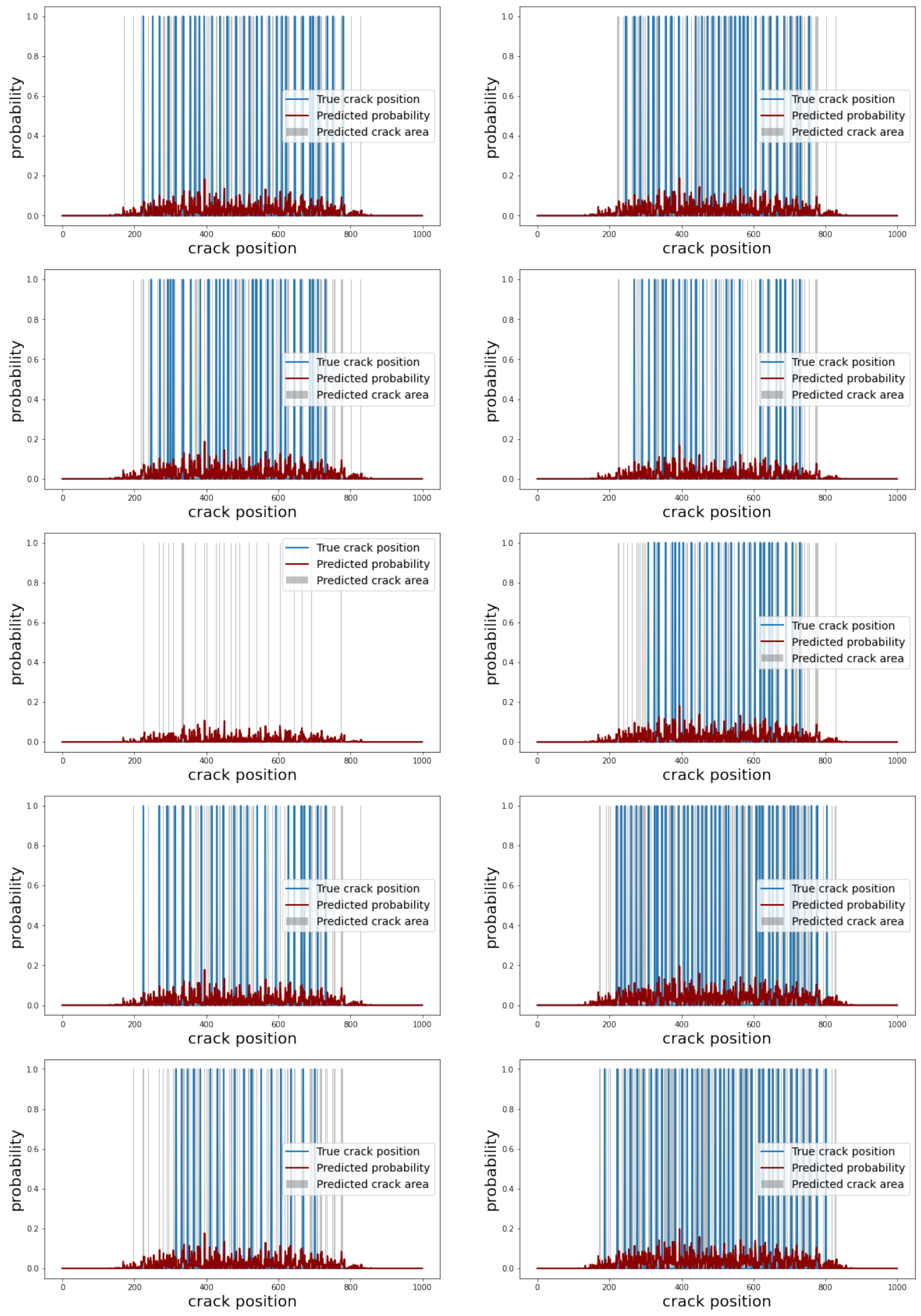

Figure 5.4: Visualization of crack position prediction at random time steps from validation set with using the optimal 1D-CNN model. Blue lines represent the true crack position. Red lines represent the predicted probability of cracks along the positions. Grey area is selected by using the best threshold that indicates the possible crack positions.

In Figure 5.4, the prediction on the validation set is trained with the optimal 1D-CNN model. The ten plots are selected randomly from the validation set for the purpose of illustrating the prediction results. The blue lines represented the true crack positions on the beams, the red lines are the predicted probability output from 1D-CNN model, and the grey area is selected by using the best threshold that indicates the possible crack positions. From the overall results, although the AUC of the optimal 1D-CNN model achieved 0.752 on the validation data, the plots in the Figure 5.4 does not found obvious correlation between the prediction probability and the true crack position.

4.1.2 Classification results using LSTM

The data is split into training, test and validation set from the samples using downsampling method:

	Training data	Test data	Validation data
Strain data	26272 x 1000	8758 x 1000	8757 x 1000

Table 5.7: Distribution of observations of the strain data obtained using downsampling method

Learning rate	Best threshold	AUC	G-mean	F1-score	recall	precision
0.001	0.034057	0.896	0.819	0.209	0.880	0.119
0.005	0.016100	0.880	0.769	0.186	0.866	0.104
0.01	0.030155	0.879	0.803	0.19	0.876	0.107
0.05	0.023400	0.894	0.817	0.2	0.88	0.117
0.1	0.026003	0.890	0.815	0.203	0.881	0.115

Table 5.8: Performance of LSTM model using different learning rate. The evaluation matrices are included AUC, G-mean, F1-score, Recall and Precision

Table 5.8, the result from the test set with the LSTM model is shown. The LSTM model is trained with different learning rate from [0.001, 0.005, 0.01, 0.05, 0.1] using Adam optimizer with 32 batch size, where the comparison of AUC, G-mean, F1-score, recall and precision is chosen as the evaluating measurements. The best threshold is selected according to the highest G-mean from the test data set so that to evaluate the best set of hyper-parameters. Our goal is to measure how many positive labels that we successfully found out in the imbalanced data. The AUC and recall are the measurements that need to be taken into account.

Table 5.8, the performance of using a learning rate of 0.001 has the highest AUC 0.896, and G-mean 0.819, and has the second-highest recall value 0.880. The second-best performance uses a learning rate of 0.05, where achieved a 0.890 AUC and a 0.817 G-mean with a 0.881 recall.

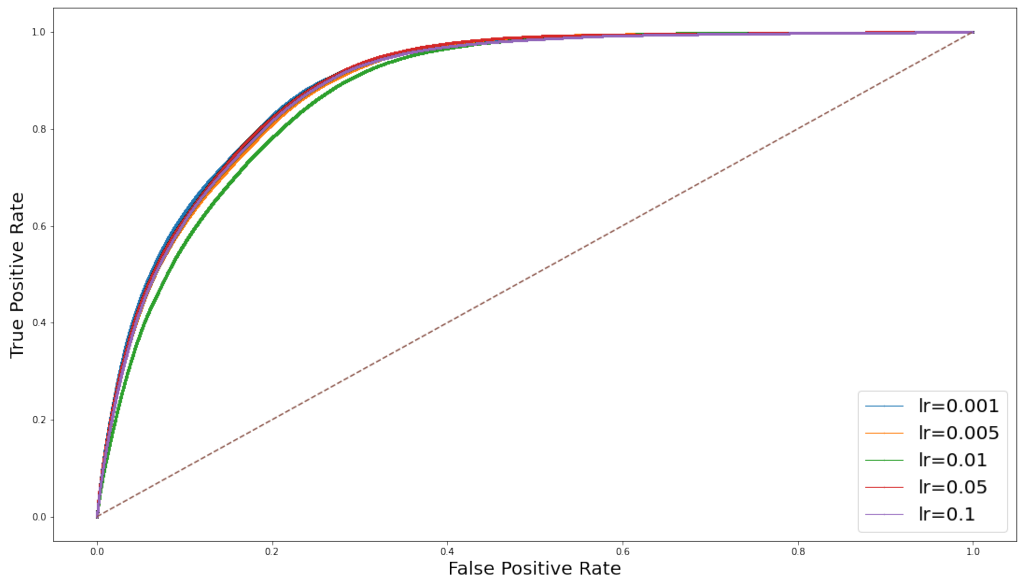

Figure 5.5: Comparison of ROC curve of different learning rate (lr) from LSTM model. The learning rate are [0.001, 0.005, 0.01, 0.05, 0.1].

