Summer 2022

# A NOVEL APPROACH TO DAMAGE DETECTION USING STRUCTURAL HEALTH MONITORING DATA AND CONVOLUTIONAL NEURAL NETWORKS

Dominic Emory
*University of New Hampshire*

# A NOVEL APPROACH TO DAMAGE DETECTION USING STRUCTURAL HEALTH MONITORING DATA AND CONVOLUTIONAL NEURAL NETWORKS

BY

DOMINIC EMORY

B. Eng, University of New Hampshire, 2019

THESIS

Submitted to the University of New Hampshire

in Partial Fulfillment of

the requirements for the Degree of

Master of Science

in

Civil Engineering

September 2022

## COMMITTEE MEMBERS

This thesis was examined and approved in partial fulfillment of the requirements for the degree of Master of Science in Civil Engineering by:

Dr. Yashar Eftekhar Azam

Assistant Professor of Civil and Environmental Engineering

University of New Hampshire

Dr. Erin Santini Bell

Department Chair and Professor, Civil and Environmental Engineering

University of New Hampshire

Dr. Fei Han

Assistant Professor of Civil and Environmental Engineering

University of New Hampshire

Dr. Philippe Kalmogo

Assistant Professor of Civil and Environmental Engineering

University of New Hampshire

Original approval signatures are on file with the University of New Hampshire Graduate School.

Table of Contents

Chapter 1. Introduction ........ 1

1. 1 Background and Problem Statement ........ 1

1. 2 Literature Review ........ 6

1. 2. 1 Introduction ........ 6

1. 2. 2 Simulated Experiments for Damage Identification ........ 7

1. 2. 3 Small-Scale Experiments for Damage Identification ........ 10

1. 2. 4 Full-Scale Experiments for Damage Identification ........ 13

1. 2. 5 Experiments with Stationary Loads ........ 15

1. 2. 6 Experiments with Dynamic Loads ........ 16

1. 2. 7 Literature Gaps ........ 17

1. 3 Scope of Research ........ 18

1. 4 Summary of Chapters ........ 20

Chapter 2. Machine Learning ........ 22

2. 1 Introduction ........ 22

2. 2 Overview ........ 22

2. 3 Application in SHM ........ 26

Chapter 3. Field Testing and Data Collection ........ 28

3. 1 Structure ........ 29

3. 2 Instrumentation ........ 30

3. 3 Testing ........ 32

3. 4 Damage Scenarios ........ 34

Chapter 4. Data Processing ........ 37

4. 1 Overview ........ 37

4. 2 Time Series Data ........ 37

4. 3 Representation as Image ........ 42

4. 4 Image Use in CNNs ........ 46

Chapter 5. Methodology ........ 48

5. 1 Introduction ........ 48

5. 2 Model Architecture ........ 54

5. 3 Model Training ........ 60

5. 4 Final Model Training and Validation Accuracy ........ 64

5. 5 Conclusion ........ 67

Chapter 6. Results and Discussion ........ 68

6. 1 Resulting Filters ........ 68

6. 2 Testing Accuracy ........ 70

Chapter 7. Conclusion ........ 76

7. 1 Summary of Work ........ 76

7. 2 Future Work ........ 77

Appendix A: Data Conversion MATLAB Script ........ 87

## LIST OF FIGURES

Figure 1: Frick Hallow Bridge Collapse ........ 2

Figure 2: Lane Place Bridge Collapse ........ 3

Figure 3: Full Height Crack on Bottom Chord of Hernando DeSoto Bridge ........ 4

Figure 4: Bridge Deck mock-up (Akintunde et al. 2021) ........ 28

Figure 5: Mock-up and Vehicle (Akintunde et al. 2021) ........ 28

Figure 6: Mock-up section view (Akintunde et al. 2021) ........ 29

Figure 7: BDI Strain Gage ........ 30

Figure 8: Overall Gage Layout (Akintunde et al. 2021) ........ 31

Figure 9: Full Dump Truck (Akintunde et al. 2021) ........ 32

Figure 10: Pickup Truck (Akintunde et al. 2021) ........ 33

Figure 11: Crash Induced Damage (D1) (Akintunde et al. 2021) ........ 34

Figure 12: Outside Face of Barrier after Impact (Akintunde et al. 2021) ........ 35

Figure 13: Inside Face of Barrier after Impact (Akintunde et al. 2021) ........ 35

Figure 14: Inside (left) and Outside (right) Face of Barrier after Concrete Barrier (D3) ........ 36

Figure 15: Deck Damage (D3) (Akintunde et al. 2021) ........ 36

Figure 17: Strain Time histories for Damage 1 Bridge under Full Dump Truck Loading ........ 38

Figure 16: Strain Time histories for Undamaged Bridge under Full Dump Truck Loading ........ 38

Figure 19: Strain Time histories for Damage 1 Bridge under Pickup Truck Loading ........ 40

Figure 18: Strain Time histories for Undamaged Bridge under Pickup Truck Loading ........ 40

Figure 20: Strain-Time histories represented as 3D Surface Plot ........ 41

Figure 21: Examples of Strain Time Histories represented as 2D greyscale images ........ 44

Figure 22: 3D Plot of Original Data ........................................................................ 45
Figure 23: 3D Plot of Greyscale Image representing Data ........................................................ 45
Figure 24: Fully Connected Neural Network Diagram (James et al., 2021) ............................... 48
Figure 25: ReLu and Sigmoid Activation Functions (James et al., 2021) .................................. 50
Figure 26: CNN Examples (James et al., 2021) ........................................................................ 53
Figure 27: Overall Final Network Architecture ......................................................................... 58
Figure 28: Detailed Final Network Architecture ........................................................................ 59
Figure 29: Cost function Example (James et al., 2021) ............................................................. 61
Figure 30: Training Progress Plot for Model with 70% Training Data ....................................... 65
Figure 31: Training Progress Plot for Model with 30% Training Data ....................................... 66
Figure 32: Original Example Image before convolutional layer ................................................ 69
Figure 33: Activations of Image after first convolutional Layer ................................................ 69
Figure 34: 70% Training Data Confusion matrix ....................................................................... 70
Figure 35: 30% Training Data Confusion matrix ....................................................................... 71
Figure 37: Strain Gage Data with 15% added Noise .................................................................. 74
Figure 36: Strain Gage Data without added Noise ..................................................................... 74
Figure 39: Reduced Sensor Confusion Matrix ........................................................................... 75
Figure 38: 15% Added Noise Confusion Matrix ........................................................................ 75

## LIST OF TABLES

Table 1: Summary of Tests ........................................................................................ 33
Table 2: Training, Validation and Testing Data split for Models ................................................ 46
Table 3: Summary of CNN Architectures .................................................................................. 56
Table 4: Summary of Hyperparameters ...................................................................................... 63

## ABSTRACT

A NOVEL APPROACH TO DAMAGE DETECTION USING STRUCTURAL HEALTH MONITORING DATA AND CONVOLUTIONAL NEURAL NETWORKS

By

Dominic Emory

University of New Hampshire

This study investigates the efficiency of Convolutional Neural Networks (CNNs) to detect structural damage from measured structural response. In this regard, strain gages were used to measure structural response from live loads. Data used during experimentation was gathered through the testing of a full-scale concrete bridge mockup. The test captured the response of the mock-up bridge utilizing 24 transversely oriented resistance-based strain gages under similar loading conditions, but with different levels of damage induced. There are three levels of damage the bridge was subjected to; crash-induced damage to the barrier, transverse cut on the entire barrier, and transverse cut along the deck. Live load testing was done for undamaged and damaged conditions by running vehicles at various speeds over the deck and recording the response overall strain gages. Previous studies have compared the effectiveness of Singular Value Decomposition (SVD) and Independent Component Analysis (ICA) to apply a novelty index to the same data to detect damage.

This study seeks to expand on this investigation by utilizing the data as a full snapshot matrix converted into a 2D greyscale image to classify frames as Damaged or Undamaged through a Supervised approach. The supervised CNN uses two convolutional layers with dropout and a fully connected output layer and reached an accuracy of 95% with only 30% of the total dataset used as training data. This study shows that CNNs provide a robust way of detecting damage from full data snapshots represented via greyscale images.

# Chapter 1. Introduction

## 1. 1 Background and Problem Statement

The maintenance and management of infrastructure is a critical part of maintaining the quality and normality of life in the United States. In 2020 alone the US Bureau of Transportation estimated that over three trillion miles were traveled by commercial and noncommercial vehicles (FHWA, 2020). This transportation of people and goods is heavily facilitated by highway systems and bridges which are a critical part of the transportation infrastructure and are vital in promoting the growth of the American economy. Since the collapse of the Silver Bridge in 1967 due to poor maintenance and peak loading, which resulted in the unfortunate death of forty-six people, the US government has made a concerted effort into the management of bridges and roads. Following the collapse, the Federal Highway Administration enacted the National Bridge Inspection Standards, which required the inspection of bridges and their components at least once every two years by inspectors who were appropriately trained.

Since this enactment, the maintenance and performance assessment of infrastructure in the US has been predominantly centered around visual inspection of critical elements of a given structure by trained engineers. The process typically requires extensive training of individuals and sometimes the extended closer of highly trafficked bridges. This has been integral for better maintaining infrastructure in the US; however, it is still subject to inherent human bias. This bias can come in the form of difference in visual inspection techniques or standards of what constitutes the need for either further inspection or repair. While inspection standards are constantly improving this bias can still sometimes lead to the failure of critical structural components, which in some instances can lead to a full bridge collapse.

Despite stricter visual inspection and bridge maintenance regulations, the Federal Highway administration still estimates over 50% of bridges are in fair or poor condition (FHWA, n.d.). One of the most recent large scale bridge failures took place in January of 2022. The Fern Hollow Bridge was a four lane, 400ft long bridge that carried Forbes Avenue over Frick Park in Pittsburgh, PA and recently collapsed. It is not yet known what the cause of failure was, however, the deck and the superstructure both scored a 4 on the PennDOT Bridge Safety Inspection condition rating and it was last inspected in the Fall of 2021 (Krauss, 2022). Even though the bridge was compliant with current standards and did not score low enough to be deemed structurally deficient it still collapsed, an image after failure can be seen in Figure 1. This bridge collapse is not a rare

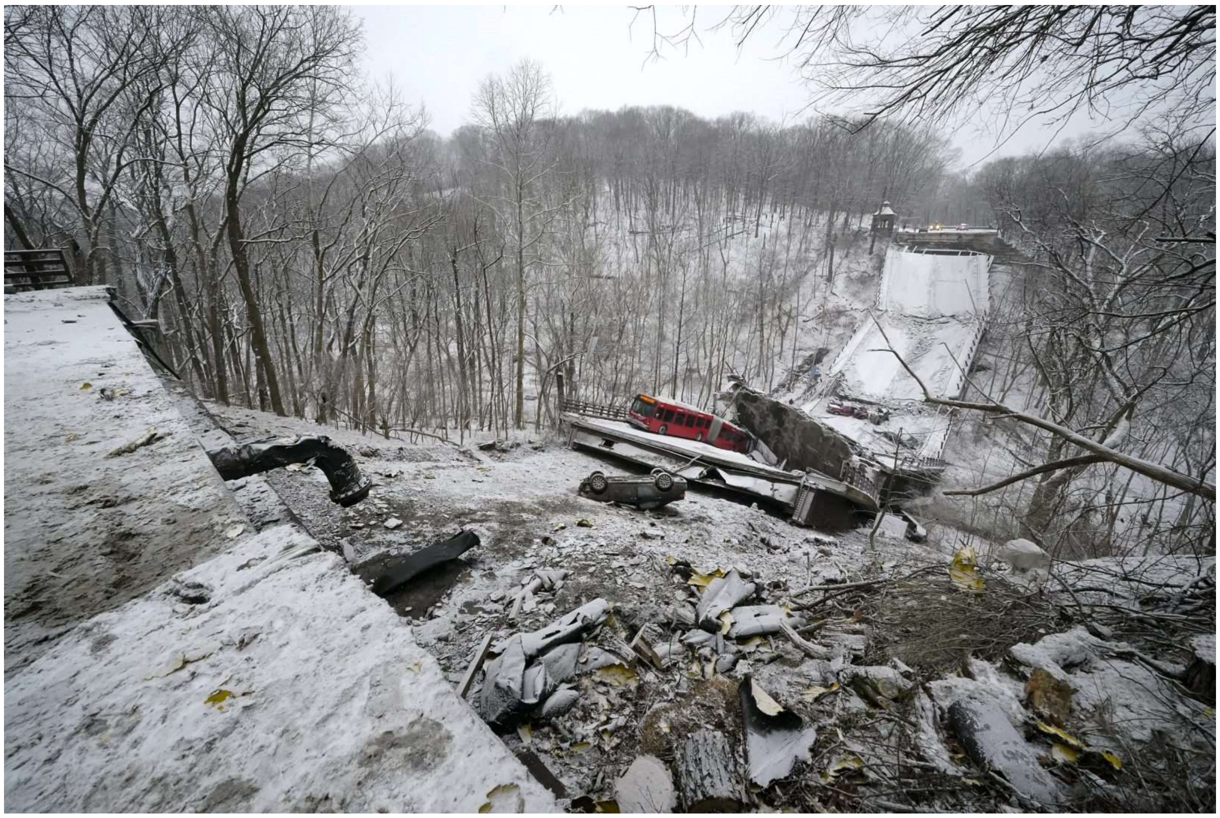


Figure 1: Frick Hallow Bridge Collapse

occurrence in the US today. In Washington DC the Lane Place bridge, which provided pedestrian traffic over Route 295 collapsed in June of 2021, the resulting damage can be seen in Figure 2.

The collapse completely shut down the six-lane highway and repairs are estimated to cost a total of $25 million (Lazo, 2022). Another bridge failure occurred in May of 2021, where a full

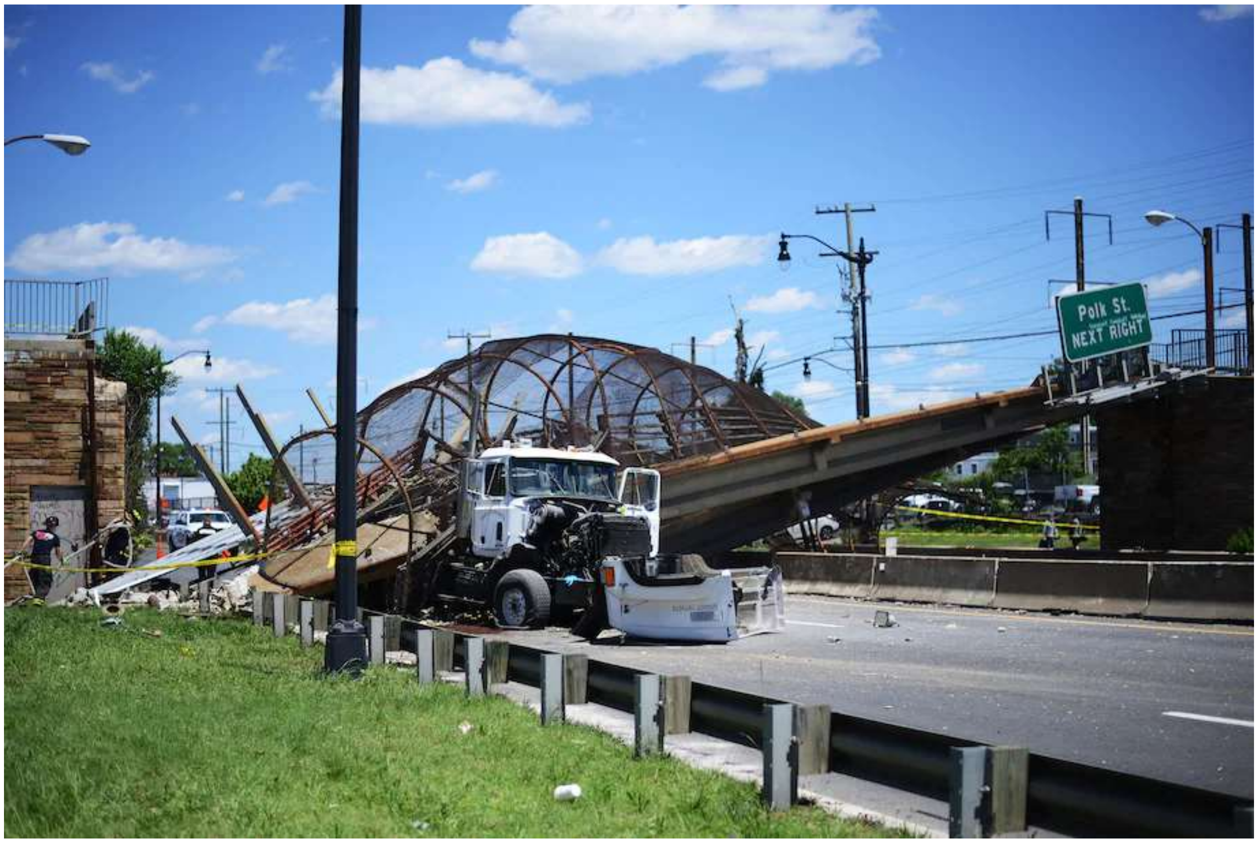


Figure 2: Lane Place Bridge Collapse

height crack was found on the bottom chord of the Hernando DeSoto Bridge, which carries Interstate 40 over the Mississippi River (Figure 3). The bridge, which carries about 60,000 cars a day, was forced to close for 3 months for repairs after the crack was found during routine inspection causing a major disruption in traffic patterns (TNDOT, 2021). While this bridge was luckily shutdown before any further damage to the structure occurred, it can still be seen as a major mishap in its maintenance. These are just a few examples of how the maintenance and upkeep of infrastructure in the US is falling short and can be improved. While regulations have been improving these types of failure are still seen throughout the United States.

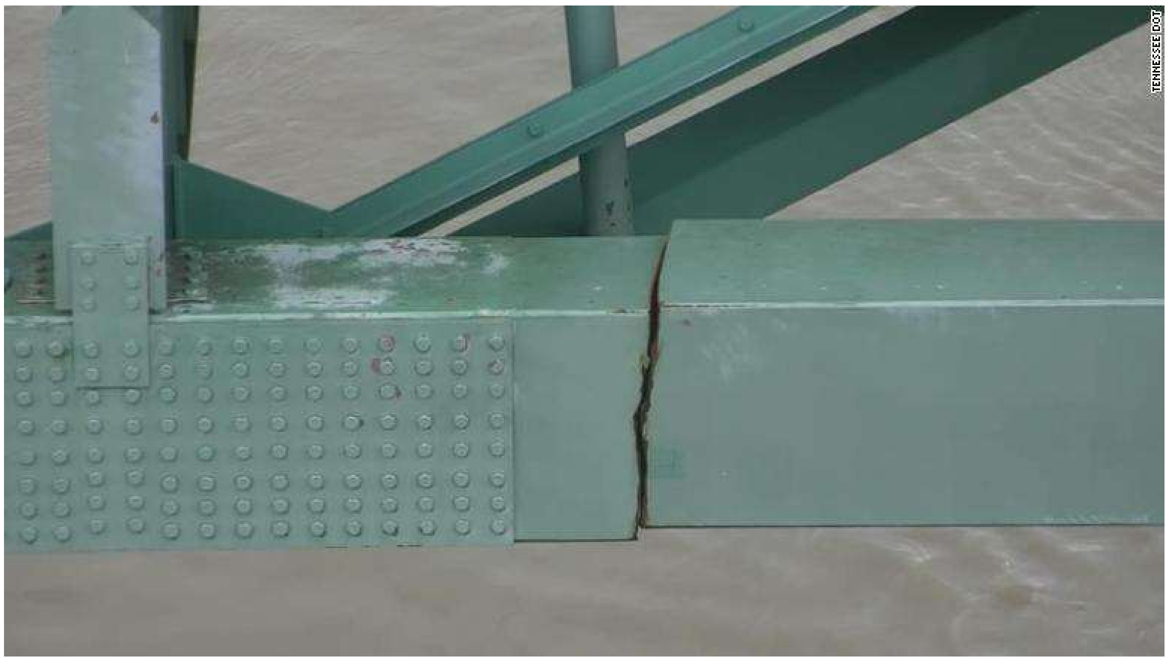


Figure 3: Full Height Crack on Bottom Chord of Hernando DeSoto Bridge

As of 2021, ASCE lowered the overall grade of bridges on its infrastructure report card from a C+ in 2017 to a C. This decrease in condition can be attributed to the growing average age of bridges and the slowdown in rate of reduction of structurally deficient bridges throughout the country (ASCE, n.d.). It is estimated that it would take 50 years and $125 billion dollars for all rehabilitation measures needed for bridges across the US. While increased inspection and stricter regulations have significantly improved the asset management practices within the US there are still significant structural failures and challenges in maintenance decisions. For this reason, engineers have begun to look towards different methods in the evaluation of bridge condition to make more informed decisions. The prospect of supplementing visual inspection with computational aid, mainly from the applications of techniques such as model-oriented and data-oriented structural analysis, has become a significant field of research in Civil Engineering.