Figure 5.5 shows the ROC curves comparing with different learning rate from [0.001, 0.005, 0.01, 0.05, 0.1]. The ROC curves are calculated from the true positive rate and false positive rate obtained from the best threshold of the above Table 5.8. Figure 5.5 the green line represented the learning rate of 0.01, which has the lowest AUC, compared with other ROC curves. The learning rate from 0.001, 0.005, 0.05 and 0.1 are very similar to each other.

	Learning rate		
Batch size	0.001	0.005	0.05
16	0.884	0.896	0.890
32	0.896	0.880	0.894
64	0.878	0.878	0.886
96	0.850	0.870	0.873
128	0.870	0.868	0.871

Table 5.9: Comparison of area under the ROC (AUC) under different batch size

In Table 5.9, the result from the test set with LSTM model and systemic sampling is shown. AUC is chosen as the evaluating measurement in tuning the LSTM model. The LSTM model is trained with a different learning rate from [0.001, 0.005, 0.05], which is selected based on the previous result's performance. See Figure 5.5. The model uses an Adam optimizer with different batch size from [16, 32, 64, 96, 128].

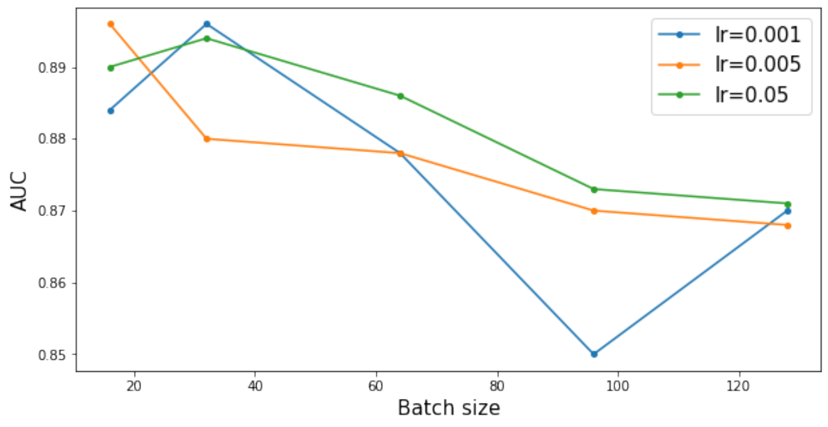

Figure 5.6: Comparison of AUC with different learning rate and batch size

Figure 5.6 visualized the results from the Table 5.9. The plot shows the AUC variation during using the different batch size and comparing it with different learning rate. It indicated that the mini-batch size has better performance with different learning rate. The batch size of 16 with a learning rate of 0.005, the batch size of 32 with a learning rate of 0.001 has the highest AUC value. The second-best AUC is from training with 32 batch size with a 0.05 learning rate. Therefore, the next training will focus on tuning the model with 32 batch size.

	Learning rate		
Dropout rate	0.001	0.005	0.05
0	0.884	0.892	0.879
0.1	0.895	0.888	0.900
0.3	0.891	0.892	0.897
0.5	0.896	0.880	0.894
0.7	0.888	0.892	0.812

Table 5.10: Comparison of AUC with different learning rate and dropout rate

In Table 5.10, the result from the test set with LSTM model and systemic sampling is shown. The result shows the AUC comparison between the learning rate [0.001, 0.005, 0.05] and dropout rate [0, 0.1, 0.3, 0.5, 0.7], with training with 32 batch size.

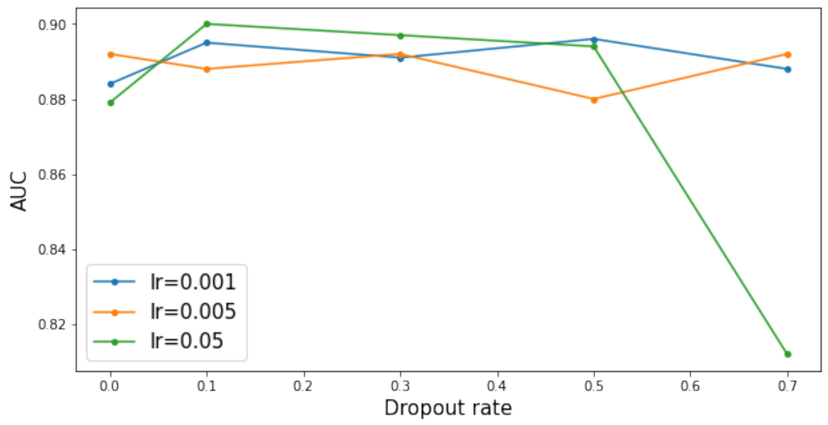

Figure 5.7: Comparison of AUC with different learning rate and dropout rate

Figure 5.7 visualized the results from the Table 5.10 with different learning rate in [0.001, 0.005, 0.05] and dropout rate [0, 0.1, 0.3, 0.5, 0.7]. From this figure, the highest AUC is at the 0.1 dropout rate with 0.05 learning rate, which achieved 0.9 AUC. The worst AUC is shown at the dropout rate of 0.7 with 0.05 learning rate, other than this, the overall AUC is concentrated between 0.88 to 0.90.

The best LSTM model

The best hyper-parameter is selected from the above hold-out method from tuning different hyper-parameters of the learning rate, batch size, and dropout rate for the LSTM model.

Learning rate	Batch size	Dropout rate
0.05	32	0.1

Table 5.11: Best selected hyper-parameters

	AUC	G-mean	Recall	Best threshold
Training	0.919	0.841	0.898	0.019690
Test	0.897	0.817	0.872	0.019931
Validation	0.905	0.827	0.880	0.022693

Table 5.12: Performance of crack position prediction from the best LSTM model

Table 5.12 shows the result from the selected hyper-parameters for the best LSTM model. The model is trained with a training set and fitted with a training set, test set, and validation set. AUC, G-mean and Recall are chosen as the evaluating measurements for the model performance. Both the test and validation set are achieved a high AUC value with the best hyper-parameters, which is 0.897 and 0.905