The potential to reduce inherent human bias and gain more knowledge on the behavior of a structure has driven the field of Structural Health Monitoring (SHM) for many years. SHM is the analysis of a structural system using several different measurements including, but not limited to, strain, acceleration, temperature, and displacement. What was once a niche field has become a fairly common research subject in Civil Engineering today. This could be attributed to the advancements in SHM making systems much more dependable and readily available for installation. SHM allows engineers to gain insight into a structure remotely and without needing to rely on inspection reports. Time-series data gathered through SHM is used for monitoring the behavior of a bridge, calibrating finite element models or detecting damage in a structure that would otherwise go unnoticed. While SHM offers information about a structure it also presents a challenge of efficiently and accurately analyzing gathered data. Traditional techniques of analyzing large datasets by an individual are time consuming and can still be prone to bias. In this regard the use of machine learning (ML) techniques on gathered data is a promising transition from traditional methods of visual inspection and analysis of data to a more efficient statistics-based framework. In this research a data driven approach for detecting damage on a concrete structure using strain gage measurements is presented.

## 1. 2 Literature Review

### 1. 2. 1 Introduction

Machine Learning (ML) and Deep Learning (DL) techniques have been gaining attention in the civil engineering research community in recent years due to the advancements in SHM systems allowing for the gathering and transferring of large amounts of data remotely. This advancement, while providing a vast amount of raw data also presents the challenge of effectively analyzing it to extract useful information. For this reason, ML is a possible solution due to its ability to analyze massive amounts of data, referred to as Big Data, quickly and efficiently. There have been various methods proposed that have the potential to effectively analyze large amounts of data and supplement typical visual inspection methods. For example, researchers have found that using Singular Value Decomposition (SVD) on raw strain gage data can identify possible changes in a structures behavior that would normally go unseen during routine inspection. (Akintunde et al., 2021)

Structural analysis that uses sensor data can typically be broken up into physics-based models and data-driven models. Physics-based models rely on the calibration of Finite element models based on measured data to analyze a structure. Whereas data driven models use only data gathered in order to analyze the behavior of a structure. The construction of physics-based models has seen a decrease in attention due to the complexity of model building and inability for models to be easily updated based on changing field measurements. On the other hand, with significant development of sensors, sensor networks and computing systems, data-driven machine and mechanical system health monitoring models have become a more viable and popular approach (Azimi et al., 2020).

This research focuses primarily on the development of data-driven damage detection techniques using ML methods. Within data-driven models, sensor data to be used as inputs into ML algorithms is either gathered through a finite element model or from real world SHM systems. Both methods are an appropriate approach to developing damage detection techniques. As such it is important to understand the advancements that have been made in both approaches and how it relates to this research.

### 1. 2. 2 Simulated Experiments for Damage Identification

The acquisition of real-world data in research can be impractical in many cases. For this reason, the use of a Finite Element (FE) model to simulate experimental data and damage scenarios is a common practice. Using FE models allows for researchers to efficiently generate different types and amounts of data and is widely accepted as a valid first approach to test different data driven models via imposition of simplifying assumptions. While this is a valid method, the simulated sensor data gathered can sometimes not account for model uncertainties or variability in loading conditions.  The following section outlines recent studies that have made progress in using simulated data along with ML algorithm to detect structural damage.

Gulgec et al. (2017) researched structural damage prediction using Convolutional Neural Networks (CNNs). The major challenge addressed in this case is the relation between measured data and damage pattern. The structure studied was an analytic model of a steel gusset plate connection, modeled using ABAQUS, in the form of shell elements. From the model, simulated strain measurements for both undamaged and damaged cases were generated for training, validation, and testing data. Uncertainties in the simulated measurements were accounted for by adding white noise, as well different loading and damaged scenarios were applied to the dataset. In this fashion, a dataset was created for testing of the CNN. Strain data, represented as tensors, is fed into the network, along with various levels of noise. The authors normalized the data by subtracting the data mean and dividing by the maximum reading. Additionally, they developed some ways to filter out "less-desired" CNNs by tuning hyperparameters of the CNN network. The best CNN model consisted of two convolution layers, each one with a subsequent pooling layer. The final CNN was able to detect damage on measured data with 2% added noise with a testing error of only 3.51%. These results show how CNNs can provide a robust damage detection framework even with noisy data and with a relatively shallow and straight forward network architecture.

Hakim et al. (2015) also studied the effectiveness of neural networks by developing a strategy for using artificial neural networks (ANNs) to predict the severity and location of damage on a simulated I-beam structure. FE model analysis was used to find natural frequencies and mode shapes for the I-beams to be used as inputs into the ANN. Data and features for the structure was found for both damaged and undamaged states. The result was an ensemble neural network that was able to accurately detect severity of damage on a beam using mode shapes and natural frequencies. The data gathered for this study not only required the creation of a finite element model which increase the time needed for data collection, but also the further analysis of developing mode shapes and natural frequencies for a structure. The approach does however account for some simulated measurement uncertainties by including white noise.

Zhong et al. (2020) also developed a neural network that utilized modal parameters. A CNN damage detection framework based on the first-order modal parameters of a steel truss was proposed. The aim of the study was to locate damaged rods in a complex steel truss. Data was obtained from an FE model without added noise for model uncertainties. Damage scenarios were modeled by simulating damage at individual members in the truss system through changes in member section properties. The final CNN was able to predict damaged members with 100% accuracy in some cases of induced damage. This method shows a way of implementing CNNs based on extracted features from the response of the bridge, but similarly to the previous study requires the extraction of features from gathered data in order to implement the method.

Rastin et al. (2021) conducted a study exploring an unsupervised way to detect damage based on Deep Convolutional Autoencoder. This method used accelerometer data and CNNs via Auto Encoders (CAEs). In ML jargon, Unsupervised damage detection refers to the case that the network is only trained considering undamaged data. The encoder part of the CNN takes in the undamaged input and compresses it to a lower dimensionality then the decoder takes the compressed data and makes its best representation of input data. In this case, damage is determined using the differences in reconstruction data, or reconstruction loss. Since the CAE is only trained on undamaged cases it will have an elevated reconstruction loss when the input is damaged data. Simulated dataset from a FE was used, the model was first run with healthy condition and then several different damaged conditions. It is important to note that the addition of white-noise was not added to simulated data to account for measurement uncertainties. The output after training is a percentage of damage based on the different scenarios from FEA model. The approach was successful at detecting damage in two different cases and presents a possible approach to unsupervised learning using Artificial Neural Networks.

Teng et al. (2019) used a CNN to classify location and level of damage in a structure using the dynamic responses of a structure along with the modal parameter as inputs into the CNN. This approach of using different parameters as inputs into the same network was done in order to increase the robustness of the model since damage can present through changes in multiple different structural behavior parameters. Simulated data with added gaussian noise from a full-scale truss bridge FE model was used with damage being represented by the reducing the Young's modulus in different elements. The resulting CNN was able to detect the level and location of damage with up to 99% accuracy using both modal strain energy and acceleration. The computational costs for this method, however, are noted by the authors to be quite high and required a lot of data for implementation.

### 1. 2. 3 Small-Scale Experiments for Damage Identification

When available, the use of sensor data gathered from live load testing of a structure can be preferable to simulated data. This is because when collecting data on an in service or mock structure the uncertainties and noise of gage measurements is accounted for, which can be overlooked in methods developed using simulated experiments. On the other hand, variable live loading can be easily gathered by using vehicles of different known weights to gather data. While sensor data from an in-service structure is preferable it also proposes a challenge of finding damaged data scenarios for structure. It is very uncommon to have access to damaged data for an in-service structure due to safety and maintenance concerns. For this reason, studies will sometimes use a mock-structure to gather data for different damage scenarios. This section outlines research that has developed methods around small-scale mock structures.

Khodabandehlou et al. (2019) presented a study with vibration-based condition assessment using CNNs. Although 1D CNNs are a common method for time-history data, the authors emphasized that concatenating the data coming from different sensors and then treating them as a single measurement could benefit the performance of the CNN since it increases the dimension significantly. In this study, a 2-D CNN was implemented that used accelerometer readings sampled at a 100 Hz. The accelerometer data was represented as an image before inputting into the CNN as to increase the efficiency of data processing and CNN categorization. Vibration response data was gathered on a one fourth scale model of a reinforced concrete highway bridge subjected to shake table testing. The final method used a 2D CNN that could predict predefined damage states with 100% accuracy using acceleration data. In this experiment the only required data was measured structural vibration response and corresponding damage conditions for training. This is an important distinction as it proposes a method for damage detection that does not require extensive data processing or feature extraction to work effectively. The data processing techniques used by Khodabandehlou et al. are similar to the ones proposed in this research, in that they both represent gage data as images before use in CNNs.

Pathirage et al. (2018) proposed an autoencoder based framework for structural damage identification, which would be able to work with a neural network structure. The autoencoder proposed unlike CNNs is a specific 1D architecture that is sometimes used for different deep learning methods, that does not have a convolving filter. Instead, it utilizes fully connected layers to first reduce the dimension of the input and then learn features in order to detect damage utilizing frequencies and modal parameters from acceleration measurements. The study implemented the procedure on both simulated data from a FE model and a small-scale shear-type steel frame structure. The steel frame structure was built to model the response of an eight-story shear building, and excited with external dynamic force implemented using a modal hammer. The acceleration at each floor was measured throughout experimentation. The uncertainties in measurement of an in-service structure were accounted for by adding noise to the original acceleration measurements. Damage cases for training were modeled in the structure by reducing the column cross-section of specific floors by a certain percentage. The results of the network show a successful identification of magnitude of stiffness loss due to reduction in stiffness of column with an Mean Squared Error (MSE) of 9.1e-6. While this method is effective the computational costs needed to perform it are higher when compared to others. The network structure required layer wise pretraining to optimize the weights of layers as well as whole network fine tuning using a joint optimization towards the final objective function. In addition to this the method used two different features extracted from acceleration data in order to be used as inputs to the network, which increasing the amount of data processing required

Teng et al. (2020) presented an approach to damage detection using simulated accelerometer data to train a CNN and validating the results using vibration experiments from mock structure. In doing so the authors sought to study the effectiveness of training a CNN using simulated data and implementing it on real world data. Training data for the CNN was gathered through simulated experiments using an FE model. The data was modeled in a way that showed damage at various locations along the structure. With all measurements of acceleration being stored into one two dimensions (2D) matrix that was fed into the CNN to determine the damage state. The CNN had a categorical output layer that classified the damage type into one of seven different damage scenarios. After training the network on simulated data it was tested on real data and had a detection accuracy of 90%.  While there are higher accuracies reported in recent articles this research addresses the problem of finding damaged datasets for an in-service structure. By training on simulated data and validating on real data this research presented a way of possibly implementing ML damage detection techniques in industry.

### 1. 2. 4 Full-Scale Experiments for Damage Identification

The use of full-scale experiments for data gathering is preferable to small scale experiments in that it better represents the response of an in-service structure. Full-scale experiments can be impractical and costly in many cases due the resources required for building and testing. For this reason, not many studies have access to these datasets. The following section outlines studies that have been able to develop datasets and damage detection methods using full scale experiments.

Abdeljaber et al. (2018) presented a novel, fast, and accurate structural damage detection framework using accelerometer data from a steel frame structure. All data gathered was from a full-scale mock structure excited under a 0-512Hz band-limited white noise shaker. Damaged and undamaged data was gathered for each joint in the structure, damaged data was simulated by loosening the connection at a joint. With collected data for each joint under both damaged and undamaged conditions a 1-Dimensional (1D) convolutional neural network (CNN) was trained for each individual joint to detect if there was damage present. After training of CNNs damage in joints was able to be detected using a probability of damage (PoD) indicator with the average training data classification error across all CNNs only being 0.54%. This research is of particular interest due to its success in using raw acceleration data. In some applications feature extraction is done on acceleration data before implementation in (ML). By not doing this it proved that heavy data processing is not necessary to detect damage from a structure.

Figueiredo et al. (2019) developed an ML algorithm for detecting damage that utilized both simulated and in-service bridge condition measurements. The algorithm was based on a Gaussian mixture model that was used to estimate the main clusters forming around normal structural condition. The damage detection was based on an outlier formation regarding the chosen cluster. This study used in situ data from an existing structure for undamaged data and simulated data from a calibrated Finite element model for damaged conditions. In this way an FE model was tuned using in-situ data so that measurements from the model matched measurements recorded by the monitoring system. Once calibrated simulated data for undamaged and damaged conditions was generated from the model. The result is an example of how FE data can be used in conjunction with real world data to increase the effectiveness and accuracy of a ML algorithm with damage detection. While this study does require the tuning of an FE model, which can be slow and labor intensive is shows an application of ML for industry. The significance being that in-service bridges that have a SHM system typically do not have measurements for a damaged structural condition. However, by using the existing data to tune an FE model before gathering simulated damaged data a much more accurate dataset was found. The final results address the issue of supervised ML needing damaged datasets, which are usually not readily available.

Feng et al. (2019) proposed a damage detection method using deep convolutional neural network and transfer learning. Unlike previous studies mentioned this method utilizes true image data of concrete surfaces for hydro-junction infrastructure. The images were gathered from an in-service structure and the different conditions and failure modes of the concrete were found. Data labeled under different categories of damage including undamaged, cracking, and spalling were used for training, validation, and testing of the CNN.  Using transfer learning the backpropagation and training sequence for the CNN took much less computational power and time. The result was a

CNN that could categorize 5 different concrete conditions with up to 95% accuracy. While this is a different approach that is taken in this research it is still important to note the different ways that ML is being used to detect damage in infrastructure.

### 1. 2. 5 Experiments with Stationary Loads

Recent studies can be further characterized by the type of loading introduced to the structure in order to gather data. In many cases a stationary load is introduced as to gather responses of either simulated data or SHM data. Stationary loading is a valid way of gathering data but does not account for dynamic or variational loading that is seen on in service structures. This is not a preferred way of data acquisition as the methods are not trained to predict or classify with variational loading making them a less viable option for application in industry.

Gulgec et al. (2017) trained a CNN in order to identify damage in a plate element modeled in ABAQUS. The data for the model was gather by subjecting the plate to constant axial loads, this allowed for easy data collection, however constant axial load on an in-service structure is not common. Hakim et al (2015) also used simulated data from an FE model, but for an I beam element. To gather data a single external excitation force was applied using a shaker. It is common for a beam to see an external excitation force, but again it is not likely to see it consistently in a single location. Teng et al (2019) modeled a truss structure and used an external force again only applied at one point along the length of the structure. Abdeljaber et al. (2018) trained a CNN based off of a full-scale model excited by a shaker. Similarly, to Hakim et al. (2015) through the shaker force was only applied to a single point throughout all experimentation. These methods all resulted in a relatively high accuracy for damage detection and have developed decent approached for ML algorithm building. However, by not considering a variational or dynamic loading effect the realistic responses of an in-service structure can be missed in the training of ML algorithms.

### 1. 2. 6 Experiments with Dynamic Loads

As opposed to stationary loading ML algorithms that use data gathered under a non-stationary or dynamic loading effects are a better representation of how these methods could be implemented in industry. The following methods apply some variation in loading conditions for data collection to be used as inputs into ML algorithms

Rastin et al. (2021) developed a damage detection framework using accelerometer data gathered from an FE model. The FE model was a multi story building which was loaded at multiple different floors in order to gather acceleration data at each floor. While the loading was not completely dynamic it did introduce some variation between applied load between data collection by changing the floors where load was applied at. Pathirage et al. (2018) used what is known as a modal hammer to introduce a dynamic load at various floors on an 8 story small scale structure. Teng et al. (2020) also used modal hammer to excite a steel truss structure, however only excited at one point. The use of a modal hammer is common but does add some variation to loading. However, the methods for finding modal properties of a structure using a modal hammer are not commonly used on in service structures making it hard to implement these methods.

Overall, not many studies consider large variation in loading when creating datasets for testing of ML algorithms. This is mainly due to limitations in resources, however in not doing so the efficacy of methods to be implemented in practice comes into question. In order to further the possibility of real-world implementation variational loading that not only considers difference in magnitude but also difference in location of loading must be considered. To address this the following study uses a dataset that is gathered through the testing of a structure that is subjected to moving vehicles loads that vary in size, weight, and speed.

### 1. 2. 7 Literature Gaps

The main goal in any study is to further the advancement of the field of research and to develop techniques that can be deployed in industry. While the mentioned studies have addressed many issues in the development of ML algorithms for damage detection there are still some gaps in methods. One main note is the lack of full-scale experiments for gathering of data. This is commonly seen throughout research as the resources needed to run a full-scale experiment are not usually readily available. However, by not using data collected during full scale testing the methods developed are not representative of an in-service structure. Another way that in-service loads are not represented in literature is the lack of moving loads for bridge excitation during data collection. In these two ways the measured responses used in the development of ML algorithms can sometimes not be an accurate depiction of real-world conditions. The level of damage detection also varies between studies, particularly the ability to detect invisible damage is not commonly seen. Invisible damage cannot be detected using traditional methods of visual inspection. Without being able to detect changes or damage in a structure that are not visually identifiable ML algorithms lose some utility, in that they are no better at detecting damage then visual inspection. The data processing needed for some methods can also be a hinderance in the usefulness of methods as well. Some approaches require multiple features to be extracted from a dataset before they can be used in an algorithm. By doing so the amount of time and resources needed to analyze data can be greatly increased

This study seeks to further this field of research by addressing some of these issues. In doing so a full-scale mockup bridge that is subjected to various moving loads is used. As well the structure is subject to different damage conditions that range from invisible damage to visible damage. The final data that is used with the algorithm also requires minimal processing, as raw data represented as a 2D greyscale image is used as an input. By using this dataset, a robust CNN detection framework is developed that is able to detect invisible damage in a full-scale structure that is subject to various magnitudes of moving loads.

## 1. 3 Scope of Research

The research presented in the following paper is a continuation of research done by (Akintunde et al., 2021)) and utilizes the same data. Akintunde et al (2021) developed a damage detection framework using unsupervised ML techniques. Strain time history data was gathered by running live load tests on a full-scale bridge deck mockup using varying loading and damage conditions. Using Left Singular Vectors (LSVs) of measured response, commonly known as Proper Orthogonal Modes (POMs), novelty indices were developed for different damage conditions. A novelty index that varies from novelty index of undamaged conditions being an indication of damage. It was found in this study that POMs have the ability to detect the lowest amount of damage caused by vehicle impact which was not clearly visible from a visual inspection. The resulting detection framework was able to identify damage to the bridge under varying weight and speed of loading condition. Additionally, the robustness of the approach was further tested by down sampling data to a sampling rate of 10Hz and only using 4 sensors.

This study investigates the effectiveness of Convolutional Neural Networks (CNNs) to detect damages in the same structure using the same data that was gathered. The approach is a data driven damage detection technique that uses strain gage measurements to categorize the response of a mock-up concrete bridge structure into one of four damage conditions. The resulting method presents a novel approach in which raw data is converted into images and used in conjunction with a CNN to detect damage levels in a structure. In this way the amount of time spent on data processing would be reduced and the behavior analyzed by one optimized convolutional neural network that is robust enough to detect damage under various types of loading.

Data used throughout experimentation was gathered through the testing of a full-scale bridge mock-up. The response of a mock-up concrete bridge, under different loading and damage conditions was captured using 24 transversely oriented gages. There are four levels of damage the bridge was subjected to; undamaged, crash-induced damage to the barrier, transverse cut on the entire barrier, and transverse cut along the deck. Live load testing was done for undamaged and damaged conditions by running vehicles at various speeds over the deck and recording the strain time histories across all gages. The recorded strain time-histories were stored in a snapshot matrix that was normalized and converted into a 2D greyscale image in order to be used in a CNN. The resulting images capture the interactions between gages throughout the structure and captures the overall behavior of the structure. Images are then used for the training, validation, and testing in two different CNN models. Data processing was done using MATLAB.

To test the effectiveness of CNNs on damage detection two different models were tested. The only parameter that changed between the models was the amount of training data used. With one model using 30% data for training and one using 70% data for training. This was done to test if this approach could be implemented with only a limited amount of training data available. The architecture for the CNN was developed using an intuitive approach based off the number of gages and gage layout. Hyperparameters were tuned to optimize the training and validation accuracy and computational efficiency during training. The MATLAB Deep Learning Toolbox and Experiment Manager was used to train and optimize architecture and parameters of the CNNs. As well results and graphs used in this research were compiled using MATLAB.

Based on results CNNs present a damage detection framework using strain time histories from a concrete bridge structure. In this regard this work continues the research presented by Akintunde et al. in expanding on (ML) techniques to be used on the data collected during experimentation. This research contributes by proposing a way to represent data as an image and by detecting damage efficiently by use of a convolutional neural network that is optimized to use reduced amounts of training data.

## 1. 4 Summary of Chapters

Chapter 1 provides background into the current state of infrastructure in the US and how ML algorithms along with SHM systems can be useful in the maintenance and asset management of structure. As well a state-of-the-art literature review and lit gap is presented followed by the contributions and scope of this research. Chapter 2 provides a brief introduction into the field of Machine Learning in order to give context to following discussions. It also gives insight into the development and use of it in SHM systems. Chapter 3 outlines the testing procedure that was used to gather a robust dataset to be used in ML algorithm development. Chapter 4 gives an overview of data in terms of loading conditions and damage scenarios and what the original data looked like. The methods of accurately representing data as a 2D greyscale image are also presented in this section. Chapter 5 gives a brief introduction into how CNNs work as to give context to the following methods of determining the architecture and hyperparameters to be used in CNN. It also outlines the performance of the CNN on training and validation data. Chapter 6 discusses the resulting trained CNNs and the final testing accuracy of both. The differences of the model that used 70% training data and the one that used 30% training data is also discussed. Chapter 7 summarizes work completed and gives a brief overview of possible continuation of research.