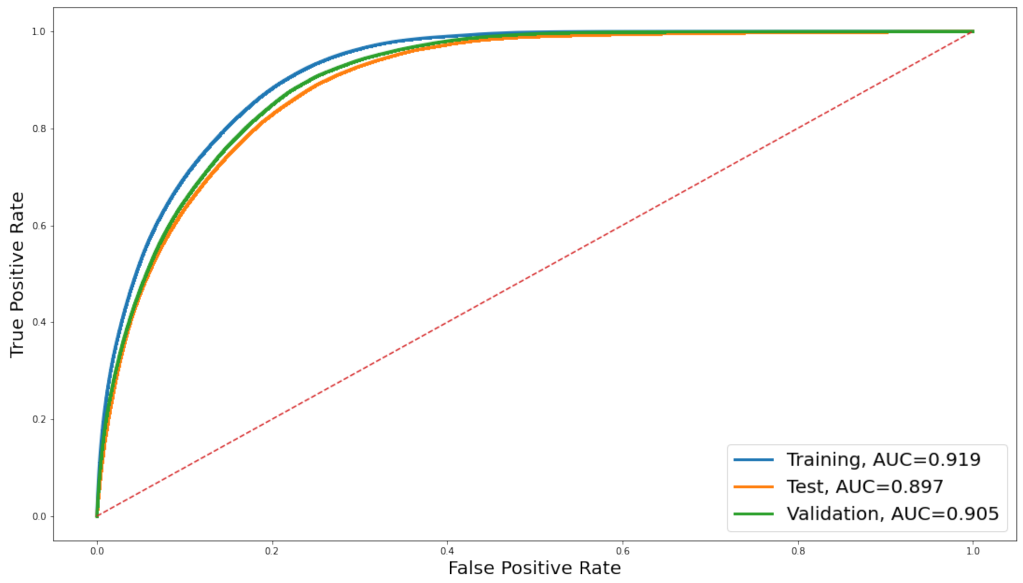

Figure 5.8: ROC curves compared with training, test and validation dataset

Figure 5.8 shows the ROC curve from the training, test and validation set fitted from the best LSTM model that trained with the training set. From the plot, we can clearly see that the training ROC curve reached the highest AUC, and the validation set reached the second-highest AUC. It implied that the best LSTM model is able to achieve good performance for crack position prediction.

Visualization for the crack position prediction

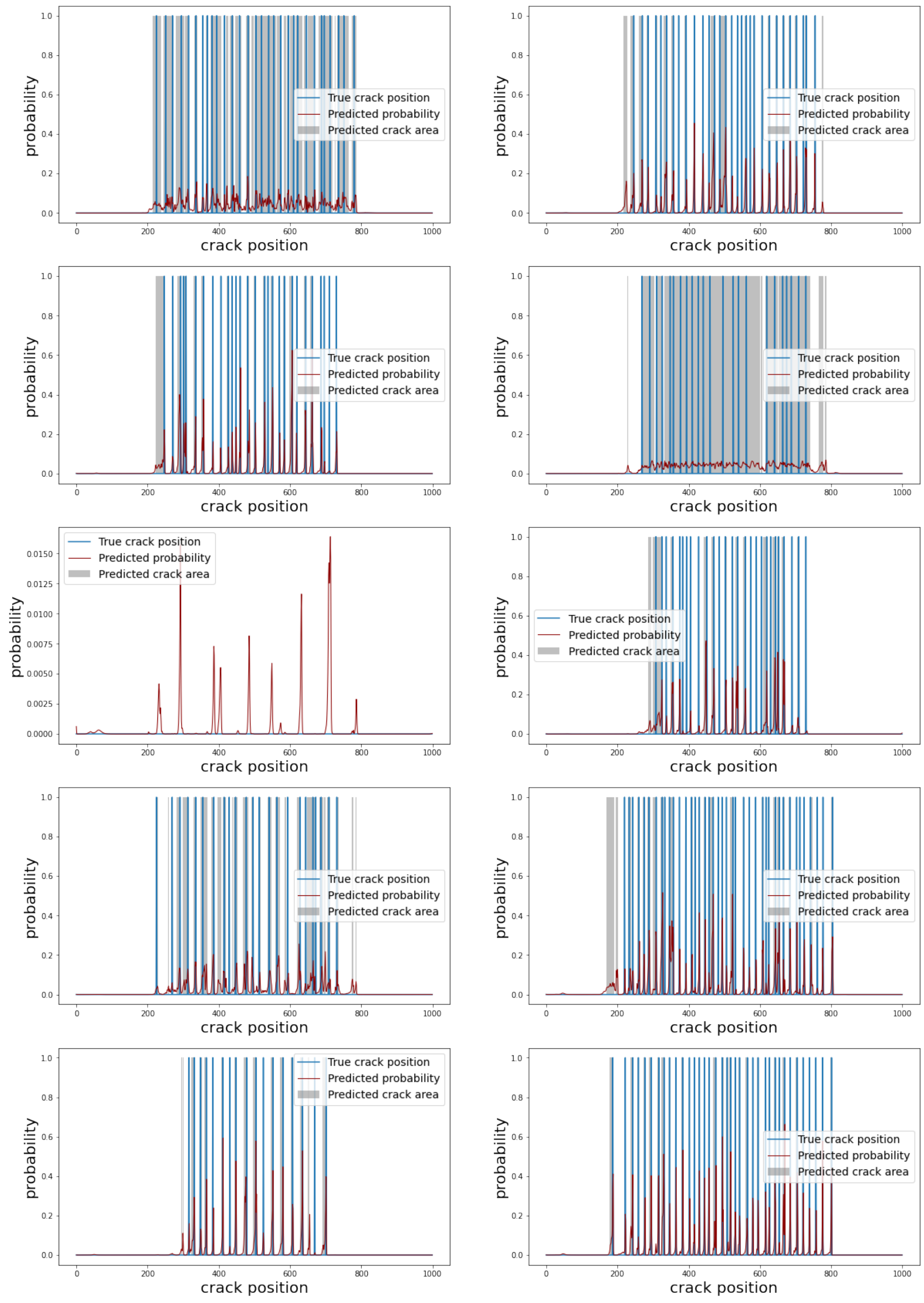

Figure 5.9: Visualization of crack position prediction at random time-steps from validation set with using the optimal LSTM model. Red lines represent the predicted probability of cracks along the positions. Blue lines represent the true crack position. Grey area is selected by suing the best threshold that indicates the possible crack positions.

Figure 5.9, the prediction on the validation set is trained from the optimal LSTM model on the selected best hyper-parameters. The ten plots are the prediction results taken out from the validation set randomly for the visualization propose; each plot represented a random time-step in the validation set. In each plot, the grey area is the predicted crack position area that reached the predicted probability of the best threshold 0.022693. See Table 5.12. The blue vertical lines represented the crack position labels, 1 means cracks, and 0 means no cracks. The red lines are the predicted probability variations that along the beam length (crack positions); if the predicted probability is too low, there will not be any predicted crack area in the plot.

The above results indicated that the predicted probability from the LSTM model correlates with the crack positions at first glance. Moreover, according to the performance results, the optimal LSTM model has a good prediction on crack position with imbalanced data labels.

5.2 Prediction of crack width

The strain data are sampled using the downsampling method from the total of 180 episodes of the dataset. Each crack width input data is sliced from the strain data matrix with 20 position length, corresponding to the crack's position. Therefore, the useful data is taken out from the position length of each episode. The sample input is sampled from every 100-time steps from each beam episode to keep the information along the time.

	Size of variables	
Beam names	Strain	Geometry
beam_2111	(357, 20)	(357, 8)
beam_2112	(567, 20)	(567, 8)
beam_2113	(832, 20)	(832, 8)
beam_2121	(341, 20)	(348, 8)
beam_2122	(586, 20)	(586, 8)
beam_2123	(689, 20)	(689, 8)
beam_2131	(695, 20)	(695, 8)
beam_2132	(816, 20)	(886, 8)
:	:	:
:	:	:
Total size	(319668, 20)	(319668, 8)

Table 5.13: Sample size of each dataset for crack width prediction with total 319668 samples. The sample size of strain data is (time-steps x the length of position), The sample of geometry is (time-steps x number of features).

	Training data	Test data	Validation data
Strain data	191800 x 20	63934 x 20	63934 x 20
Geometry	191800 x 8	63934 x 8	63934 x 8

Table 5.14: Distribution of observations of the strain data obtained using downsampling method from each 100 time-steps.

4.2.1 Prediction using linear regression for crack width prediction

The linear regression model will be trained with different set of inputs as following:

Input training set	Input size
Max_Strain	(191880, 1)
Mean_Strain	(191880, 1)
All_Strain	(191880, 20)
Mean_Strain+geo	(191880, 9)

Table 5.15: Selection of input variables set and their input shape. Max_strain: the maximum strain value of all position at each time step. Mean_strain: the mean strain value of all position at each time step. All_strain: all of the position (1000 length) that the strain is taken. Mean_Strain+geo: the Mean_strain variable plus eight variables from the geometry features.