# Chapter 2. Machine Learning

## 2. 1 Introduction

Machine learning (ML) is a subset of artificial intelligence (AI) that refers to a vast set of tools used for better understanding trends and features in datasets. ML on one hand has recently found popularity due to advances in technologies allowing for the gathering and storing of large sets of data. On the other hand, improvements in computing machines have made executing complex algorithms relatively quick. These advancements were necessary for ML techniques since they require a relatively large amount of data to train the algorithms for either regression of categorical models. The basic idea is that a ML algorithm is trained to classify or predict outputs of a dataset to aid in decision making. In the field of Civil Engineering the ability to use ML was mostly hindered by the technology surrounding instrumentation and data acquisition of SHM systems. With recent advancements in SHM, data is becoming more readily available, and ML has the potential to help analyze and make decisions based on data. The following section gives a brief introduction and background into the field of ML by presenting its development, a simple example using linear regression, and how it is applied in the field of SHM.

## 2. 2 Overview

ML can be used in several different applications and is a rapidly developing field. One of the most popular advances in early ML came from IBM Deep Blue chess computer developed in 1997, which was able to win a match against a reigning world champion. The program utilized a database of over two hundred million chess positions in order to determine the next best move. IBM also developed Watson, that utilized multiple language analysis algorithms in parallel to analyze sentence fragments to find statistically related phrases. The application of Watson was tested in 2011 by playing a game of Jeopardy! against champions Brad Rutter and Ken Jennings and winning. These applications show how useful and powerful ML can be when implemented using appropriate training data, since then great strides have been made and ML can be seen in many applications. These instances of ML algorithm implementation are seen in many industries and has found a significant place in today's world with applications in speech recognition, costumer service, computer vision, recommendation engines, and even automated stock trades.

Typically, ML is divided into supervised, unsupervised, and reinforcement learning depending on available data, type of data, and application. Supervised methods are the most common approach and requires a dataset that is labeled for both input and outputs. Labeled data is necessary because supervised approaches learn patterns in data during a training phase. Once algorithms are trained, they are able to predict or categorize outputs based on inputs. In damage detection this would require a dataset that has examples of both damaged and undamaged structural condition. Common methods for this approach include linear regression, logistic regression, K-nearest neighbors, Artificial Neural Networks and Convolutional Neural Networks. Unsupervised approaches attempt to predict an output without labeled training data. This approach can be a goal itself, in that is requires unique approaches. Unsupervised approaches in structural engineering typically try to achieve damage detection through anomaly detection, in which the undamaged dataset is initially utilized to either set a baseline for what is normal or for training of a network. Damage is then detected through determining if a dataset has anomalies based on differences in output when compared to "normal" output. When implemented for damage detection the ML algorithm is not shown any damaged data during the training phase. Unsupervised approaches can be preferable to supervised in damage detection, since the availability of damaged datasets is not always available. However, the implementation and development of unsupervised methods can be more difficult. Reinforcement learning interacts in an environment in which it must perform a task. This approach is not commonly used in SHM damage detection. Moreover, once using an unsupervised method an anomaly would be detected, the next step would be to characterize the damage type and its severity, and finally, prognosis. The latter cannot be accomplished using an unsupervised method.

An example of a supervised ML algorithm in its simplest form is linear regression. Which predicts a response Y based on inputs X, by fitting a function to it during training. This approach assumes that there is a linear relationship between X and Y. Mathematically it can be represented as:

$$Y \approx \beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n + \epsilon \qquad \text{Eq. 1}$$

In this form the $\beta_i$ represent the coefficients or weights of the equation that are meant to capture the relationship between the inputs $X$ and the output $Y$. The learning of this method is done by finding an optimized set of β values that maximize the likelihood that predicted $Y$ terms would be equal to the actual $Y$ terms. This can be done by minimizing an error term, which is commonly taken as the residual sum of squares, which is defined by Eq. 2 below (James et al., 2021)

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y_i})^2. \qquad \text{Eq. 2}$$

In this equation $\hat{y_i}$ is the predicted and $y_i$ is the actual. Minimizing the value of this function by adjusting the weights is what takes place during the training process. This is the basic goal in any supervised machine learning algorithm, however the parameter that is used for adjusting weights and the way in which weights are applied to inputs can change. For example, some approaches seek to minimize the mean squared error (MSE) or to maximize the log likelihood between predicted and actual values (James et al., 2021).This is the approach to most supervised machine learning methods and is why labeled data and training is needed. In damage detection SHM systems would require a dataset that has examples of damaged data and undamaged data. While linear regression is not used for this study the method of learning patterns in a dataset through training is similar to the process used in ANNs.

ANNs have a much more complex structure but follow the same basic principle. The network takes an input vector of variables and builds a nonlinear function to predict the response $Y$.The parameters in ANNs are weights that are applied to each variable in the input layer and the output is either a categorization or prediction based on inputs. The network adjusts weights by a process known as backpropagation which seeks to maximize model accuracy by minimizing the MSE (James et al., 2021) . Backpropagation takes place during the training phase in which the algorithm learns to perform some tasks by analyzing examples. For instance, if a network were to be trained to recognize different fruits it would be feed thousands of different labeled images of apples, pears, blackberries, and so on to find patterns that the algorithm correlates with a particular label. The recognition of these patterns is loosely based on a human brain's neuron firing when it recognizes something. In this way filters or weights are applied to the input through "neurons" in the ANN, which will fire based on an activation function if the feature they are looking for is present in the image (James et al., 2021). This structure allows for a network to also capture the interactions between features present in the system, by having multiple filters that are sequentially applied to the input to build up layers of feature and pattern identification.

For this research a categorical feed forward CNN is utilized. CNNs are a type of ANN that have found popularity recently for their ability to categorize images accurately and efficiently. The main differences between ANNs and CNNs is the way the filters are applied to the input. CNNs do not have an associated weight for every input variable or are not fully connected, instead is applies smaller filters that convolve about the image. This allows for certain features to be identified in different areas of the input. In some applications CNNs have found success in a one-dimensional setting used on time series data, however this is not a typical approach and other algorithms such as recurrent neural networks (RNNs) or long short-term memory (LSTM) are more commonly used for time series data. For this reason, image data is best to be used in CNNs and has led to studies on how to represent data as images effectively (Zhu et al., 2021). In this paper an approach to representing time series strain gage data to be used in a CNN is presented in Section 4.

## 2. 3 Application in SHM

SHM has become a powerful tool in Civil Engineering and allows for engineers to gain a better understanding of structural behavior. In the past decade techniques have improved in SHM allowing for better sensors, sensor layout, and data acquisition. This improvement has led to widespread use of SHM but has presented a new challenge in Civil Engineering. With efficient systems large datasets are gathered and need to be analyzed, classical analysis by an individual is time consuming and can be inherent to bias (Teng et al., 2020). The most common way of reducing the amount of required analysis time it to set thresholds on sensors, so that if they reach a certain reading an alarm will be triggered that alerts engineers. While this is a good strategy and alleviates some data processing it leaves a lot of data unused.

The issue of too much data, also known as Big Data, may be why ML techniques have become a popular research topic in the field of Civil Engineering. This is due to the nature of ML needing large datasets for training and implementation. Several types of algorithms can be used on time series data to transform it into useful information that can aid in the asset management of infrastructure. In a data driven approach, ML is typically used on time-series data to either detect damage or any anomalies in data. By doing so engineers can use the information to determine if the behavior of a structure indicates a need for further analysis. Physics driven approaches require the tuning of FE models using time series data, which can be incredibly complex and time consuming. The potential to automate or assist in the tuning of FE models through ML algorithms is also being researched. The main goal in any ML application is to provide information to the user to assist in decision making. ML has the potential to reduce the amount of data analysis required by Civil Engineers but is not a direct replacement for engineering judgement or analysis.

# Chapter 3. Field Testing and Data Collection

Field testing and live load testing was done in previous research by Akintunde et al., 2021 in conjunction with the Midwest Roadside Safety Facility (MwRSF) at the University of Nebraska-Lincoln. A bridge deck mockup was tested with varying loading conditions and damage scenarios. In this way a robust dataset that represents not only different damage scenarios, but different vehicle loading was gathered to be used for CNN development.

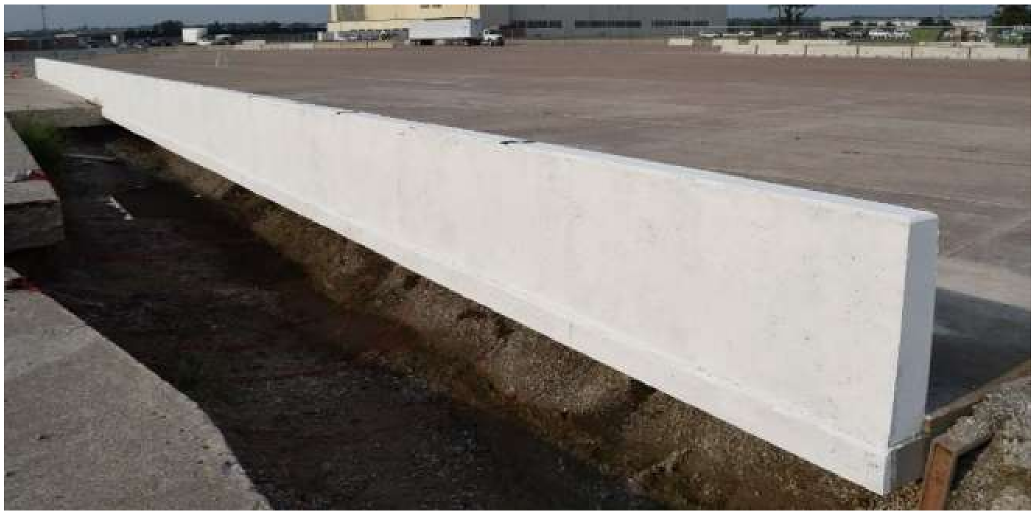


Figure 4: Bridge Deck mock-up (Akintunde et al. 2021)

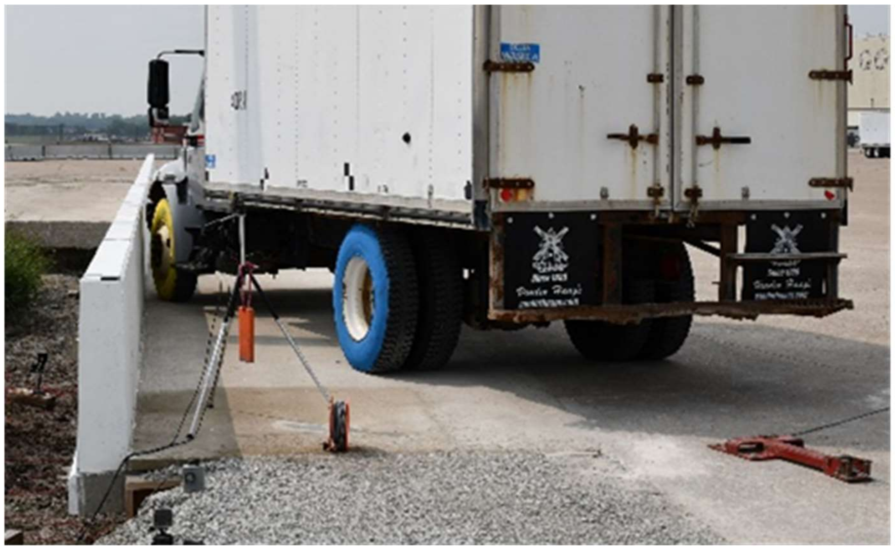


Figure 5: Mock-up and Vehicle (Akintunde et al. 2021)

## 3. 1 Structure

To create an accurate dataset representative of an in-service structure a bridge deck mock-up was used for testing and data collection. The structure was designed to replicate responses from a cantilevered portion of a bridge deck with a concrete barrier and consisted of a grade beam, deck, concrete barrier, and concrete overlay. Figure 4 and Figure 5 show the mock concrete bridge. In order to best represent an in-service structure the grade beam was designed to replicate the realistic moments and forces that a cantilevered bridge deck would see. As well, the grade beam was fully supported along its length by the soil underneath. The structure was designed in accordance to three design cases that were provided in Section 13 of American Association of State Highway and Transportation Officials (AASHTO) LRFD BDS and design loads recommended by National Cooperative Highway Research Program (NCHRP) (Akintunde et al., 2021).

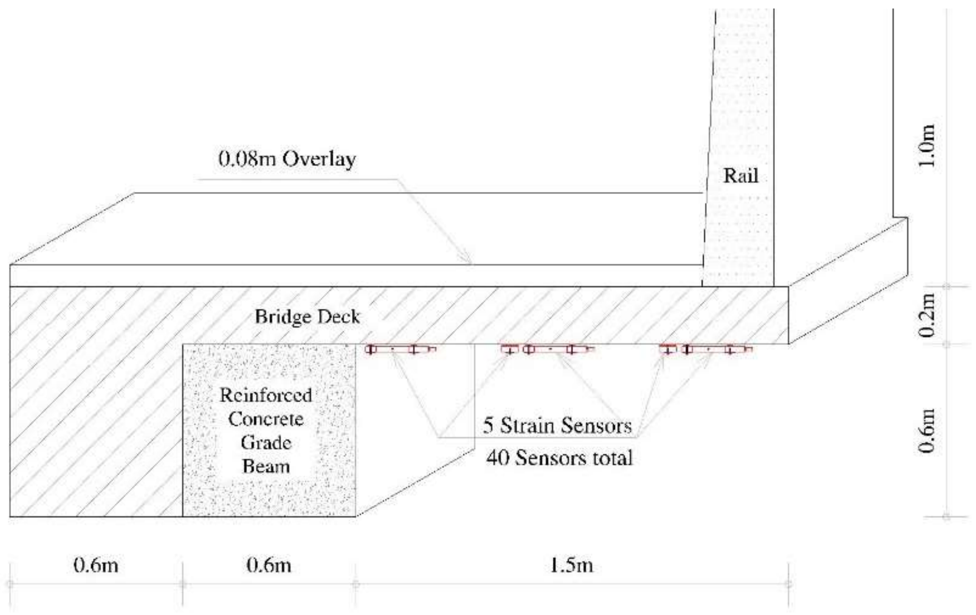


Figure 6: Mock-up section view (Akintunde et al. 2021)

## 3. 2 Instrumentation

To capture the behavior of the bridge under live load it was instrumented with several strain gages. Forty resistive based strain gages manufactured by Bridge Diagnostics, Inc. were used across the structure. An example of gages used can be seen in Figure 7. The instrumentation consisted of eight rows of three gages spaced equally across the length of the bridge. Each row consisted of three transversely oriented gages located on the outer edge, the middle and near the grade beam along with two longitudinally oriented gages located on the outer and middle gage columns. Transverse gages were oriented perpendicular to traffic while longitudinal gages were parallel to the direction of traffic. Data was recorded for all strain gages during live load testing. For this study only the twenty-four transversely oriented gages were used, the overall gage layout is shown in Figure 8.

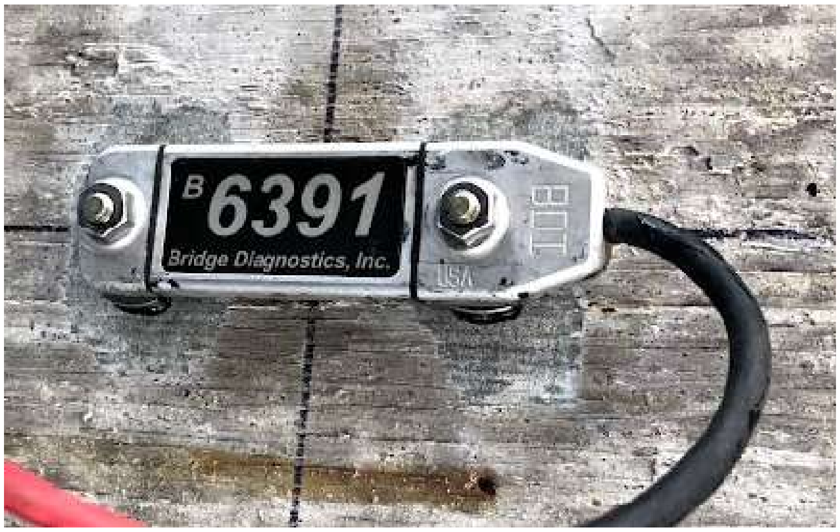


Figure 7: BDI Strain Gage

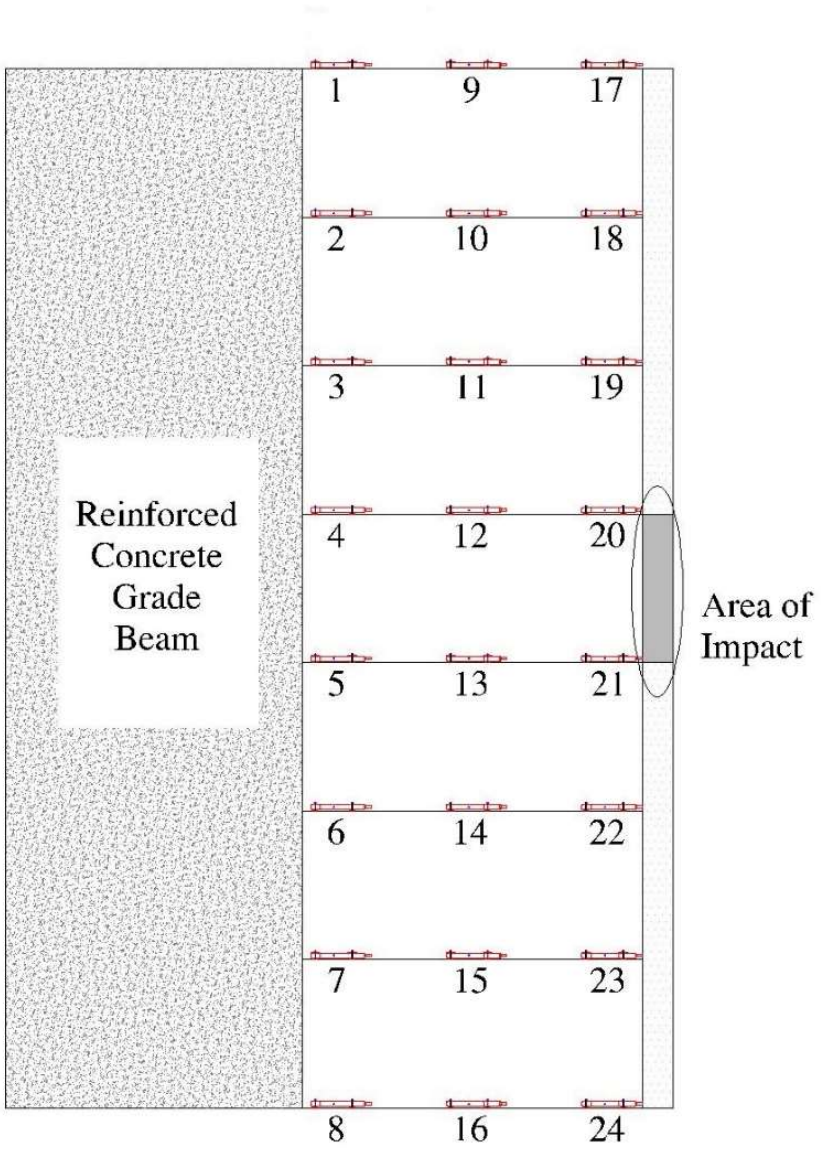


Figure 8: Overall Gage Layout (Akintunde et al. 2021)

## 3. 3 Testing

Live load testing of the mock bridge consisted of varying the damage level and loading of the bridge in order to create a robust dataset that captured the bridge response under varying conditions. The bridge was tested using two different trucks, a dump truck and a pickup truck, seen in Figure 9 and Figure 10, respectively. The dump truck was run over the bridge at 5mph both fully loaded and empty. The pickup was run over the bridge with a constant weight but at varying speeds of 5mph, 10mph and 15mph.  In total there were five different loading scenarios that the bridge was subjected to for each damage scenario. For each combination of loading and damage scenario a test was ran where strain time histories across all sensors was recorded at 100 Hz. A summary of loading scenario and loading conditions can be found on Table 1.