In table 5.15 shows the training size of a different set of input variables. We would like to see how the input variables in the linear regression model would affect the performance. Therefore, the input sets are taken into account of the max value of strain data in each time step, the mean value of strain data in each time step, as well as the original strain data that sliced with 20 lengths of positions. Also, the geometry variables with eight features at each step are taken with the mean of strain data as nine feature input variables. Therefore, there is a total of four sets of inputs to fit into the linear regression model separately.

	MSE	MAE	RMSE
Max Strain	0.007	0.059	0.086
Mean Strain	0.007	0.058	0.082
All Strain	0.004	0.046	0.064
Mean Strain + geo	0.007	0.060	0.081

Table 5.16: Performance of linear regression model using different set of variables as predictor

Table 5.16 shows the performance of using a different set of variables as input using a linear regression model fitted with test data. The MSE, MAE and RMSE are the measurements for evaluating the performance of the different set of inputs. From the results, using all 20 positions of the strain data (191800 x 20) has the best performance among other input sets, where achieved the smallest MSE, MAE and RMSE. By investigating the MAE and RMSE values, the mean strain with geometry (191800 x 9) 9 features variables had the worst performance, which meaning that the geometry data is unnecessary to input variables comparing only using strain data as the predictor. Using input with max and mean value of the strain is comparably better than the stain+geo input, but worsen than using all features of strain value, which means that the max and mean value did not contain as much information to have a better result on the prediction.

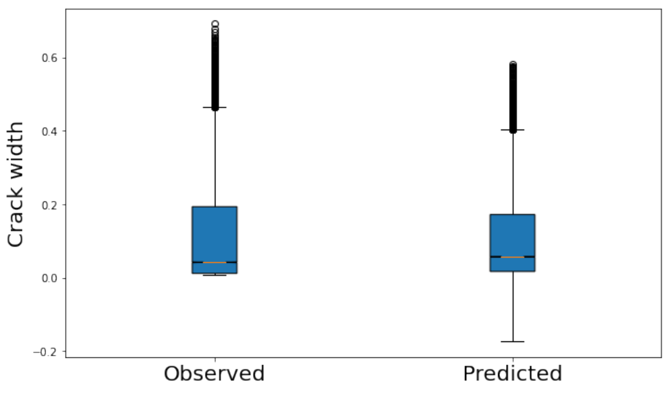

Figure 5.10: Observed and predicted crack width for optimal linear regression model. The black line in the middle of the blue box is median. The black points on the top of the blue box are the outliners.

Figure 5.10 shows a box plot comparing observed data and predicted data from the optimal linear regression model. The model predicted validation data using all 20 positions of the strain data (191800 x 20) as training data. The horizontal line in the blue box is the median of the crack width data.

4.2.2 Prediction using 1D-CNN for crack width prediction

	Training data	Test data	Validation data
Strain data	191800 x 20	63934 x 20	63934 x 20

Table 5.17: Distribution of observations of the strain data obtained using downsampling method

For tuning an optimal 1D-CNN model, the hyper-parameters of the number of dense layers, optimizer, dropout rate, and learning rate will be optimized by cross-validation method.

Number of dense layers	Optimizer	Dropout rate	Learning rate
[0, 1 ,2]	[adam, rmsprop]	[0.3,0.5,0.7]	[0.001,0.005,0.05]

Table 5.18: Hyper-parameters for tuning 1D-CNN model

	Number of dense layers	Optimizers	Dropout rate	Learning rate
Best parameters	2	adam	0.5	0.001

Table 5.19: The optimal hyper-parameters selected from the grid search using 3-folds cross validation

Table 5.19 shows the optimal hyper-parameters selected after optimized by grid search method with 3-folds cross-validation, which taken the lowest error among different sets of hyper-parameters. The model is trained with the training set. The optimal hyper-parameters has 2 dense layers, 0.5 dropout rate, 0.001 learning rate with an ’adam’ optimizer.

	MSE	MAE	RMSE
Training	0.0028	0.037	0.053
Test	0.0024	0.035	0.049
Validation	0.0023	0.033	0.048

Table 5.20: Performance of crack width prediction from the optimal 1D-CNN model

Table 5.20 shows the result from the optimal hyper-parameters for the best 1D-CNN model. The model is trained with a training set and fitted with a training set, test set and validation set separately. MSE, MAE and RMSE are chosen as the evaluating measurements for the performance. The validation set has the lowest error rate among the performance metrics by using the optimal hyper-parameters.

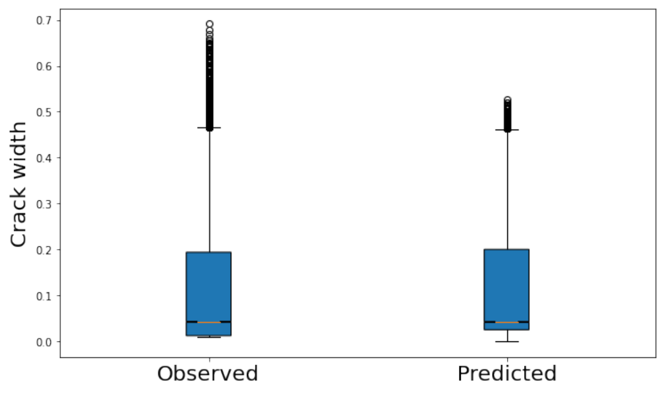

Figure 5.11: Observed and predicted crack width for optimal 1D-CNN model. The horizontal line in the middle of the blue box is median. The black points on the top of the blue box are the outliners.

Figure 5.11 shows a box plot comparing observed data and predicted data from the optimal 1D-CNN model. The model predicted validation data using all 20 positions of the strain data (191800 x 20) as training data. The horizontal line in the blue box is the median of the crack width data.

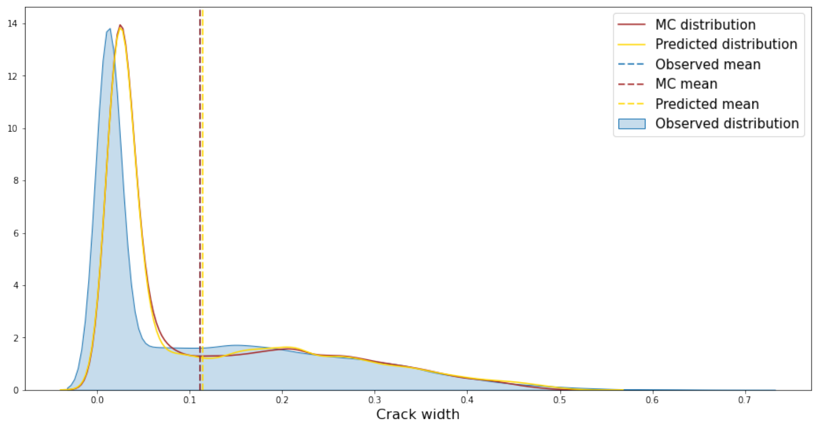

Figure 5.12: The distribution comparison of Monte Carlo dropout, predicted and observed.

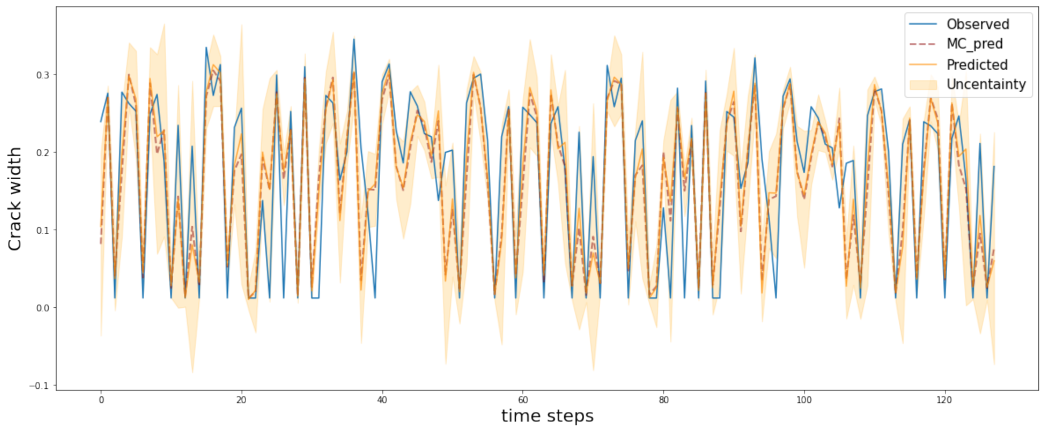

Figure 5.13: Uncertainty of 128 samples extracted from validation set using Monte Carlo dropout. The grey area is the predictive uncertainty.

4.2.3 Prediction using LSTM for crack width prediction

	Training data	Test data	Validation data
Strain data	191800 x 20	63934 x 20	63934 x 20

Table 5.21: Distribution of observations of the strain data obtained using downsampling method

For tuning an optimal LSTM model, the hyper-parameters of number of dense layers, optimizer, dropout rate and learning rate are going to be optimized by cross validation method.

Number of dense layers	Optimizer	Dropout rate	Learning rate
[0, 1 ,2]	[adam, rmsprop]	[0.3,0.5,0.7]	[0.001,0.005,0.05]

Table 5.22: Hyper-parameters for tuning LSTM model

	Number of dense layers	Optimizers	Dropout rate	Learning rate
Best parameters	1	adam	0.7	0.005

Table 5.23: The optimal hyper-parameters selected from grid search method using 3-folds cross validation

Table 5.23 shows the best hyper-parameters trained from the grid-search with 3-fold cross-validation using the parameter candidates from Table 5.22, which taken the lowest error rate among all the sets of hyper-parameters. The model is trained with a training set. The optimal hyper-parameters are 2 dense layers, a 0.7 dropout rate, and a 0.005 learning rate with the 'adam' optimizer.

	MSE	MAE	RMSE
Training	0.0032	0.030	0.057
Test	0.0028	0.029	0.053
Validation	0.0025	0.027	0.050

Table 5.24: Performance of crack width prediction from the optimal LSTM model

Table 5.24 shows the result from the optimal hyper-parameters for the best LSTM model. The model is trained with a training set and fitted with a training set, test set and validation set separately. MSE, MAE and RMSE are chosen as the evaluating measurements for the performance. The validation set has the lowest error rate among the performance metrics using the optimal hyper-parameters.

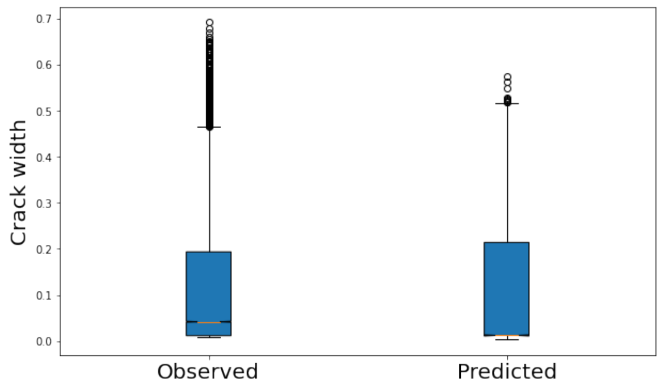

Figure 5.14: Observed and predicted crack width from optimal lstm model. The horizontal line in the middle of the blue box is median. The black points on the top of the blue box are the outliners.

Figure 5.14 shows a box plot comparing observed data and predicted data from the optimal LSTM model. The model predicted validation data by using all 20 positions of the strain data (191800 x 20) as training data. The horizontal line in the blue box is the median of the crack width data.

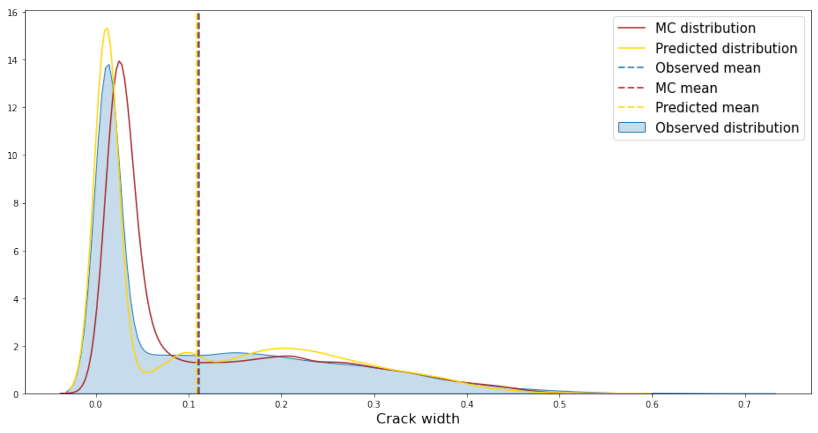

Figure 5.15: The distribution comparison of Monte Carlo dropout, predicted and observed.

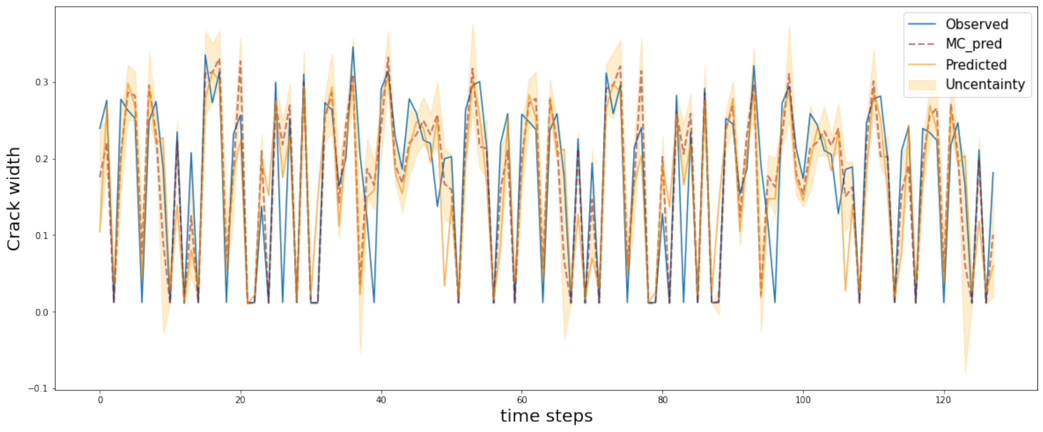

Figure 5.16: Uncertainty of 128 samples extracted from validation set using Monte Carlo dropout. The grey area is the predictive uncertainty.

5.3 Prediction of deflection

The strain data are down-sampled from total of 180 episodes of the dataset. All the inputs and outputs for deflection are sampled from every 50-time steps of each episode. The sample size of each episode is time steps x the length of the positions. Results from the parameter tuning on the test and validation data are shown using Linear Regression and 1D-CNN.