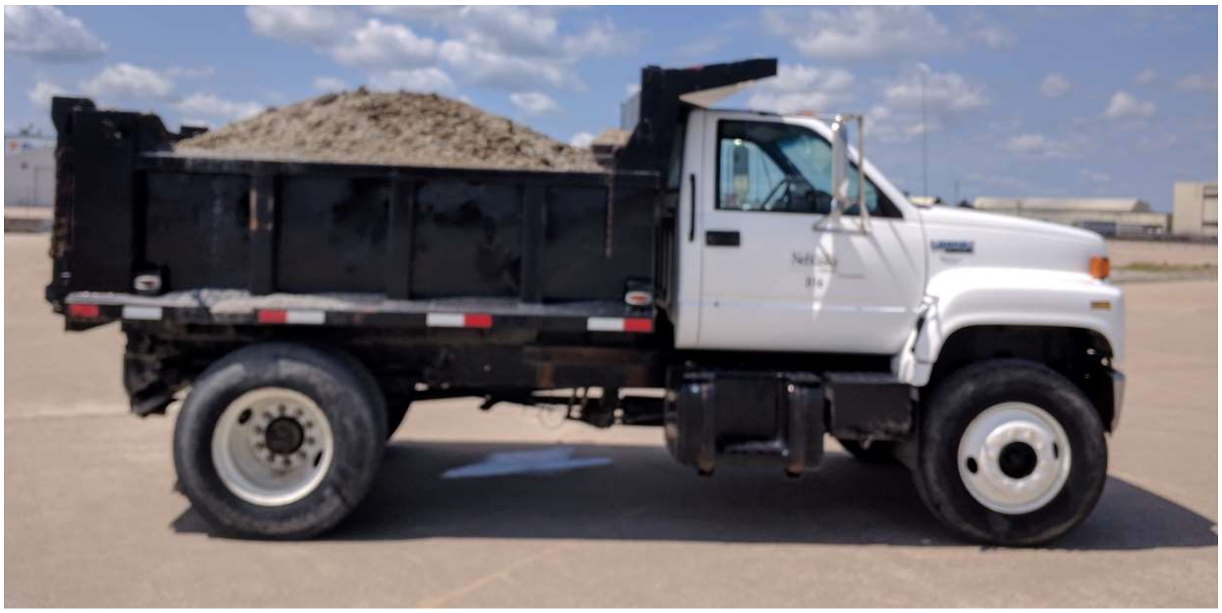


Figure 9: Full Dump Truck (Akintunde et al. 2021)

Figure 10: Pickup Truck (Akintunde et al. 2021)

Table 1: Summary of Tests

| | | Damage Condition | | | |
|---|---|---|---|---|---|
| **Vehicle** | **Speed (mph)** | **Undamaged (UN)** | **Crash Induced (D1)** | **Concrete Barrier Damage (D2)** | **Deck Damage (D3)** |
| **Dump Truck Empty (DTE)** | 5 | 23 | 21 | 23 | 23 |
| **Dump Truck Full (DTF)** | 5 | 21 | 21 | 23 | 23 |
| **Pickup Truck** | 5 | 9 | 10 | 12 | 13 |
| | 10 | 10 | 12 | 13 | 13 |
| | 15 | 12 | 12 | 13 | 14 |

## 3. 4 Damage Scenarios

The first damage scenario tested was the “healthy” or undamaged (UN) case where no damage was induced to the structure. Once testing of the undamaged state was completed the bridge was subjected to a crash on the barrier by a 10-ton box truck traveling at 58 mph which impacted the barrier at shallow angle. The state of the bridge after impact was labeled the crash induced case (D1), this type of impact was chosen due to how frequently it is seen on in-service bridges. The truck used for the crash induced condition is shown in the figure below. The crash induced

Figure 11: Crash Induced Damage (D1) (Akintunde et al. 2021)

condition resulted in minimal visible deck damage. The result after collision can be seen in Figure 12 a number of “gouges” were identified on the barrier. A “C” shape gouge was present 9.3 in from the target location and 12.5 in below the top of the barrier. The gouge was 8.74 in long and was 6.77 in deep. The presence of another gouge is noted that existed 8.5 in below the top of the barrier; beginning 11 in downstream from the target and extending for 106.3 in. A small gouge was noted 15.9 in below the top of the barrier, 17 in downstream of the target and extending for

22 in. An additional fourth gouge was located 39 in downstream of the target and extending for 130 in. The sharpie markings shown in Figure 12 are used to place where sensors are in relation to gouges. During the crash, the barrier also vibrated transversely from its original position and returned to its original position after contact. (Akintunde et al. 2021)

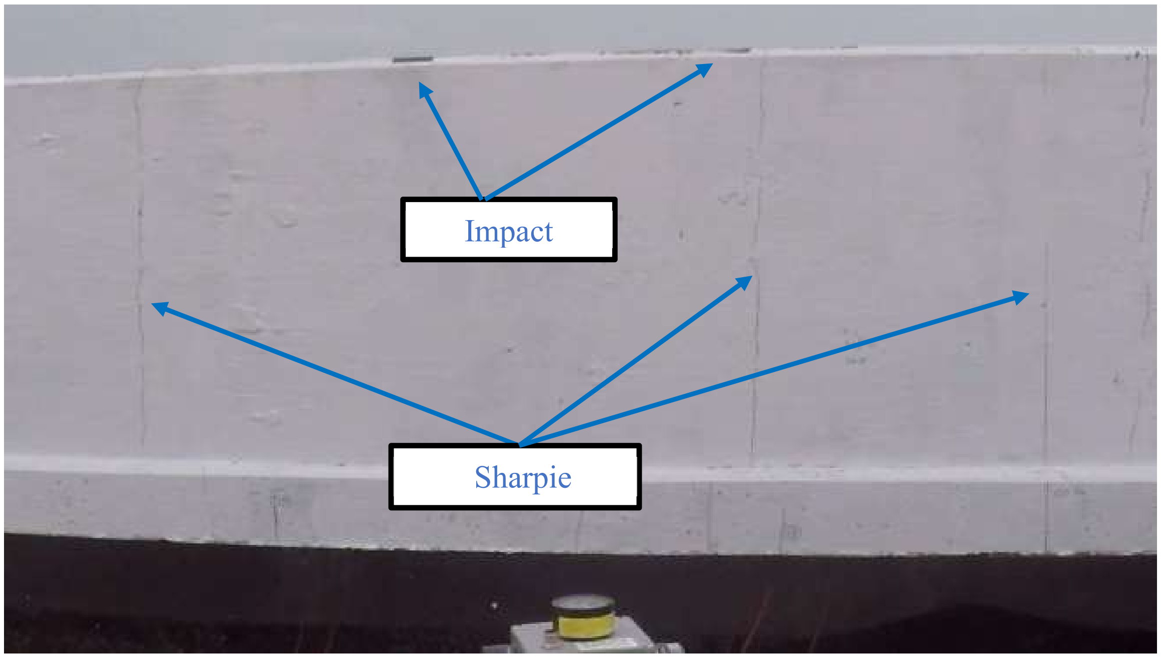


Figure 12: Outside Face of Barrier after Impact (Akintunde et al. 2021)

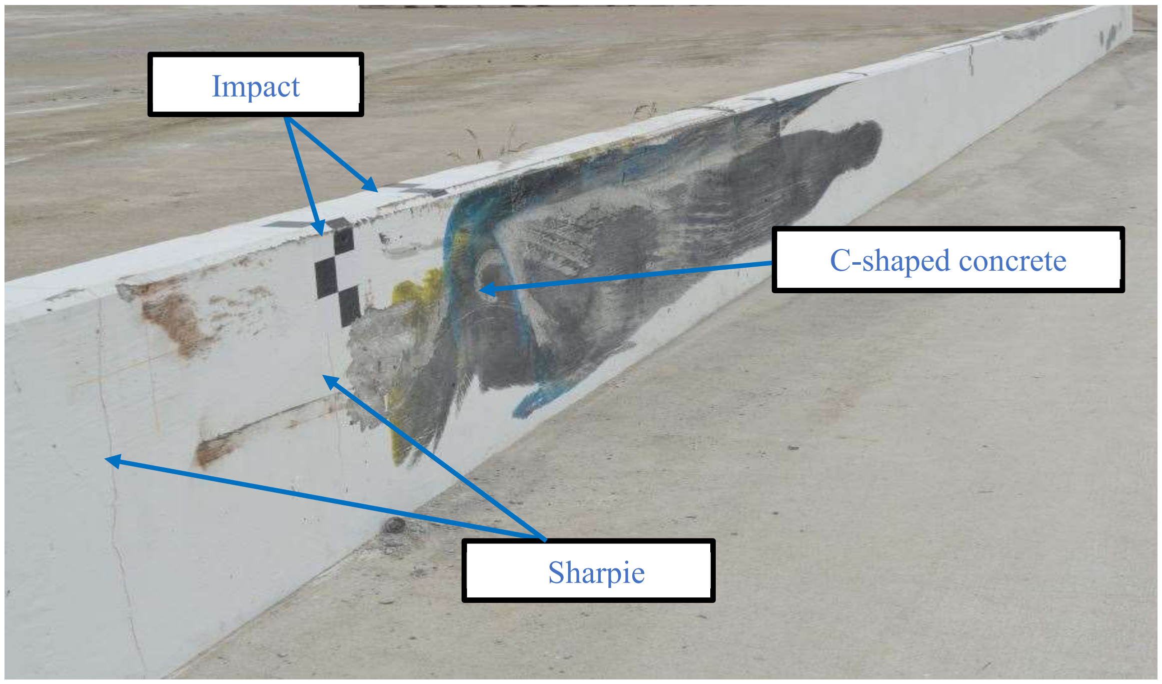


Figure 13: Inside Face of Barrier after Impact (Akintunde et al. 2021)

The second damage condition labeled; concrete barrier damage (D2) was induced by taking a full length cut transversely along the barrier at midspan of the bridge. Similarly, the third damage condition, deck damage (D3), was induced by extending the saw cut at the barrier along the width of the bridge deck. These cuts reduce the stiffness of the structure significantly and can be representative of section loss in an in-service structure. For each damage scenario and loading condition several vehicle passes were done to ensure enough data is gathered for analysis.

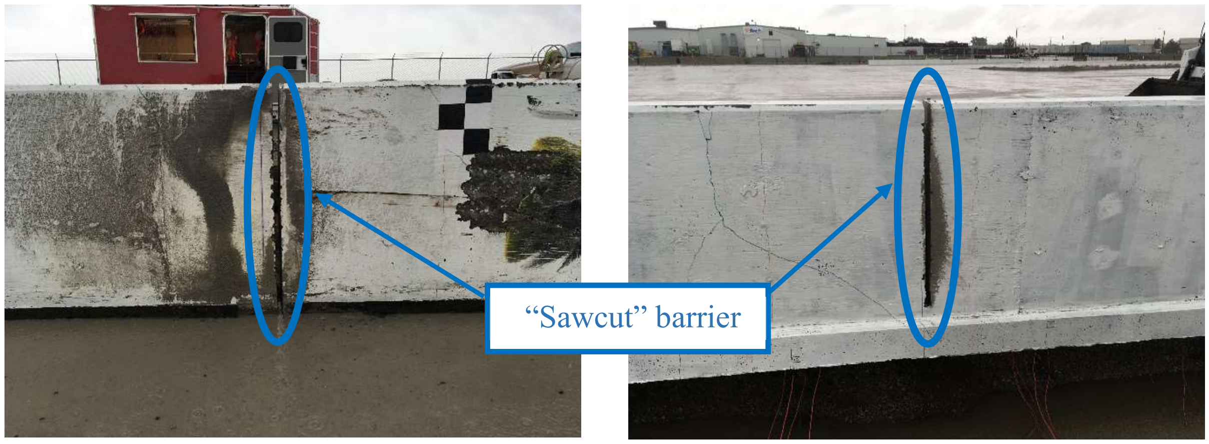


Figure 14: Inside (left) and Outside (right) Face of Barrier after Concrete Barrier (D3) (Akintunde et al. 2021)

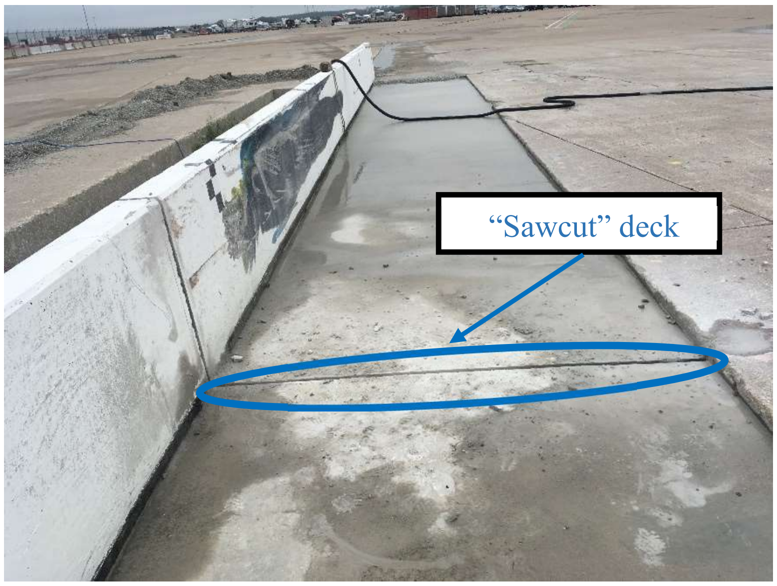


Figure 15: Deck Damage (D3) (Akintunde et al. 2021)

# Chapter 4. Data Processing

## 4. 1 Overview

Once testing and measurements were completed, the data was verified for valid readings and the data was stored in MATLAB snapshot matrices. The snapshot matrices contained the time histories across all sensors for an individual run. Data was cleaned using a lowpass filter and linear drift filter. Lowpass filtering is common in practice and helps remove any unwanted noise from the sensor that occurs during testing (*Strain Gauge and Wheatstone Bridge - MATLAB & Simulink*, n.d.). Linear drift is another common filter that is applied due to the nature of gage readings to drift throughout testing. This drift can usually be attributed to the gage temperature increasing during testing due to current being ran through it to gather data. This change in temperature effects these readings because the sensors work using high accuracy resistive based measurements that can detect even slight changes in temperature (*Manual-Strain-Transducer-ST350-201511-Rev-A-.Pdf*, n.d.).

The main goal of this phase of experimentation was to find a way to represent data quickly and accurately to be used in CNNs. As discussed earlier two dimensional CNNs typically perform best when being implemented on images. This being the case a way to represent time series strain gage reading as a grey scale image was developed. The result is 2D greyscale images that capture the response of all sensors on the mock bridge for each individual run.

## 4. 2 Time Series Data

As mentioned previously original data was gathered and stored as time series in snapshot matrices. Strain responses for undamaged and damage conditions under a full dump truck live load of 11 tons are shown in Figure 16 and Figure 17. Each color in the graph represents the response from each sensor. The Y-Axis shows strain in micro strain (με) and the X-Axis shows time. The

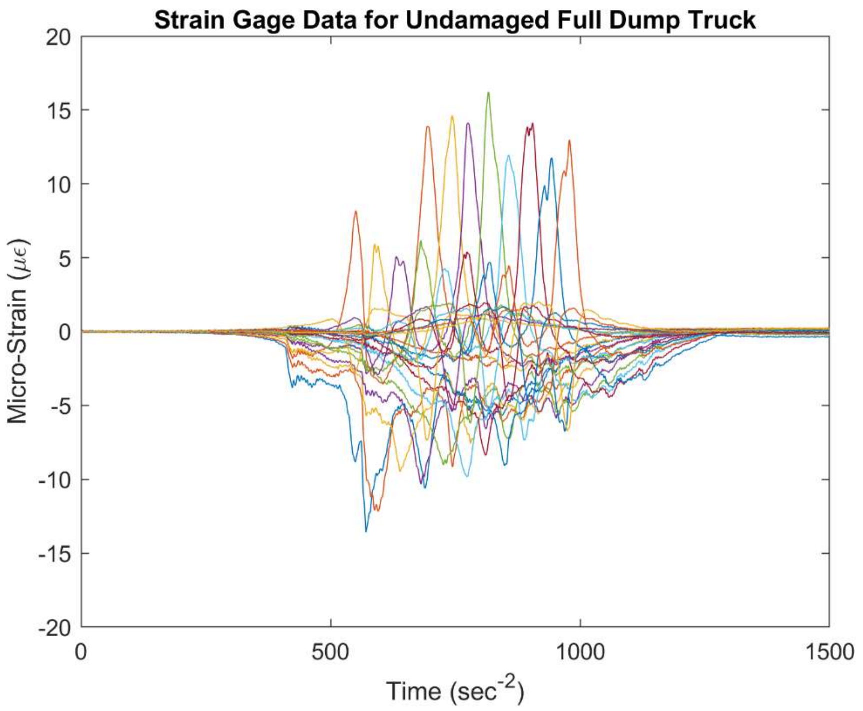


Figure 16: Strain Time histories for Undamaged Bridge under Full Dump Truck Loading

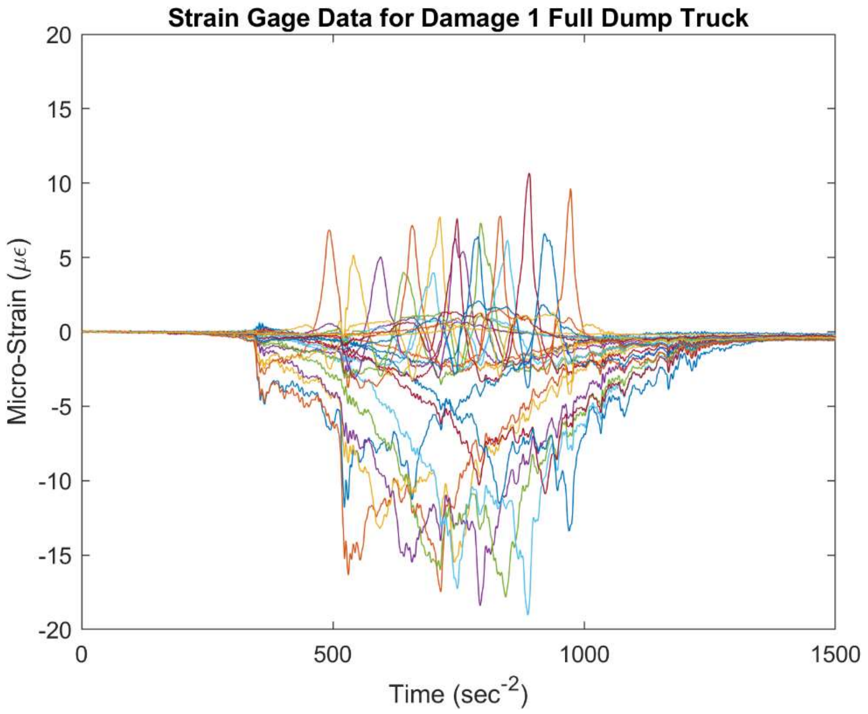


Figure 17: Strain Time histories for Damage 1 Bridge under Full Dump Truck Loading

comparison between strain measurements of the damaged and undamaged state can help better understand the structures behavior. The undamaged condition shows higher positive strain response while the damaged condition shows higher negative strain response. These features can be important when analyzing a structure but are not a clear indication of damage. Figure 19 and Figure 18 shows the same bridge conditions under loading from a 2.5-ton pickup truck, which is significantly lighter than the full dump truck. Again, some differences are seen between the undamaged and damaged condition, however when comparing the same bridge condition under different loading conditions it becomes apparent why variational loading is important. If only one loading condition were to be considered the peak gage response may be weighted too much in the damage detection framework. Having a dataset that captures variational loading along with different damage conditions is integral in developing a robust damage detection framework.

To better understand how these images may be converted, another representation of data is shown in Figure 20. The three-dimensional surface plot shows the same data as previous figures however now the different strain gages, that were once shown as different colors, are represented on an axis. By doing this the interactions between gages are easier to identify, recall that the gages are installed in three columns of either, with gages one through eight being in the same column and so on. As seen on the figure gage columns generally have the same sign for strain. This interaction of gage readings may be an important feature in distinguishing damage. This being the case the decision to keep all gage readings represented as one image was made as to not lose any important information from runs. Based on the brief analysis of time series data, the images to be used in the CNN had to be representative of all gages during a run and account for variational loading.