	Size of varaibles	
Beam names	Strain	Geometry
beam_2111	(80, 1000)	(80, 8)
beam_2112	(88, 1000)	(88, 8)
beam_2113	(104, 1000)	(104, 8)
beam_2121	(76, 1000)	(76, 8)
beam_2122	(84, 1000)	(84, 8)
beam_2123	(99, 1000)	(99, 8)
beam_2131	(87, 1000)	(87, 8)
beam_2132	(96, 1000)	(96, 8)
beam_2133	(114, 1000)	(114, 8)
:		:
:		:
Total size	(17584, 1000)	(17584, 8)

Table 5.25: Input size from each episode for deflection with total 17584 input samples. The sample size of strain data is (time-steps x the length of position). The sample size of geometry data is (time-steps x the features of geometry)

4.3.1 Deflection Prediction using Linear Regression

	Training	Test	Validation
Strain data	(10550, 1000)	(3517, 1000)	(3517, 1000)
Geometry	(10550, 8)	(3517, 8)	(3517, 8)

Table 5.26: Distribution of observations of the strain data obtained from down-sampling method

Input training set	Input size
Max Strain	(10550, 1)
Mean Strain	(10550, 1)
All Strain	(10550, 1000)
Mean Strain+geo	(10550, 9)

Table 5.27: Selection of input variables set and their input shape. Max strain: the maximum strain value of all position at each time step. Mean strain: the mean strain value of all position at each time step. All_strain: all of the position (1000 length) that the strain is taken. Mean Strain+geo: the Mean Strain variable plus 8 variables from the geometry features.

Table 5.27 shows the training size of a different set of input variables. Similar to the width prediction in 4.2.1., we can see how the input variables affect the performance. There are the max value of the strain in each time step, the mean value of strain data in each time step, the all position strain value in each time step, and the mean strain with geometry variables. Therefore, there are four sets of input to fit into the linear regression model separately in deflection prediction.

	MSE	MAE	RMSE
Mean Strain	0.0091	0.066	0.095
Max Strain	0.0102	0.072	0.101
All Strain	0.0001	0.006	0.008
Mean Strain+geo	0.0028	0.037	0.053

Table 5.28: Performance of LR model using different set of input variables as predictor

Table 5.28 shows the performance of using the four sets of input variables with a linear regression predicted on test data. The MSE, MAE and RMSE are the evaluation measurements for the performance of the four sets of inputs. Table 5.28, the best performance comes from using all the strain value of 1000 positions at each time step, which achieved the lowest MSE, MAE, and RMSE.

The results from Table 5.28 is similar to the Table 5.16, that using all features of strain data has the best performance. Table 5.28, using the mean and max value of the strain value could not achieve a good performance, meaning that they might lack some information and essential features when only taking the mean and maximum value. However, the Mean Strain+geo set has a good performance for the deflection prediction, though it is still worse than using All Strain.

	MSE	MAE	RMSE
Training	0.0001	0.0054	0.0073
Test	0.0001	0.0063	0.0091
Validation	0.0001	0.0062	0.0089

Table 5.29: Performance of deflection prediction from the optimal LR model

Table 5.29 shows the result from the optimal LR with the All Strain set as training and predicted with the training set, test set and validation set separately. By investigating MSE, the training, test and validation set had the same 0.0001 MSE, so we have to look at MAE and RMSE. From MAE and RMSE, the performance of the test and validation set are very close.

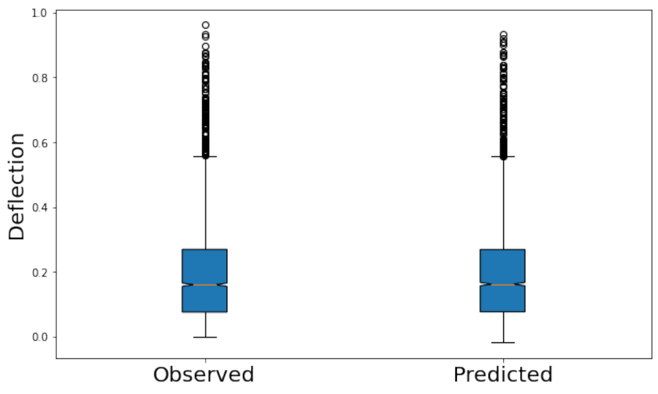

Figure 5.17: Observed and predicted deflection from optimal linear regression model

Figure 5.17 shows a box plot of comparison of observed data and predicted data from the optimal linear regression model. The model is predicted on validation data by using All Strain input variables as training data. The horizontal line in the blue box in the median of the crack width data, the black points on the blue box are the outliners of the data.

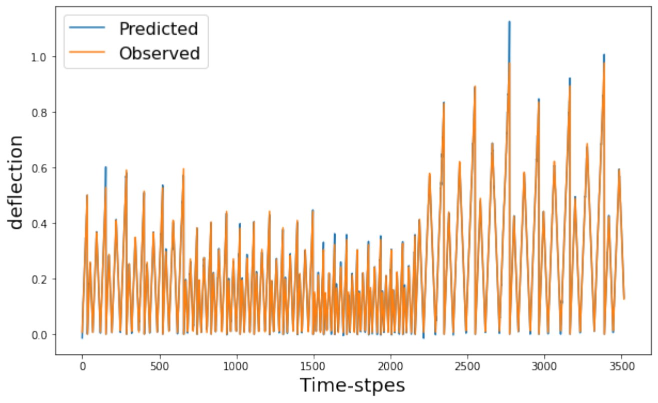

Figure 5.18: Visualization of observed and predicted deflection on validation data using LR model. The blue one the prediction, and the orange one is observations

4.3.1 Deflection Prediction using 1D-CNN

	Training	Test	Validation
Strain data	(10550, 1000)	(3517, 1000)	(3517, 1000)

Table 5.30: Distribution of observation of the strain data obtained

For tuning an optimal 1D-CNN model, a 3-folds cross validation with a grid search method is used for selecting the best hyper-parameters. The candidates of hyper-parameters are shown as Table 5.31, where are the number of dense layers, optimizer, dropout rate.

Number of dense layers	Optimizer	Dropout rate	Learning rate
[0, 1 ,2]	[adam, rmsprop]	[0,0.3,0.5,0.7]	[0.001,0.005,0.05]

Table 5.31: Hyper-parameters for tuning 1D-CNN model

	Number of dense layers	Optimizers	Dropout rate	Learning rate
Best parameters	2	adam	0	0.001

Table 5.32: The optimal hyper-parameters selected from the grid search using 3-folds cross validation

In Table 5.32, the optimal hyper-parameters are selected for the optimal 1D-CNN model by 3-folds cross-validation and grid search method, which takes the lowest error among different sets of hyper-parameters. The model is trained with a training set and selected the optimal one from predicted on the test set. The result of optimal hyper-parameters for the 1D-CNN model on deflection prediction is: 2 dense layers, 0 dropout rate, 0.001 learning rate and 'adam' optimizer.

	MSE	MAE	RMSE
Training	0.0003	0.0132	0.0172
Test	0.0015	0.0262	0.0382
Validation	0.0030	0.0414	0.0549

Table 5.33: Performance of deflection prediction from the optimal 1D-CNN model

Table 5.33 shows the result from the optimal 1D-CNN model with the selected hyper-parameters from using grid search and 3-fold cross-validation method. The training for the 1D-CNN model is predicted on the training set, test set and validation set separately. The best result is on a training set with 0.0003 MSE, 0.0132 MAE and 0.0172, but compared with the validation set where 0.003 MSE, 0.414 MAE and 0.0549 RMSE, the model might be slightly overfitting.

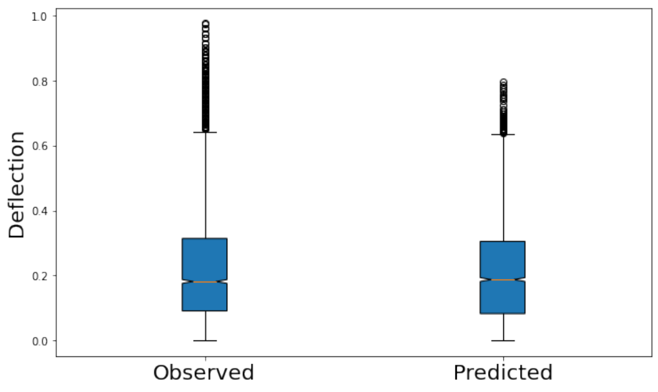

Figure 5.19: Observed and predicted deflection prediction on validation data using optimal 1D-CNN model. The horizontal line in the blue box is the medium of the data, and the black points on the box are the data outliners

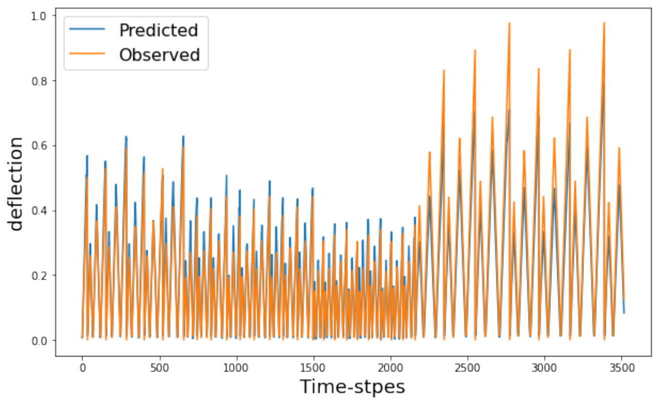

Figure 5.20: Visualization of observed and predicted deflection on validation data using optimal 1D-CNN model

6 Discussion

The main goal of this thesis is to explore deep learning methods to apply on SHM for future usage on damage detection and improve the automation on the prediction of crack position, crack width and deflection without human intervention.

To first select a model for the prediction on crack position and an evaluation method for the results in a fair way is a challenging part due to the imbalanced classes in the data. It is easy to fall into the trap that a classification model predicts all the classes as negative while having high accuracy. In this case, we would like to know where the cracks will fall within the 1000 positions; even though the predicted position is not exactly where the true crack is, a predicted position that next to the true crack position is also what we concern about. Therefore, a model can predict where the position is and return the probability that indicates the possible area that a crack could be there. Given three-dimensional input and output data, the model needs to handle non-linear classification with higher efficiency for processing a large amount of training data samples and learning from the pattern. To this end, a deep learning method is a handy solution for this task.

This thesis introduces an LR model, a 1D-CNN model and an LSTM model for solving the regression problem of crack width and deflection problem. Also, 1D-CNN and LSTM models are introduced to solve the classification problem of crack position prediction. Therefore, there are one classical machine learning method and two deep learning methods for comparison.

From the research on machine learning and deep learning application on SHM, this is the first time that a 1D-CNN model and an LSTM model are implemented for crack position prediction using numeric data obtained from DOFS. The result shows that an LSTM model is a reasonable and promising method for crack position prediction while only using strain data for training the model. Moreover, the LR is also a promising method for crack width and deflection prediction, as well as CNN and LSTM model achieved good performance on the results.

6.1 Results

5.1.1 Prediction of crack positions

A 1D-CNN model and an LSTM model is applied for the prediction of crack position. The result shows that the LSTM model performed better than the 1D-CNN model. The LSTM model does not only provide a higher AUC and recall than 1D-CNN model, it also successfully predict the probability of possible crack areas that related to the true cracks located. However, a 1D-CNN is well-know for dealing with time-series data, the result is not very impressive in this case.

As shown in Figure 5.4 which illustrated the predicted crack positions from tuned 1D-CNN model, it does not distinguish the relationship from the sequences data. It implies that 1D-CNN model could fail to handle high-dimensional classification when it comes to sequences data. The crack position can be difficult to detect when the positive labels are sparse within each time step. At first glance, the cracks are located at the peaks of strain variation, however, even though the strain value is decreased at a small value by the time, the cracks are still located at the same places. Therefore, it will confused the model to determined the positions. The results tell us very roughly about the crack positions, but it might not be a alternative for damage detection in the case.

The main characteristic of LSTM is that the model is capable of memorized the pattern for a long duration of the time. The result at Figure 5.9 shows a clear relationship between the predicted crack position and true positions in most of the cases. The pattern is clearly extracted using LSTMs while feeding the model with sequence data. It is very essential that LSTM can remember the information from the past predictions, so it helps to determine the predicted by looking at where the crack located in the previous time steps.

The AUC is the main evaluation metric for this binary classification problem, since other evaluation methods such as F1-score and precision are not able to represent the fair evaluation of the prediction. The purpose is to investigate the performance of the models on imbalanced data classification, it is more relevant to use AUC and recall to measure the result instead of using accuracy. The F1-score and precision value is very low in both 1D-CNN and LSTM results, it happened because there are very few positive labels of our data. The ROC curve implied the a trade-off between true positives rate and false positive rate. It will be difficult to find the true balance in this case, because sometimes the predicted position might be very close to true one, but it is classified as false positive which the predicted crack is still a useful prediction for the situation.

5.1.2 Prediction of crack width

The crack width prediction can be seen as a simple regression problem that use multivariate time series data as input. In this task, multiple sets of input variables are used to train in a linear regression model in order to explore how the strain and geometry variables affect to the performance. From Table 5.16, it shows the summary of the performance using different set of input variables, where the strain data with 20 features have the best performance than only use the mean of strain plus geometry variables. It is interesting to find out the the geometry settings are not contributed more information to the prediction. Using purely strain data achieved the best result compares with other set of variables.

The comparison of different models such as 1D-CNN and LSTM with LR model, the deep learning models are achieved good results. While the 1D-CNN model achieved the lowest MSE, and LSTM achieved the second best MSE, the LR model has relatively worse performance. The evaluation results show that a multivariate regression problem can be easily solved by deep learning models and achieved better performance.

Figure 5.13 visualized the uncertainty using Monte Carlo dropout with 1D-CNN model to predict the crack width, compared to Figure 5.16, the uncertainty is relatively higher than using LSTM model, which is matched with the result from the MSE. Moreover, Figure 5.12 and Figure 5.15, the mean prediction in LSTM model is closer to the true mean than in the 1D-CNN model. From the Monte Carlo dropout method, it is obvious that using LSTM model for predicting width has lower uncertainty than using 1D-CNN even when they are fitting with same amount of training samples.

For the future usage on the crack width prediction, the LR model and 1D-CNN model is recommended according to the previous chapter. Although the LSTM model also achieve a good result, the model is time-consuming and relatively complicated for processing a simple prediction.

5.1.3 Prediction of deflection

The prediction of deflection is very similar to the crack width as they are belong to regression problem, so the expectation results for the deflection should be close to the crack width. Therefore, only a LR model and a 1D-CNN model is selected to perform the prediction for this task.

The Table 5.27 shows that using strain data with all 1000 features achieved the highest performance compared with other sets of variables. The MSE reached 0.0001 which implied more features for the prediction can achieve a better result; if we compare this result with the crack width performance which only achieved 0.004. The only different between the inputs of deflection and crack width is the length of position features. The strain input for deflection contains 1000 features and the strain input for the crack width only contains with 20.

The comparison of LR and 1D-CNN shows that the LR achieved a better performance than 1D-CNN, while LR is more intuitive model and consumed enormous less time for the training process. Therefore, a LR model is highly recommended in this task when the deflection prediction is able to use all features from the input data.

6.2 Methods

This thesis is mainly focused on the prediction on crack positions, crack width and deflection problem using 1D-CNN, LSTM and LR models, while there are many other ML and DL models that can be applied on this case.