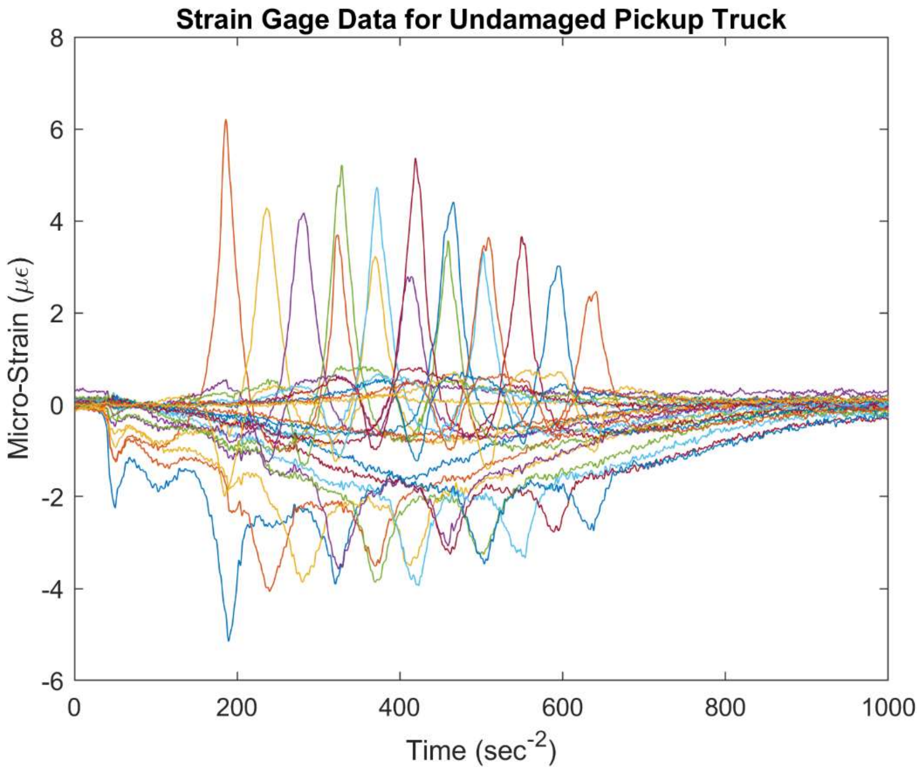


Figure 19: Strain Time histories for Undamaged Bridge under Pickup Truck Loading

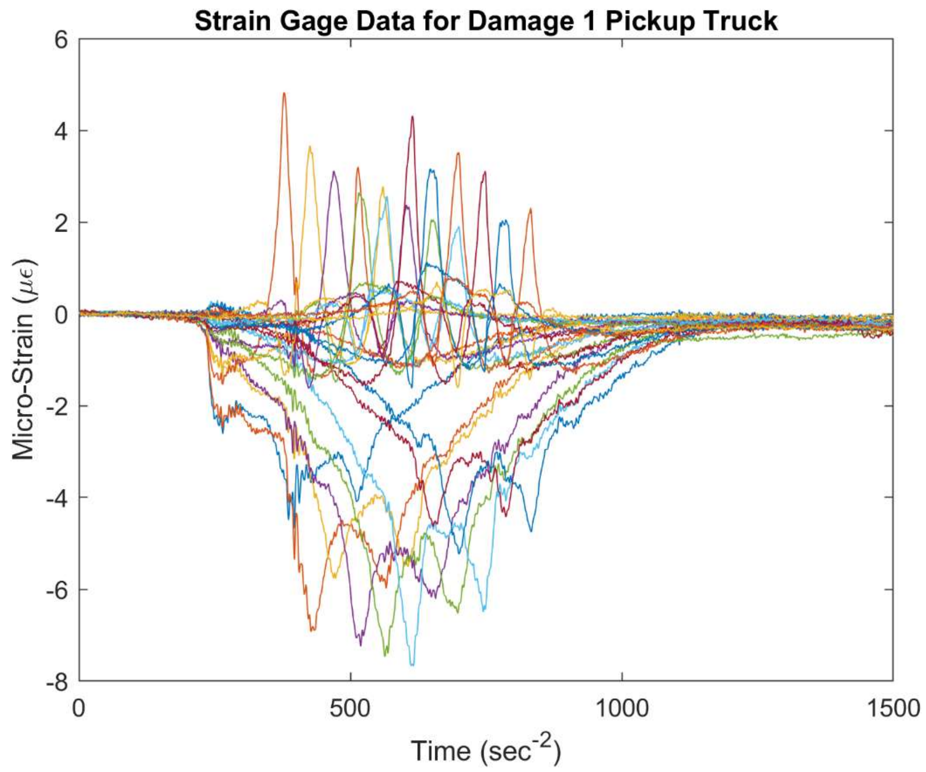


Figure 18: Strain Time histories for Damage 1 Bridge under Pickup Truck Loading

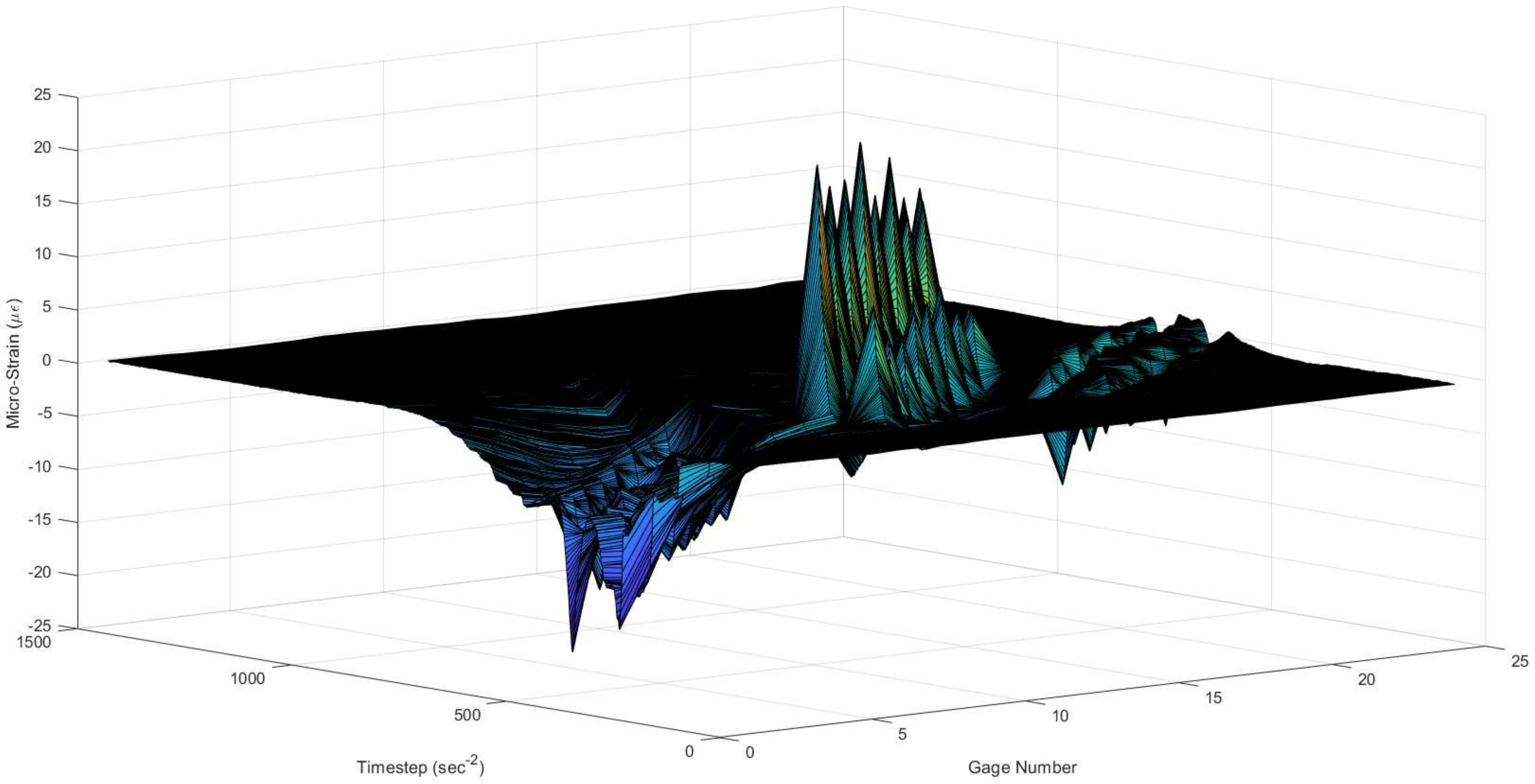


Figure 20: Strain-Time histories represented as 3D Surface Plot

## 4. 3 Representation as Image

CNNs have been shown to be most successful in image recognition application. Therefore, to increase the accuracy of the network, the time-series data was converted into two-dimensional greyscale images that represents all sensor readings across a run. The images to be used in the network are of dimensions 1000 by 24 by 1. In this format the 1000 refers to the timestep of the data, 24 refers to each gage, and 1 refers to the strain value, which are converted into image height, width, and pixel value respectively.

Data first had to be normalized to accurately convert strain readings to pixel values. Normalization was done by finding the maximum and minimum strain value across all runs and using z normalization to have all readings in a range of 0 to 1. Across all runs the max and min strain value were found to be 34 and -199 micro strain respectively. Using these two values all strain gage readings were normalized using Equation 3, below.

$$z_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}, \qquad \text{Eq. 3}$$

where $z_i$ is normalized strain reading, $x_i$ is the original strain reading, $min(x)$ and $max(x)$ are the minimum and maximum reading of strain across all runs, respectively. Through normalization all strain values are now between 0 and 1, with 0 being equivalent to the max negative value and 1 being equivalent to the max positive value. Once normalized the datasets are converted to a greyscale image using the imwrite() function in MATLAB. This function writes an image from a specified dataset into either a greyscale or RGB color. For a greyscale image the function assumes a range of 0 to 1 for data values and converts them to pixel values of 0 to 255. This range of 0 to 255 is based on typical greyscale images and is used for this research as well (Zhu et al., 2021).

The final pixel values after writing to an image is therefore proportional to the strain, now with the minimum strain proportional to a pixel value of 0 and the maximum strain being proportional to a pixel value of 255. For greyscale images pixel values of 0 are black and 255 are white, so in the final images white pixels will correspond to higher strain values and vice versa.

The last step for data representation is to convert images into a consistent size of 1000x24x1. The only dimensions that is not consistent between runs is the height of the image that varies from 1000 to 1500. As to not filter out any information all images were scaled down to the smallest dimension in the range. To do this the imresize() function in MATLAB was utilized which uses bicubic interpolation to resize image. The output pixel value for this method is a weighted average of pixels in the nearest 4-by-4 neighborhood. This ensures the final resulting images are all consistent dimensions.

An example for final images created from responses measured from each damage condition is shown in Figure 21. Two examples of the undamaged scenario are shown to display the difference in measurements between runs on the same bridge condition. Figure 22 and Figure 23 show a final verification of the approach adopted in this study to ensure the pixel value was proportional to the measured strain time histories. Figure 22 was created by plotting original time series data while Figure 23 was created by reading the final image back into MATLAB using the imread() function and then plotting the pixel value on the same axis as strain. By comparing the two-surface plot it was confirmed that the images correctly represented strain values across an entire run.

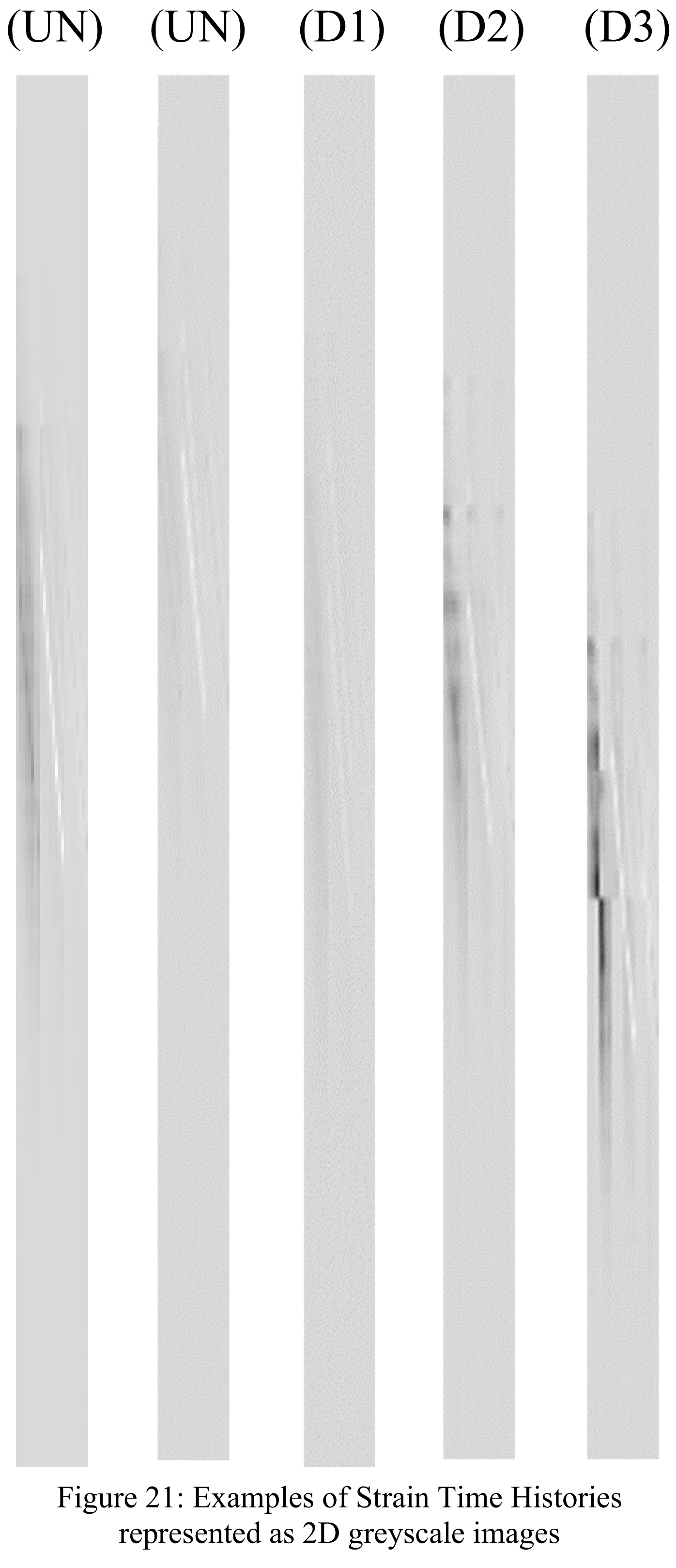


Figure 21: Examples of Strain Time Histories
represented as 2D greyscale images

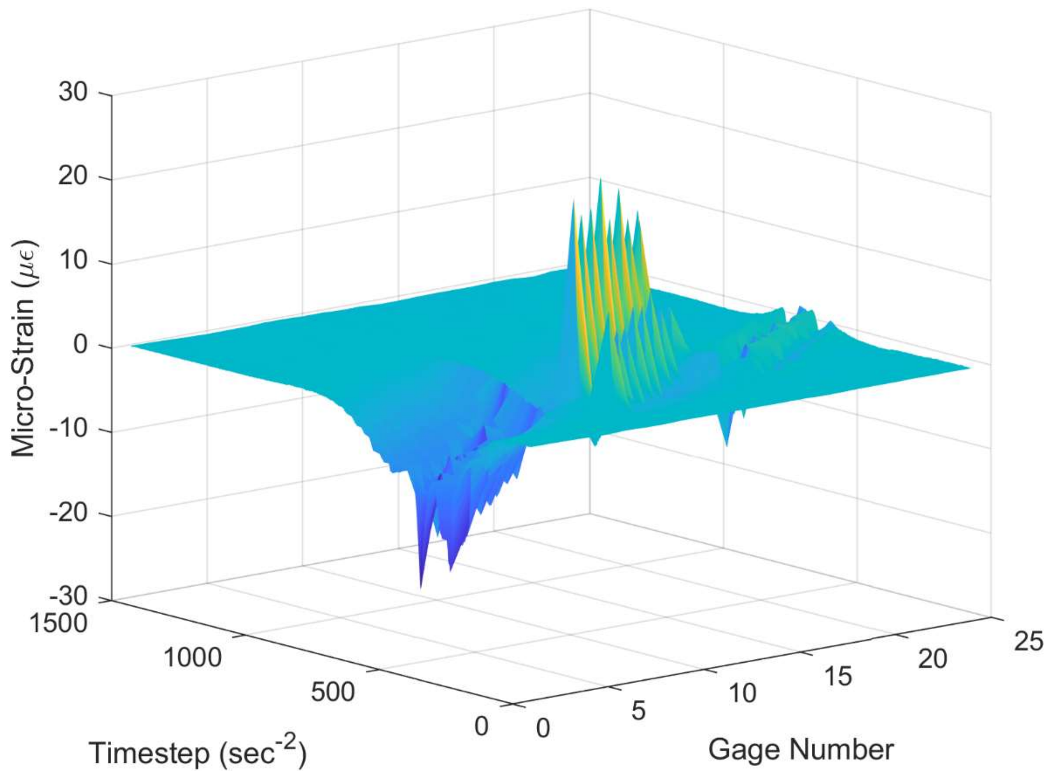


Figure 22: 3D Plot of Original Data

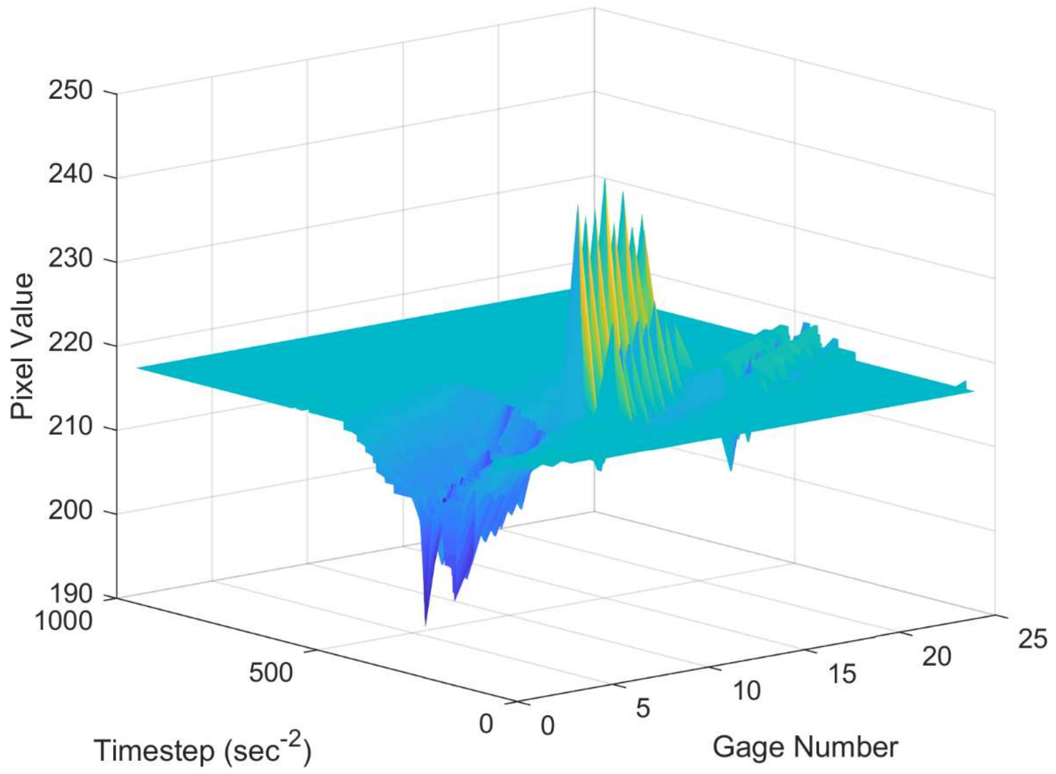


Figure 23: 3D Plot of Greyscale Image representing Data

## 4. 4 Image Use in CNNs

In order to properly train the CNN, images not only need to be representative of the data but also the need to have a damage scenario label associated with them. This was simply done by filing the images into folders that had the corresponding damage scenario. This filing structure is then used as a directory when the CNN is accessing the images for training. After conversion of data, images had to be separated into training, testing, and validation categories for CNN. Training and validation data are used for model training and hyperparameter tuning. These images are also what the model ultimately used to determine filters to apply to data in order to classify images. Testing data is used as a way of testing the final accuracy of the model without adding any bias. This is achieved by only predicting using testing data after the model is trained and finalized, and the reported prediction accuracy pertain to testing data.

A total of 321 images were available for CNN modeling after data conversion. Two different models were trained one that used 30% training data and one that used 70% training data. This was done in order to test the robustness of the model regarding limited training data. The breakdown of used images is outlined in the table below.

Table 2: Training, Validation and Testing Data split for Models

| | Training Data | Validation Data | Testing Data |
|---|---|---|---|
| Model 1 (30% Training Data) | 25% (81 images) | 5% (16 images) | 70% (224 images) |
| Model 2 (70% Training Data) | 70% (32 images) | 10% (32 images) | 20% (64 images) |

This section outlines the methods used to process data in order to be used in the CNN model building and testing phase of the study. The final process for converting time series data to images can be easily implemented in a short MATLAB script (Appendix A). Final images are easily stored and accurately represent strain gage values as pixel values. The model therefor does not require any feature extraction from time histories and does not require extensive data processing. This was intentionally done as the methods proposed in this study seek to minimize the amount of time required for data processing and analysis. Once all images are converted and verified, they are used to train a CNN to categorize them as one of the four damaged conditions of the bridge.

# Chapter 5. Methodology

The main goal of the research presented in this chapter is to develop the architecture and tune the training parameters for a CNN in order to efficiently and accurately categorize strain gage data as undamaged or damaged with various levels. Efficiency of the network is defined as the time it takes to properly train a network and accuracy is defined by the test set classification error. In this section a brief overview of how Neural Networks and CNNs work will be presented. Then the proposed architecture and training parameters for a damage detection framework that utilizes CNNs will be presented.

## 5. 1 Introduction

CNNs are a type of ANN, which are particularly well equipped for image recognition and classification (James et al., 2021). As an example, a fully connected neural network is considered which is like a linear regression algorithm in that it takes an input vector of variables and builds a

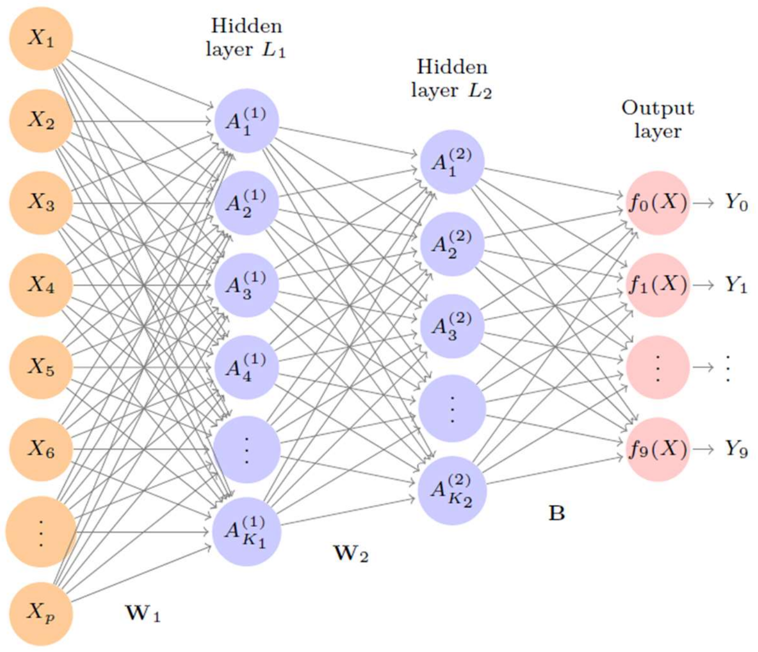


Figure 24: Fully Connected Neural Network Diagram (James et al., 2021)

function to predict a response Y. As opposed to linear regression this the ANN has nonlinear functions in its layers and can have substantially larger number of weights used in it. In the instance of this research the inputs would be pixel values from a labeled image and the output would be the damage scenario classification of the input image. In a fully connected network prediction is achieved through a number of layers that apply different weights to inputs in order to detect features and classify based on them. Figure 24 shows a basic architecture for a multi-layer neural network and is used to explain how a basic ANN works. In this figure the weights being applied are denoted by the black lines, the inputs by $X$s, and the neurons are labeled $A_k^{(1)}$ and $A_l^{(2)}$. In a fully connected network, each neuron applies a weight to each input value. The output for this example is a classification layer that has the likelihood that the input belongs to a class 0 through 9. ANNs can also have regression outputs for a range of other application, this research however only considers classification. ANNs have a distinctive architecture in that the original image gets transformed once the first layer is applied, in this way the inputs to the second layer are the outputs from the first layer. Through this the network can build a complex transformation of the input that ultimately feeds into the classification output layer as features. This allows for the interaction of certain features to be captured. A mathematical form of neuron values in layer 1 and 2 is defined below:

$$A_k^{(1)} = g(\omega_{1,0} + \sum \omega_{1,p} X_p), \qquad \text{Eq. 4}$$

$$A_l^{(2)} = g(\omega_{2,0} + \sum \omega_{2,k} A_k^{(1)}). \qquad \text{Eq. 5}$$

In these equations, $A_k^{(1)}$ and $A_l^{(2)}$ are the neuron values for layer one and two, $\omega_{1,0}$ and $\omega_{2,0}$ are bias terms and $\omega_{x,y}$ are the weights being applied to the inputs of the layers. The resulting value from summing the weights applied to the inputs and adding a bias term is then fed into an activation function denoted by g(). This function is meant to loosely replicate the firing of a neuron in a human brain. If the value is below a certain number the value will be zero or very low, however if the value is greater the function will yield a high value. Figure 25 shows an example of two

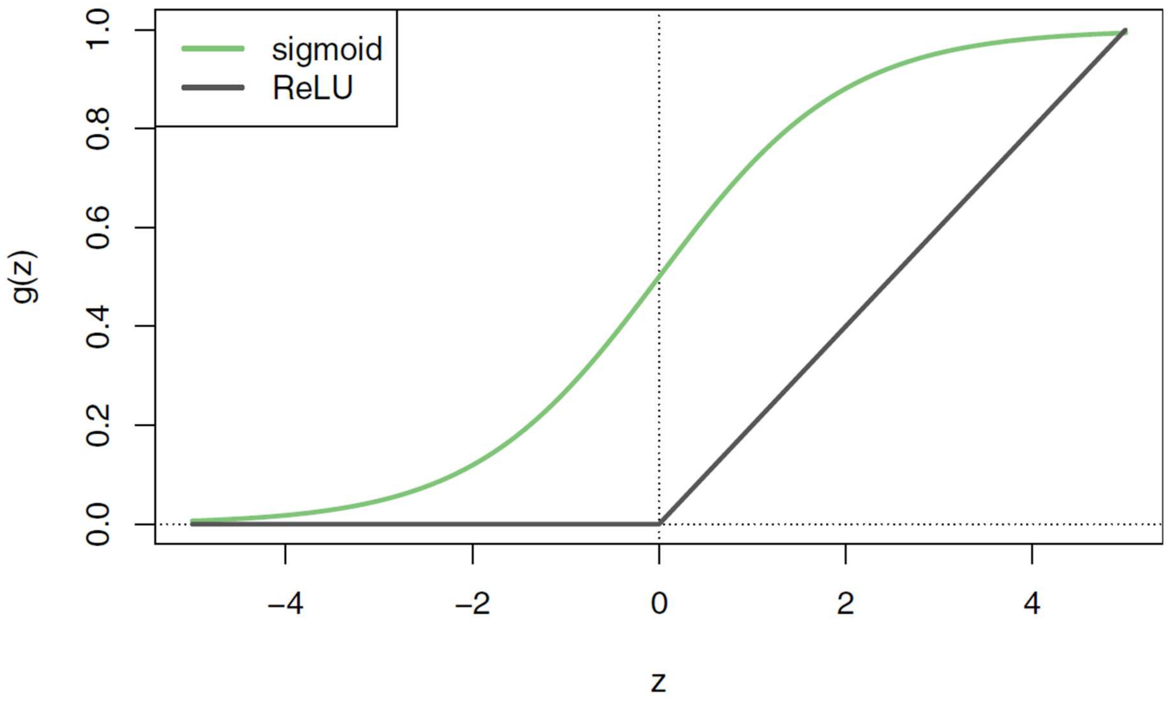


Figure 25: ReLu and Sigmoid Activation Functions (James et al., 2021)

common activation functions, a sigmoid function is shown in green, and a piecewise-linear (ReLu) function is shown in black (James et al., 2021). The final value of the function is on the Y-Axis denoted by $g(z)$ and the number fed into the function on the X-Axis denoted by $z$. In the case of a fully connected network the number $z$ would be as defined above, the sum of weights multiplied by inputs with an added bias term. This function is "activated" based on the value of $z$, when looking at the ReLu function if $z$ is above 0 the functions value increases quickly, however if it is less than zero it is always 0. There are many different activation functions that can be used in ANNs for different applications and performance reasons. The most used function today is the ReLu function defined below (James et al., 2021):

$$g(z) = \begin{cases} 0 \; if \; z < 0 \\ z \; otherwise \end{cases}. \qquad \text{Eq. 6}$$

This simple activation function can be computed and stored more efficiently than others making it preferable, since training networks can become computationally expensive. The final output layer in this example is a categorical SoftMax layer. The initial value feeding into the 9 categorical neurons in the final layer can be defined as,

$$X = Z_m = \beta_{m0} + \sum \beta_{ml} A_l^{(2)}. \qquad \text{Eq. 7}$$

This is similar to the defined Eq. 5 for layer 2, but now is not assigned to an activation function. Instead, since the goal of this network is to categorize the estimated $Z_m$, is represented as class probabilities of outputs 1 through 9. To do this a SoftMax layer is used:

$$f_m(X) = \Pr(Y = m | x = X) = \frac{e^{Z_m}}{\sum_{l=0}^{9} e^{Z_l}}. \qquad \text{Eq. 8}$$

This ensures that the final ten outputs, $f_{0-9}(X)$, are non-negative and sum to one, in other words each output neuron value is transformed into a probability that the input belongs to that class. Using this final layer of probabilities, the network then assigns the input to the class that has the highest probability.

This is the basic way in which an ANN classifies inputs, through a series of fully connected layers that transform the input to identify certain features and interactions and classify based on them. CNNs work similarly in that the outputs are functions of weights and activation functions, but the way the weights or filters are applied is different. CNNs in some capacity mimic how humans classify images, by recognizing certain feature or patterns anywhere in the image (James et al., 2021). The weights in these networks are commonly referred to as filters and are usually smaller than the input image. Meaning that for values in the input layer there is not necessarily an individual weight associated with it per neuron. Instead filters search for instances of small patterns in the image by convolving a set of weights about the image in convolutional layers (Madhavan, 2021). A convolutional filter is a template that determines whether a particular feature is present at any point in an input image (James et al., 2021) . To put it in mathematical terms an image can be thought of as a matrix of pixel values.

$$Input\ Image\ = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & g \end{bmatrix}. \qquad \text{Eq. 9}$$

A convolutional filter can also be thought of as a matrix of weights, in this instance a 2 by 2 matrix:

$$Filter\ = \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}. \qquad \text{Eq. 10}$$

The filter is then applied to the input image and the output is a new convolved image, which is a function of the filter being applied to the input image.

$$Convolved\ Image\ = \begin{bmatrix} a\alpha + b\beta + d\gamma + e\delta & b\alpha + c\beta + e\gamma + f\delta \\ d\alpha + e\beta + g\gamma + h\delta & e\alpha + f\beta + h\gamma + g\delta \end{bmatrix}, \qquad \text{Eq. 11}$$

It is important to note that the dimensions of the convolved image are a function of filter size, step, and padding. Filter step being the amount the filter moves both horizontally and vertically across the images and padding being the number of zeros added around the image. Padding is done for a filter to be able to detect parts of features that appear only partially on the edge of an image and not entirely on the image. The actual filter values are learned through a process called back propagation that is discussed later in this section. The filters of a CNN can also be applied in multiple layers similarly to a fully connected network. In this way the first layer can identify low level features in the image, such as edges of color patches and combine them in subsequent layers to form higher level features that can identify a range of different objects depending on application.

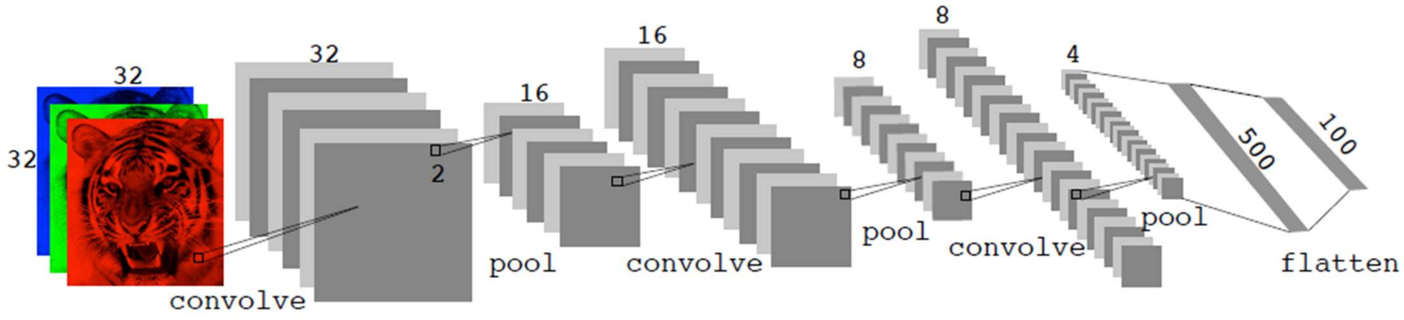


Figure 26: CNN Examples (James et al., 2021)

This application of a convolving filter can also act as a dimensionality reduction of the original image, effectively transforming it into a new image of smaller dimensions. An example of a CNN meant to classify images of animals is shown on Figure 26. In this example the convolutional filters are denoted by the black box outline on the original image and the convolved image is shown by the greyscale squares in the subsequent layer. The input layer in this case is a three-dimensional image of 32x32x3 dimensions, where the third position represents the color channels RGB. The first convolutional layer reduces the color dimension but effectively increases the third channel by having multiple filters applied. In this case six convolutional filters are applied to the original image, denoted by the number of grey squares in the first layer, resulting in six new convolved images. The second layer shows a pooling layer that reduces the size of the feature map and helps with computational costs of a network by efficiently reducing the dimensions of layers. Pooling is usually applied with a 2 by 2 filter and either takes the maximum, minimum or average of the pixels in the filter. In the example the convolution layer to pooling layer sequence is repeated until the two-dimensional features are small enough to be flattened, where pixel values are treated as separate units. The network then works similarly to the fully connected network discussed previously with a SoftMax output layer.

This is a basic overview of how ANNs and CNNs work, this research utilized a 2D categorical feed forward CNN that utilizes the same concepts discussed. The inputs to the network are strain gage measurements represented as 2D grey scale images. The images are labeled based on the damage scenario the bridge was subjected to during testing. This being the case the network ultimately categorized input images as either Undamaged (UN), Damage 1 (D1), Damage 2 (D2), or Damage 3 (D3). The architecture and model training parameters were determined using an iterative process to reduce the network training time and improve the accuracy of the network prediction. Model accuracy during the development of architecture and training parameters was determined using the validation data and training data accuracy. This is a common method used in ANN development as it reduces bias in final model decisions image (James et al., 2021).

## 5. 2 Model Architecture

The first phase of developing the CNN model was to determine an architecture. Development and testing were done using the MATLAB Deep Learning Toolbox and Experiment Manager. The architecture of a network refers to the number and size of filters, step size or stride, padding, type of pooling and organization of filters. To develop an intuitive procedure for damage detection, two models that used filter sizes based on gage location and gage layout were compared to a more common architecture in a third model. The validation and training accuracy was used to determine which architecture performed the best.

Recall that the original size of images to be used in the CNN was 1000 by 24 by 1, representing time step, gage number and pixel value respectively. To test accuracy of filter sizes three different general layouts were tested. The first two architectures determine filter size directly based on gage layout and the third determines filters based on architectures that have been generally found to work in other applications of CNN, such as image recognition. To properly test the effect of filter size all hyperparameters such as weight initialization or learning rate were held at a constant setting. The first two models used two convolutional layers with a fully connected flattening layer and SoftMax classification layer. A flattening layer simply converts a 2D matrix of values into a 1D vector by concatenating values from 2D array to a 1D column vector (Jeong, 2019). The only difference between the two models was the size of the filter in the first layer. With the first being a 100-by-24 filter and the second being a 50-by-8 filter. These were decided in order to capture the interaction between gages, recall that gages are laid out in a 8-by-3 configuration with gages 1 through 8 being in the first column and so on. The first model therefore, captures the interaction of all gages in one filter and has a step size such that it effectively only moves down the image not across with a stride in the horizontal direction equal to the length of the filter. The second model used a smaller filter or 50x8 size to try capturing more refined local interactions between gages. A summary of the first two model architectures is shown in Table 2. The third model was based on common neural network architectures which use smaller filters. The model used four convolutional layers that all had a 3 by 3 filter size which ultimately fed into a 4x1 fully connected and SoftMax classification layer. The three models were trained using 70% and 10% data for training data and validation, respectively. Through testing it was found that the two models that had the highest validation accuracy were model 1 and 3 however model 1 took significantly less amount of time to train. This being the case the overall architecture for the filter sizes was chosen based on the first model.

Table 3: Summary of CNN Architectures

| Model | First Conv Layer | Second Conv Layer | Third Layer | Output |
|---|---|---|---|---|
| 1 | Filter Size: 100x24<br>Step Size: [50 24]<br>Padding: Same | Filter Size: 2x1<br>Step Size: [1 1]<br>Padding: Same | 4x1 Fully Connected Layer | Softmax Classification with 4 classes |
| 2 | Filter Size: 50 x 8<br>Step Size: [25 8]<br>Padding: Same | Filter Size: 2x1<br>Step Size: [1 1]<br>Padding: Same | 4x1 Fully Connected Layer | Softmax Classification with 4 classes |

The remaining architecture discussed for the CNN was optimized using an iterative approach along with the Experiment Manager application in MATLAB. A diagram of the final overall architecture of the CNN can be seen on Figure 27. The optimal number of filters for the first convolutional layer and the second convolutional layer were found to be 100 and 50, respectively. Sigmoid, tanh, and ReLu functions were all tested and ReLu ultimately found to be the most effective activation function. As well between the two layers an average pooling layer with a filter size of 2x1 and stride of 1x1 was used. All parameters were optimized using the Experiment Manager where different combinations of parameters are tested using an exhaustive sweep, the results are stored, and the highest validation accuracy taken from all runs has the optimized parameters. Average pooling layer performs down sampling on the convolved image by dividing it into rectangular pooling regions and computing the average values in each region. Maximum pooling which takes the max value of the pooling region was tested as well, but consistently had a lower validation accuracy. Pooling layers reduce the number of connections to the next convolutional layer and do not perform any learning. While pooling does not learn, it helps with computational efficiency by reducing the number of parameters a subsequent layer needs to learn, as well they assist in preventing overfitting of a network. Overfitting occurs when a statistical model fits too closely to a training set and therefore cannot accurately predict on validation or test data (James et al., 2021). Another way in which overfitting is addressed in the network is by adding 50% dropout to both convolutional layers (James et al., 2021). This dropout layer effectively "turns off" 50% of the filters in each convolutional layer nullifying its contribution to prediction (James et al., 2021). Dropout layers tend to make the training process noisy by causing neurons in a layer to predict while using only half the available filters. It is also suggested that dropout may help in situations where neurons co-adapt over the training process, which is not ideal since unique filters are preferable. Lastly batch normalization layers were added between the convolutional layers and the ReLu activation layers. Figure 28 shows the final flow of the CNN, by following the black arrow in the diagram the sequence of operations that the data is passed through can be seen.

The final architecture without any model training parameters tuned was found to have a training accuracy of about 85% and a validation accuracy of about 70% it is important to note that between iterations of training these values vary slightly. These are promising results for an architecture and the remaining needed accuracy to have a reliable system is achieved through hyperparameter tuning discussed in the next section.