6.2.1 Crack positions

To investigate the performance of the models on imbalanced classification in this case, AUC might not be the only and the best solution for crack position predictions. One of the alternatives on improving the evaluation metric is to consider with slacks on the prediction, where if a predicted crack position that close to the true position could be considered as a successful one. Therefore, adding slacks on the evaluation metrics would get a better performance from the same models. The other option is to find the best classification threshold under the ROC, the study of [31] discussed in details about applying a best threshold algorithm in order to improve the performance on the imbalanced classification problem. Due to the disadvantage of using ROC-AUC values only reflect the ranking power of positive prediction probability, the ROC-AUC does not ensure a high prediction accuracy. In this thesis, the threshold is simply selected by the highest G-mean on the ROC.

In this study, a 1D-CNN is applied for predicting crack positions but it did not have a impressive performance. On the other hand, a temporal convolutional network (TCN) could be another possible solution for predicting with three-dimensional time-series data. TCN is one of the popular deep learning method for solving time-series problem. Another suggestion for this case is gated recurrent unit (GRU) model, it is also an recurrent network that similar to LSTM, but GRU is more simple and consume less time than the traditional LSTM model. Therefore, a TCN and a GRU could be possible solutions for continue working and improving for this study.

6.2.1 Crack width and deflection

For prediction crack width and deflection in this study, the LR model is compared with 1D-CNN model and LSTM model. However, there are other possible solutions for regression problem, such as random forest or XGboost etc. Deep learning method could be unnecessarily complicated in this case. Due to the limited time for this work, a LR model is a best solution compare the deep learning methods.

7 Conclusion

In this thesis, three models are proposed to applied for SHM to detect damage conditions on RC beams. The two deep learning methods such as 1D-CNN and LSTM are applied on predicting crack position, crack width and deflection. The LR model applied on prediction crack width and defection and compared with the two deep learning models. The main advantages of using the proposed models for SHM is the model is capable to provide useful insight on crack positions, crack width and deflection. The models are able to provide a consistently good performance with using time-series numerical data for imbalanced classes prediction. The performance of the deep learning approaches, such as 1D-CNN and LSTM models, provide the possibility of detecting damage automatically as crack position. A LSTM model shows a better performance on ROC-AUC than a 1D-CNN model with predicting with the sequence data. Moreover, a linear regression, 1D-CNN and LSTM also provide a equally-good performance on the crack width and deflection prediction.

Future work

There are some things that can be improved and followed up for the future work. First, the model trained with the simulation data can be tested with the experimental data in the future. If the model performs well on the experimental data, which implies this study would be a big step forward to apply machine learning and deep learning methods on SHM in real-life. Second, the imbalanced classification is a big topic to discuss, other models such as TCN or GRU can be applied on the classification problem; and a detailed study on exploring evaluation metrics on imbalanced classes can be conducted as well in the future.

Bibliography

[1] Jacopo Acquarelli, Twan van Laarhoven, Jan Gerretzen, Thanh N Tran, Lutgarde MC Buydens, and Elena Marchiori. "Convolutional neural networks for vibrational spectroscopic data analysis". In: *Analytica chimica acta* 954 (2017), pp. 22–31.

[2] Melanie Andersson. *Multi-Class Imbalanced Learning for Time Series Problem: An Industrial Case Study*. 2020.

[3] Carlos Avila, Yukikazu Tsuji, and Yoichi Shiraishi. "Crack width prediction of RC structures by Artificial neural networks". In: *Adaptive and Natural Computing Algorithms*. Springer, 2005, pp. 92–95.

[4] António Barrias, Joan R Casas, and Sergi Villalba. "A review of distributed optical fiber sensors for civil engineering applications". In: *Sensors* 16.5 (2016), p. 748.

[5] Carlos G Berrocal, Ignasi Fernandez, Mattia Francesco Bado, Joan R Casas, and Rasmus Rempling. "Assessment and visualization of performance indicators of reinforced concrete beams by distributed optical fibre sensing". In: *Structural Health Monitoring* (2021), p. 1475921720984431.

[6] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006. Chap. 5. Neural Networks, pp. 225–229.

[7] Andre Brault, Neil A Hoult, Tom Greenough, and Ian Trudeau. "Monitoring of beams in an RC building during a load test using distributed sensors". In: *Journal of Performance of Constructed Facilities* 33.1 (2019), p. 04018096.

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

[9] François Chollet et al. *Keras*. `https://keras.io`. 2015.

[10] Armen Der Kiureghian and Ove Ditlevsen. "Aleatory or epistemic? Does it matter?" In: *Structural safety* 31.2 (2009), pp. 105–112.

[11] Sattar Dorafshan, Robert J Thomas, and Marc Maguire. "Comparison of deep convolutional neural networks and edge detectors for image-based crack detection in concrete". In: *Construction and Building Materials* 186 (2018), pp. 1031–1045.

[12] Charles R Farrar and Keith Worden. *Structural health monitoring: a machine learning perspective*. John Wiley & Sons, 2012.

[13] Tom Fawcett. “An introduction to ROC analysis”. In: *Pattern recognition letters* 27.8 (2006), pp. 861–874.

[14] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. *The elements of statistical learning*. Vol. 1. 10. Springer series in statistics New York, 2001. Chap. 3. Linear Regression Models and Least Squares, p. 44.

[15] Yarin Gal and Zoubin Ghahramani. “Dropout as a bayesian approximation: Representing model uncertainty in deep learning”. In: *international conference on machine learning*. PMLR. 2016, pp. 1050–1059.

[16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*. Vol. 1. 2. MIT press Cambridge, 2016. Chap. 10. Sequence Modeling: Recurrent and Recursive Nets.

[17] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[18] Jin Huang and Charles X Ling. “Using AUC and accuracy in evaluating learning algorithms”. In: *IEEE Transactions on knowledge and Data Engineering* 17.3 (2005), pp. 299–310.

[19] Richard Arnold Johnson, Dean W Wichern, et al. *Applied multivariate statistical analysis*. Vol. 5. 8. Prentice hall Upper Saddle River, NJ, 2002. Chap. 7.2 The Classical Linear Regression Model, p. 362.

[20] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Shun Chen. “LSTM fully convolutional networks for time series classification”. In: *IEEE access* 6 (2017), pp. 1662–1669.

[21] Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J Inman. “1D convolutional neural networks and applications: A survey”. In: *Mechanical Systems and Signal Processing* 151 (2021), p. 107398.

[22] Do-Hyung Kwon, Ju-Bong Kim, Ju-Sung Heo, Chan-Myung Kim, and Youn-Hee Han. “Time series classification of cryptocurrency price trend based on a recurrent LSTM neural network”. In: *Journal of Information Processing Systems* 15.3 (2019), pp. 694–706.

[23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[24] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. “Efficient mini-batch training for stochastic optimization”. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 661–670.

[25] Salim Malek, Farid Melgani, and Yakoub Bazi. “One-dimensional convolutional neural networks for spectroscopic signal regression”. In: *Journal of Chemometrics* 32.5 (2018), e2977.

[26] Cláudia Neves. “Structural health monitoring of bridges: Model-free damage detection method using machine learning”. PhD thesis. KTH Royal Institute of Technology, 2017.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[28] Ronald Seoh. “Qualitative Analysis of Monte Carlo Dropout”. In: *arXiv preprint arXiv:2007.01720* (2020).

[29] Panagiotis Seventekidis, Dimitrios Giagopoulos, Alexandros Arailopoulos, and Olga Markogiannaki. “Structural Health Monitoring using deep learning with optimal finite element model generated data”. In: *Mechanical Systems and Signal Processing* 145 (2020), p. 106972.

[30] Renzhuo Wan, Shuping Mei, Jun Wang, Min Liu, and Fan Yang. “Multivariate temporal convolutional network: A deep neural networks approach for multivariate time series forecasting”. In: *Electronics* 8.8 (2019), p. 876.

[31] Quan Zou, Sifa Xie, Ziyu Lin, Meihong Wu, and Ying Ju. “Finding the best classification threshold in imbalanced classification”. In: *Big Data Research* 5 (2016), pp. 2–8.