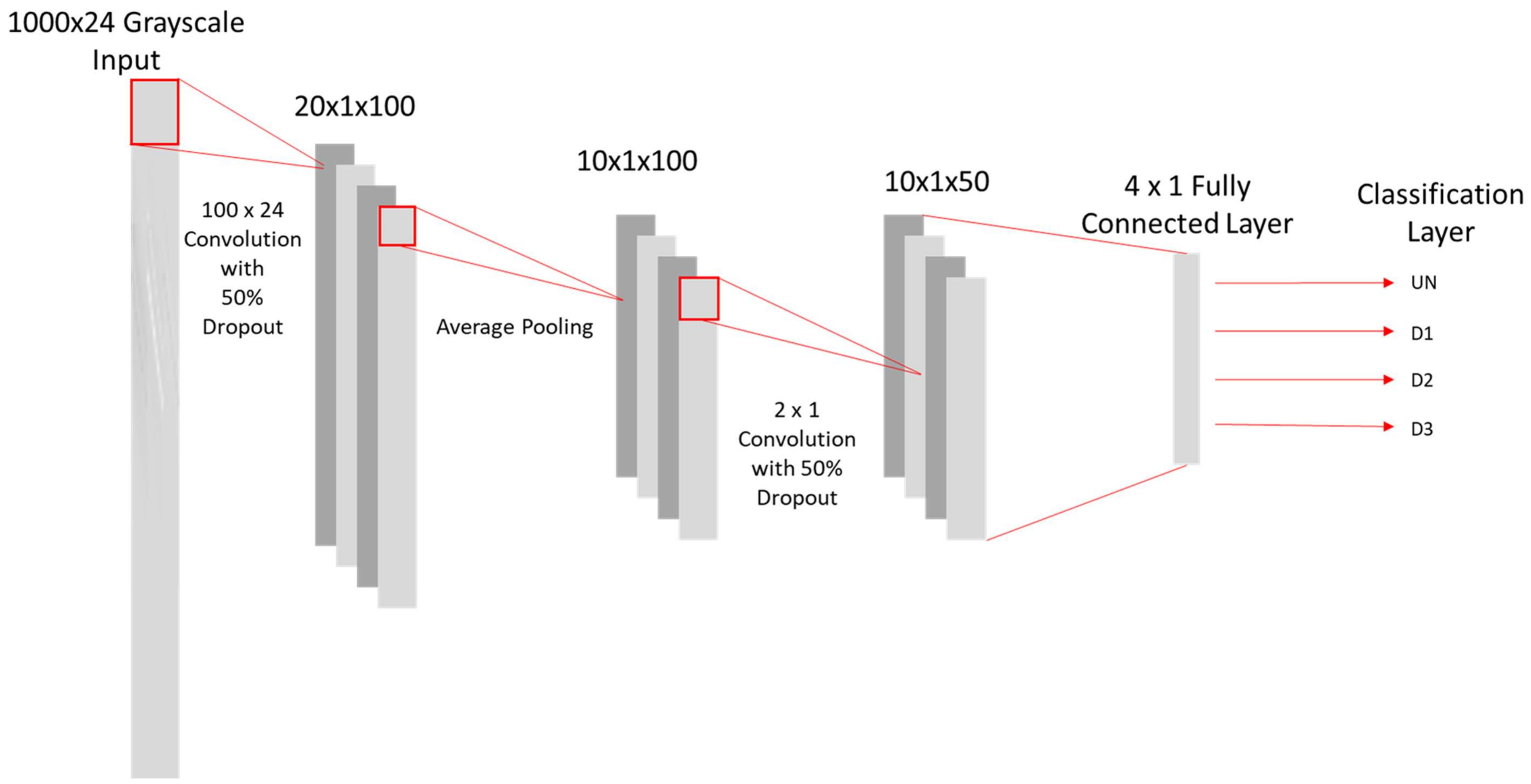

1000x24 Grayscale Input
20x1x100
100 x 24 Convolution with 50% Dropout
Average Pooling
10x1x100
2 x 1 Convolution with 50% Dropout
10x1x50
4 x 1 Fully Connected Layer
Classification Layer
UN
D1
D2
D3

Figure 27: Overall Final Network Architecture

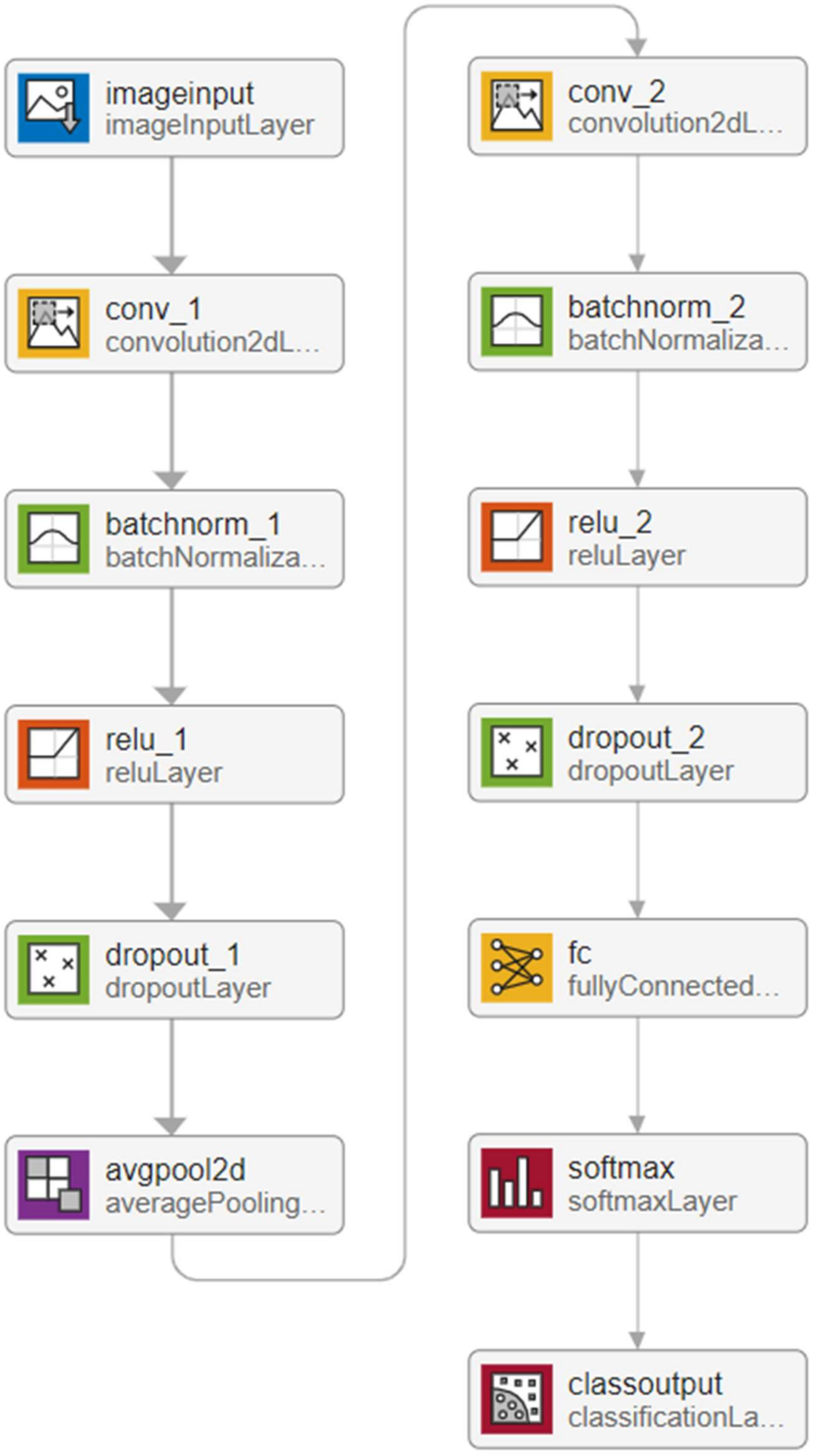


Figure 28: Detailed Final Network Architecture

## 5. 3 Model Training

Calculating the filter values, which is the learning process, in order to identify features and classify inputs based on them is done during the training phase of a network through a process called backpropagation. Backpropagation is an algorithm used in neural networks that seeks to adjust the filter values in a network in order to minimize a loss function (James et al., 2021). The loss function for a classification network is typically taken as the MSE between the probability an input belongs to a class and the actual probability (James et al., 2021). This function is defined as $R(\theta)$ (James et al., 2021);

$$R(\theta) = \frac{1}{2}\sum_{i=1}^{n}\left(y_i - f(x_i)\right)^2, \qquad \text{Eq. 12}$$

where $f(x_i)$ is the predicted likelihood that the input belongs to category $i$ and $y_i$ is the actual likelihood that the input belongs to category $i$. Backpropagation uses gradient descent to minimize this function by finding the gradient of the cost function and adjusting weights as to move in the negative direction of gradient. The gradient of the cost function is defined below (James et al., 2021);

$$\nabla R(\theta^m) = \left.\frac{\partial R(\theta)}{\partial \theta}\right|_{\theta\, =\, \theta^m}. \qquad \text{Eq. 13}$$

The gradient of the function is taken as a vector of partial derivatives at the point of interest m, where $\theta^m$ is the current guess for the weight of the function. Once gradient is found the weight, in this example $\theta^m$, is adjusted as to take a step in the negative direction of $\nabla R(\theta^m)$. So $\theta^m$ would be adjusted to a value $\theta^{m+1}$ as follows (James et al., 2021),

$$\theta^{m+1} \leftarrow \theta^m - \rho \nabla R(\theta^m) \qquad \text{Eq. 14}$$

In this equation $\rho$ represents the learning rate or the amount $\theta^m$ is adjusted to reach $\theta^{m+1}$. A more intuitive way of understanding backpropagation is by thinking of the gradient as a two-dimensional function. Figure 29 shows this by plotting $R(\theta)$ against $\theta$, in the instance of CNN training $\theta$

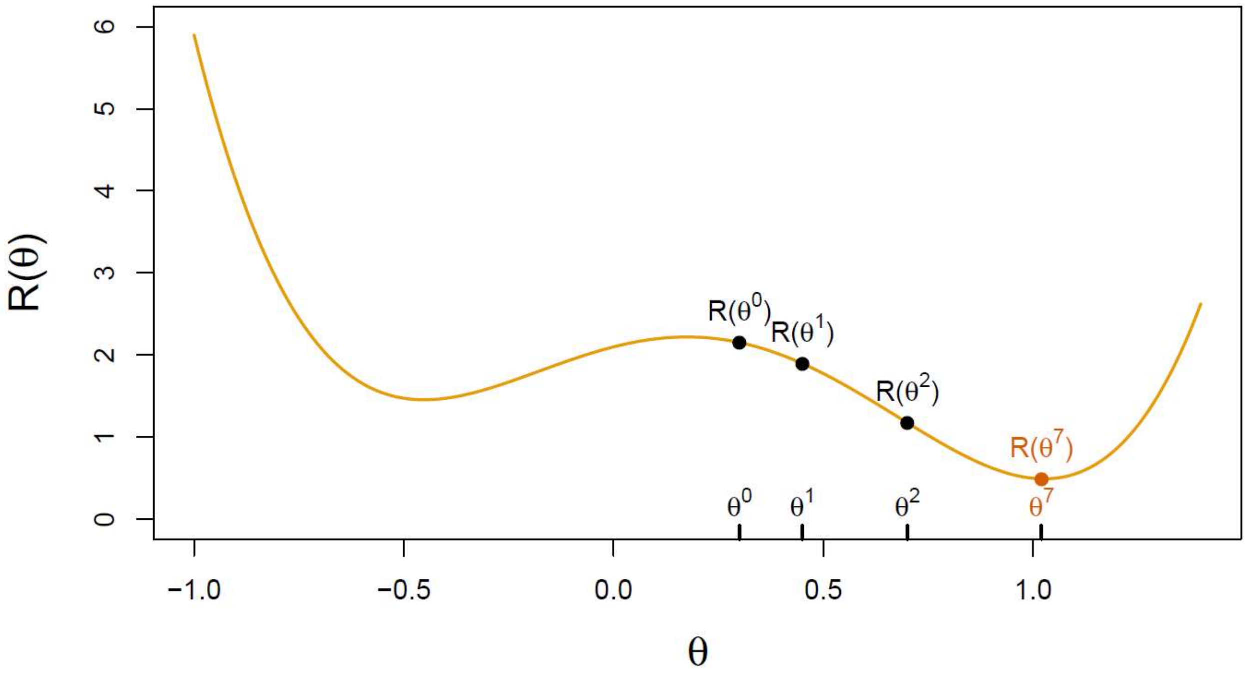


Figure 29: Cost function Example (James et al., 2021)

would be the filter weights. The orange line represents the MSE defined above, if the initial guess is $\theta^0$ the gradient is calculated at the point and a step is taken in the negative direction until a $\theta$ value that results in the lowest MSE is found, in this figure it is denoted by $\theta^7$. This process of taking steps in the negative gradient of the MSE function is known as gradient descent, the process of adjusting weights in the network using this method is called backpropagation.

This study uses a type of gradient descent commonly referred to as Adam (Kingma & Ba, 2014) to optimize weights during backpropagation that uses the same general concepts, but with added terms for momentum and learning decay factors. More information about Adam optimizer can be found in (Kingma & Ba, 2014).

The process of determining what parameters for backpropagation result in the highest validation accuracy is called hyperparameter tuning (James et al., 2021). Using the Experiment Manager in MATLAB different parameters were tested using an iterative approach and compared for effectiveness based on computational time and validation accuracy. The two most influential hyperparameters are the weight initializer and the learning rate (Sen & Ozkurt, 2020). Weight initializers determine the values used the first iteration of optimization before backpropagation process would start. It was found that the weight initializer developed by He et al., 2015 and a learning rate of 0. 006 resulted in the highest validation accuracy for this network architecture. He weight initialization calculates the values as a random number with a Gaussian probability distribution (G) with a mean of 0 and a standard deviation of $\sqrt{2/n}$, where n is the number of input into the neuron. In mathematical terms it is,

$$\omega_i = G(\,0.0\,, \sqrt{2/n}\,), \qquad \text{Eq. 15}$$

where $\omega_i$ is the random initial value for a weight, which is a component of a filter. The learning rate determines how much the weights are adjusted throughout each iteration of backpropagation during the training phase. On one hand if the learning rate would be too low, the training could take too long, and on the other hand if it would be too high, training may not converge to an optimal solution. A range of learning rates were tested with the optimal rate found to be 0.0006. The learning rate was held constant throughout training.

The remaining hyperparameters to be tuned were amount of dropout, maximum number of epochs, minimum batch size (Minibatch), and the validation frequency. Amount of dropout refers to the percentage of nodes “switched off” in dropout layers and was found to be 50%. Minibatch size refers to a subset of the training set that is used to evaluate the gradient and loss functions in order to update weights. Minibatch is commonly used since many datasets have a large amount of training samples and using all of them for backpropagation would be unreasonable. In this study a minibatch size of 128 is used. An epoch refers to one compete pass of training dataset forward and backwards through the CNN algorithm. This parameter controls how long the system will train for; 150 epochs were used for this algorithm. Validation frequency is the frequency at which the algorithm will categorize validation data. This parameter is mainly used to monitor the training process to make sure overfitting is not occurring and is done every 10 epochs in this instance. A summary of the final hyperparameters is in the table below.

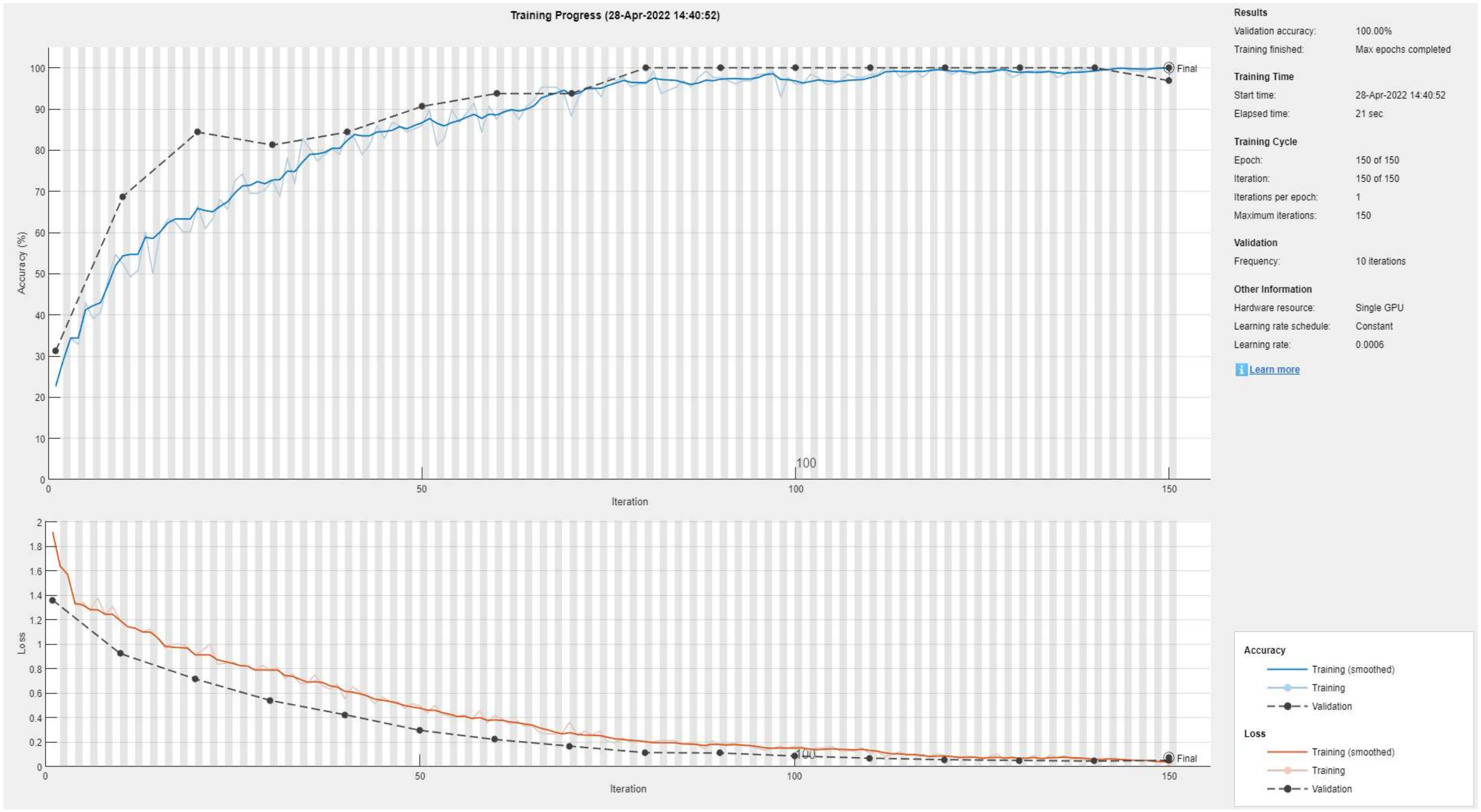
Table 4: Summary of Hyperparameters

| **Hyperparameter** | **Values** |
|---|---|
| Dropout | 50% |
| Weight Initializer | He |
| Minibatchsize | 128 |
| Epochs | 150 |
| Validation Frequency | Every 10 Epochs |
| Gradient Decay Factor | 0. 9 |
| Squared Gradient Decay Factor | 0. 999 |

## 5. 4 Final Model Training and Validation Accuracy

The final CNN architecture and hyperparameters were chosen based on the highest validation accuracy of the model over the training process. As previously discussed, the effectiveness of this model would be evaluated using two different data splits, one that utilized 70% training data and one that only used 30% training data. All tests for the actual development of the model were run using the 70% training data model.

Final training of the two different models was done and the training progress plot generated shown in Figure 30 and Figure 31 for 70% and 30% training data used, respectively. The figures show both the accuracy and loss of training and validation throughout the training process. The top plot in the figures show the accuracy of the training and validation sets throughout epochs, with each different epoch being represented by the alternating shaded of grey. The solid blue line indicates the training accuracy, and the dotted black line indicates the validation accuracy. In a model that is overfitting the validation accuracy would be much lower than the training accuracy throughout training. From both plots it is apparent that overfitting is not occurring, as validation accuracy is consistently either at or above the model training set accuracy. In this plot the orange solid line and dotted black line represent training and validation set loss, respectively.



Results
Validation accuracy: 100.00%
Training finished: Max epochs completed

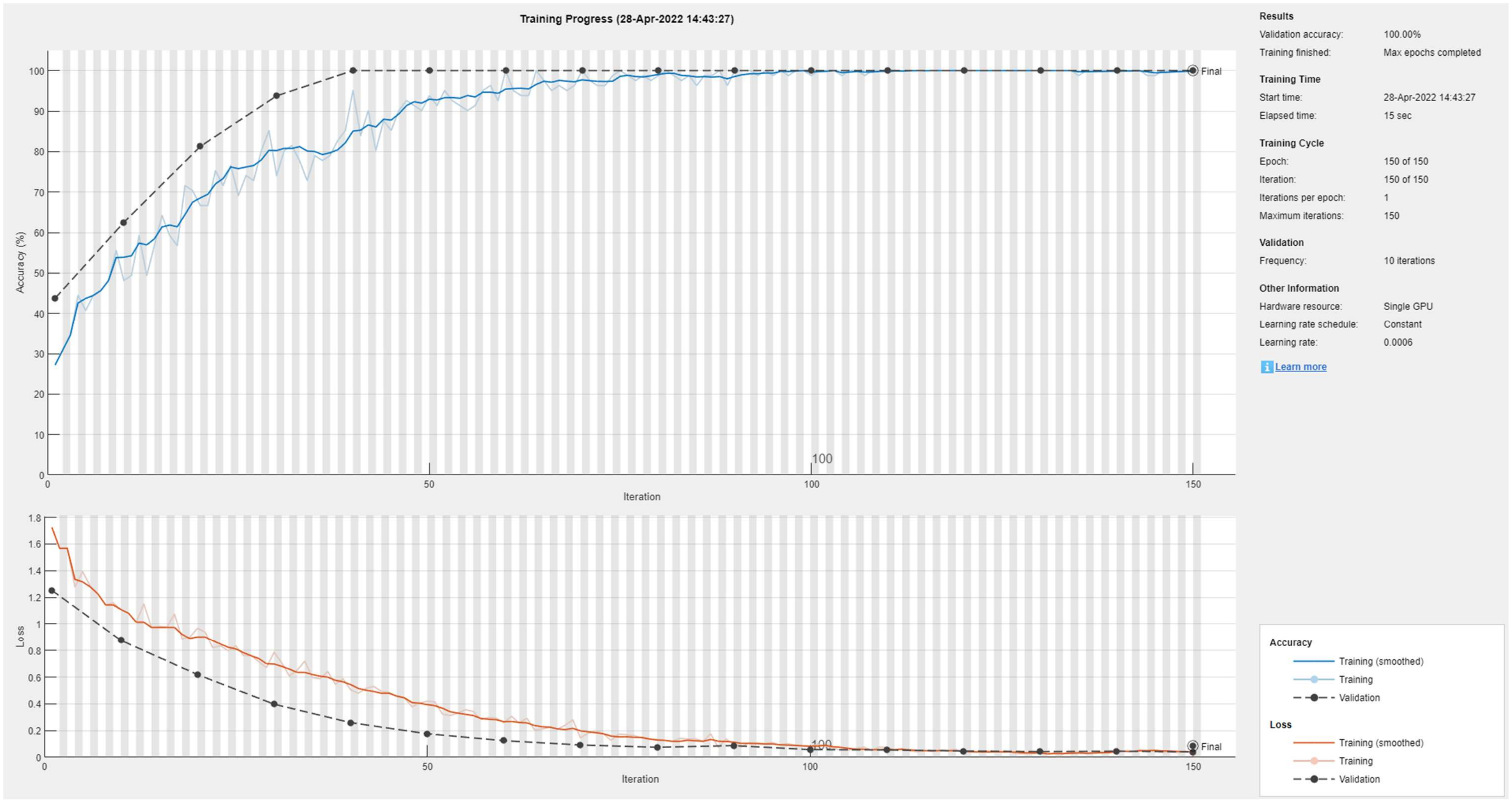
Training Time
Start time: 28-Apr-2022 14:40:52
Elapsed time: 21 sec

Training Cycle
Epoch: 150 of 150
Iteration: 150 of 150
Iterations per epoch: 1
Maximum iterations: 150

Validation
Frequency: 10 iterations

Other Information
Hardware resource: Single GPU
Learning rate schedule: Constant
Learning rate: 0.0006

Figure 30: Training Progress Plot for Model with 70% Training Data



**Results**

Validation accuracy: 100.00%
Training finished: Max epochs completed

**Training Time**

Start time: 28-Apr-2022 14:43:27
Elapsed time: 15 sec

**Training Cycle**

Epoch: 150 of 150
Iteration: 150 of 150
Iterations per epoch: 1
Maximum iterations: 150

**Validation**

Frequency: 10 iterations

**Other Information**

Hardware resource: Single GPU
Learning rate schedule: Constant
Learning rate: 0.0006

Learn more

**Accuracy**

Training (smoothed)
Training
Validation

**Loss**

Training (smoothed)
Training
Validation

Figure 31: Training Progress Plot for Model with 30% Training Data

These plots are meant to visually represent the performance of the network throughout the training process. The accuracy of the network should increase as the number of epochs increase, since at the end of each epoch the network preforms backpropagational and refines filters. At the end of training in both models the training and validation accuracy reach 100% and the total elapsed time that training took is under 30 seconds for both trials.

## 5. 5 Conclusion

The final architecture was developed based on the gage layout and indicated that an intuitive approach can be taken when developing a CNN for damage detection. This is an important distinction as designs vary between different application in SHM, meaning the filter sizes can be easily adjusted based on differing gauge configurations. As well the iterative hyperparameter tuning approach was effective for finding optimum settings.

The results of model development in this section show a promising accuracy for the model based on the validation and training accuracy. It is noteworthy that the final model accuracy is not represented by the validation accuracy but instead by the testing set accuracy which is presented in the next section. The final two models are stored after training and then implemented for prediction using a test set that has not yet been seen by the model. The reason for this is to avoid any bias in reporting of the final accuracy since the model is developed using validation accuracy as a main performance parameter.

# Chapter 6. Results and Discussion

In this chapter, the accuracy of the two models developed is reported as the accuracy of the trained network using the testing dataset. It is a common practice to not develop CNNs using testing accuracy as this can lead to bias (James et al., 2021). Instead, CNNs are developed in order to maximize the training and validation accuracy as discussed in the previous section. The accuracy of the CNNs is therefore determined after training of the network. Recall that the main goal of research was to develop a CNN to detect damage with up to 95% accuracy with as minimum training data used as possible.

## 6. 1 Resulting Filters

The final CNNs had a total of 252,454 learnable parameters or weights. These weights were applied to the network in the form of filters that created a convolved image which indicated if certain features were present in the image or not. To visualize what areas were activated in an image when it was passed through the first convolutional layer, an example original image is shown in Figure 32. This original image once passed through the first convolutional layer was transformed into a number of different images that represented different features present. A number of these convolved images can be seen in Figure 33. This figure is a layout of the 16 out of the 100 different convolved images that resulted from the filters that were applied in the first layer. It is important to note that these images do not need to be high resolution or large in order to be useful; in fact, the dimension of these convolved images after the first layer is 20x1x1, which is significantly smaller than the original image.

Figure 32: Original Example Image before convolutional layer

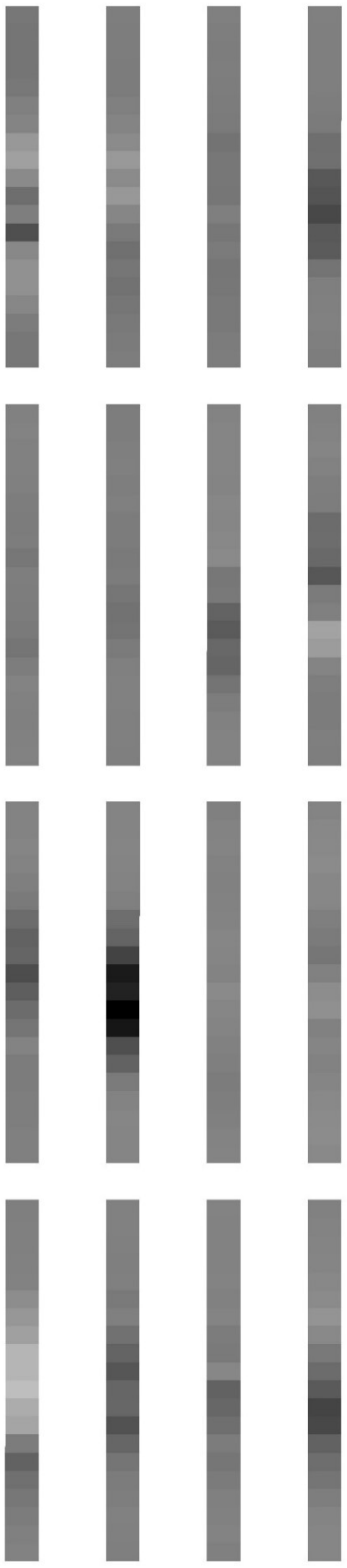


Figure 33: Activations of Image after first convolutional Layer

**6. 2 Testing Accuracy**

As previously mentioned, the final accuracy of the network was determined based on its accuracy in categorizing test data. Testing data consisted of images that the network had not used or seen at all during the training phase. By only using test data for final model accuracy and not during the development of architecture and hyperparameter tuning the amount of bias in deciding on a final model is reduced.

The first model trained with 70% training data had a total of 64 images available in the testing set, Figure 34 shows the resulting confusion matrix a of predicted categories for test images. A

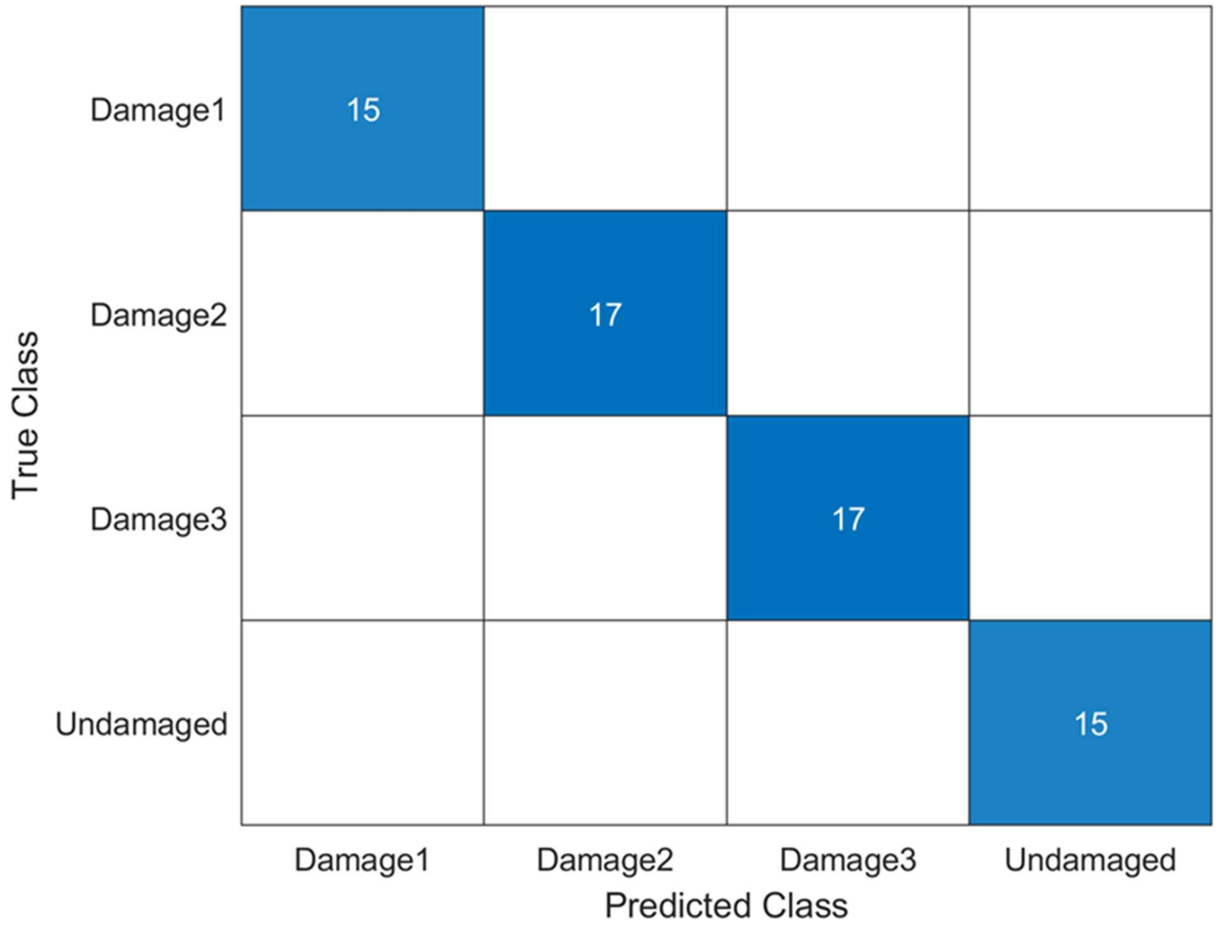


Figure 34: 70% Training Data Confusion matrix

confusion matrix is a common way of visualizing what class was predicted and what the actual true class is. The predicted classes of the network are along the bottom of the matrix and the true classes are along the side. The number inside the box indicates the number of images corresponding to the respective predicted and true class. A correct categorization with the CNN therefore puts the image along the diagonal square of the matrix. A correct classification is further identified by the color of the squares being blue. Figure show the confusion matrix for the network using 70% training data. As seen the network achieved 100% accuracy when predicting on testing damage. This of course is a very good accuracy of the model and indicated that a high level of accuracy can be achieved with sufficient data. The confusion matrix for the model that was trained

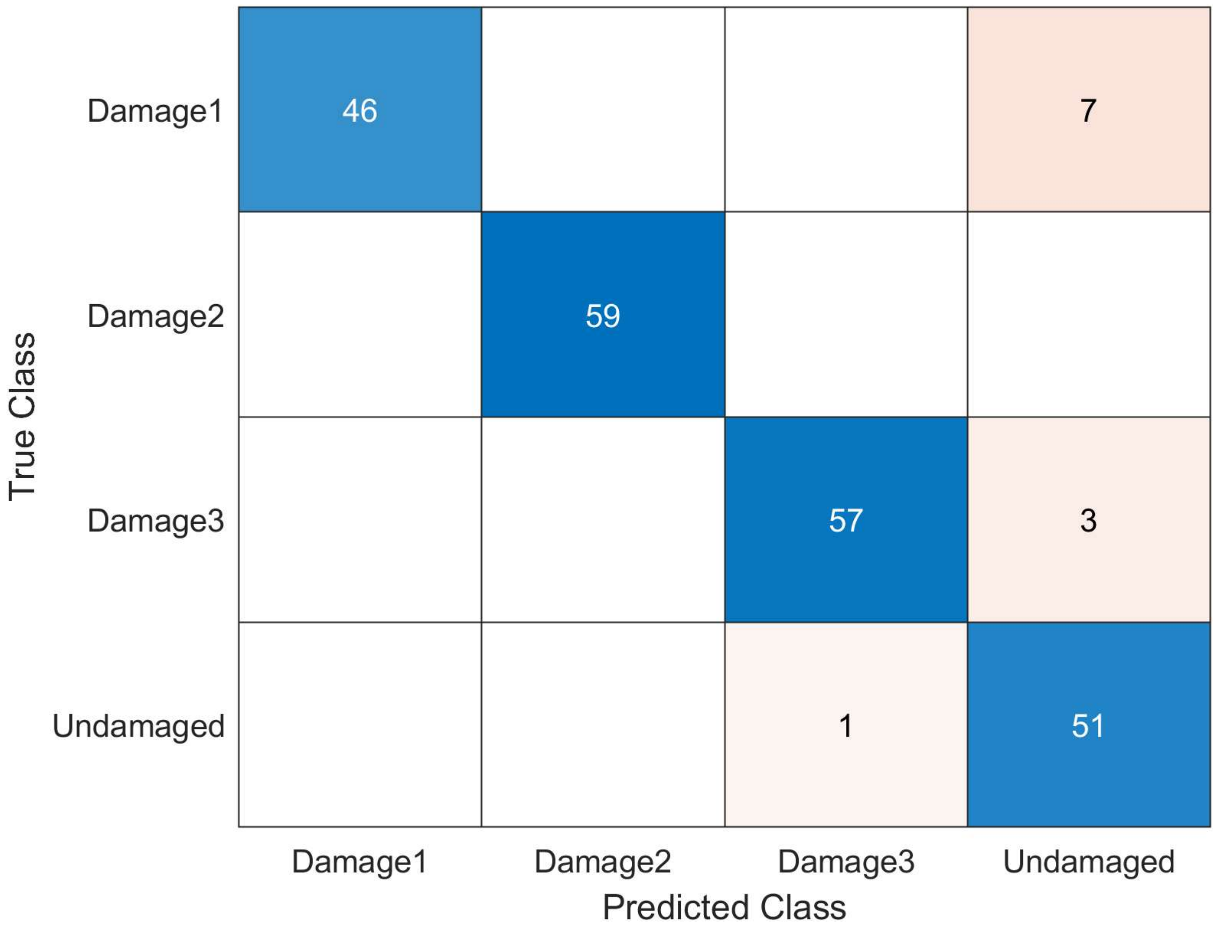


Figure 35: 30% Training Data Confusion matrix

with 30% training data is shown below in Figure 35. This model had a total of 224 test images and only categorized 11 images incorrectly, which resulted in a 95% accuracy. The red boxes indicate the incorrectly categorized image. The most images categorized incorrectly was between the CNN thinking an image is Undamaged, but it true class being Damage 1. This frequency of misclassification is consistent with observed damage scenarios as Damage 1 is the closest to the undamaged scenario and corresponds to damage conditions that are invisible when using traditional visual inspection techniques. The difference in accuracy of the two models is expected as the 70% training data model has much more data to train with. This is most likely caused by the filters in this model being much more refined and detailed because it had more data to adjust weights on. While the accuracy of the 30% training model is lower it still achieved a final testing accuracy of 95% and only took a total of 15 seconds to train, indicating that robust damage detection can be efficiently achieved using CNNs with minimal training data.

To further test the robustness of the network two more variations of data were used. The first was original data with added 15% white Gaussian noise. The signal used to convert into images had noise added using the awgn() function in MATLAB, which ultimately added noise by implementing a 15% noise to signal ratio. An example of the original signal and signal with added noise can be seen on Figure 37 and Figure 36 respectively. By adding noise, the ability of the model to detect damage when gage measurement is not ideal can be tested. The model was trained using 70% and noisy images and still achieved a testing accuracy of 98% with only 1 misclassified image. The confusion matrix is shown in Figure 39.

Another way to test the network was by reducing the amount of sensors used for damage detection. To do this only 4 sensors were used, sensor 1, 6, 12, and 24 were used in this instance. These sensors were chosen arbitrarily. Since the number of sensors changed the size of the image to represent them became 1000 by 4 by 1 as opposed to the original image that had a width of 24. This being the case the dimensions of the first convolutional layer also had to change, however since the architecture was based on the number of sensors it was straightforward. The width and step size of the first convolutional layer were simply changed from 24 to 4 for both. The final testing accuracy of the network with only 4 sensor reading used was 93% and the confusion matrix can be seen on Figure 38.

The final architecture and hyperparameter tuning resulted in a CNN that can effectively categorize data as damaged or undamaged. With the proper amount of training data, the network was able to detect damage with 100% accuracy with no noise and the full range of sensors used. The robustness of the network was further tested by decreasing the amount of training data, adding 15% Gaussian white noise to data and decreasing the amount of sensor data used as inputs into the network. Throughout all variations of data, the network was able to achieve above 90% accuracy.

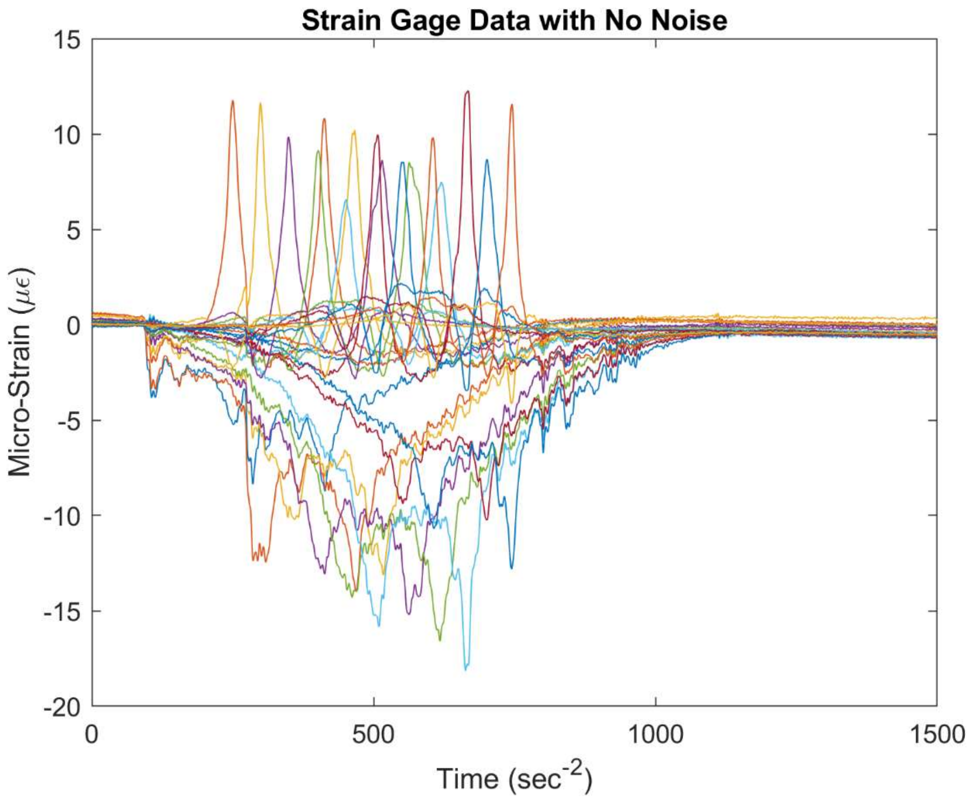


Figure 37: Strain Gage Data without added Noise

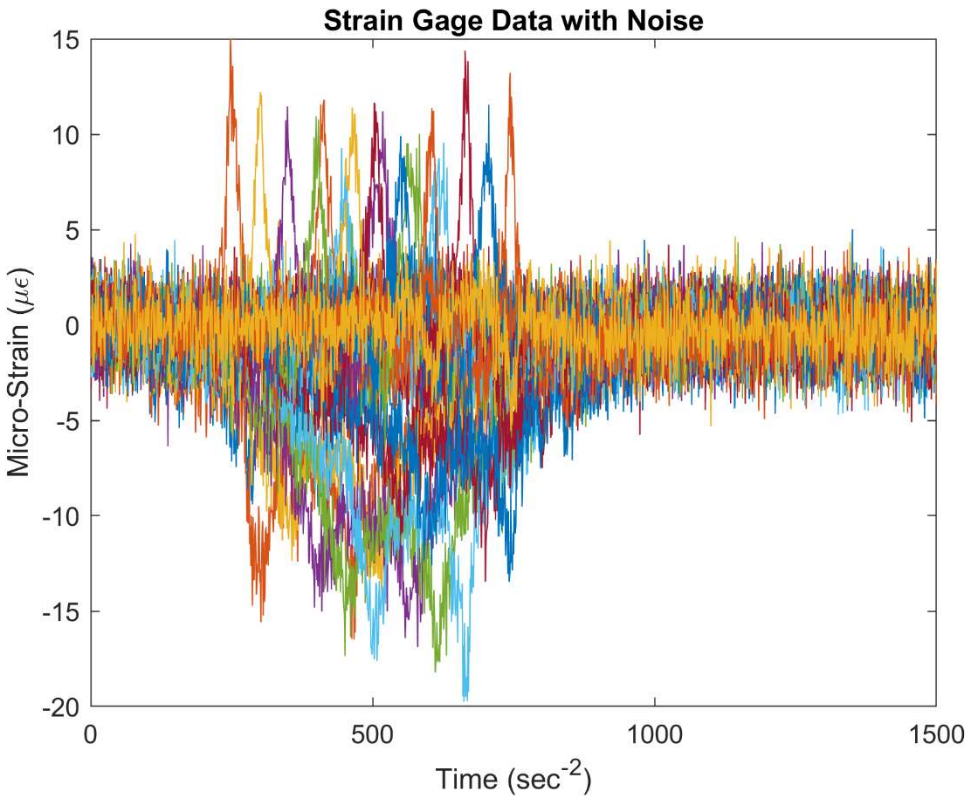


Figure 36: Strain Gage Data with 15% added Noise

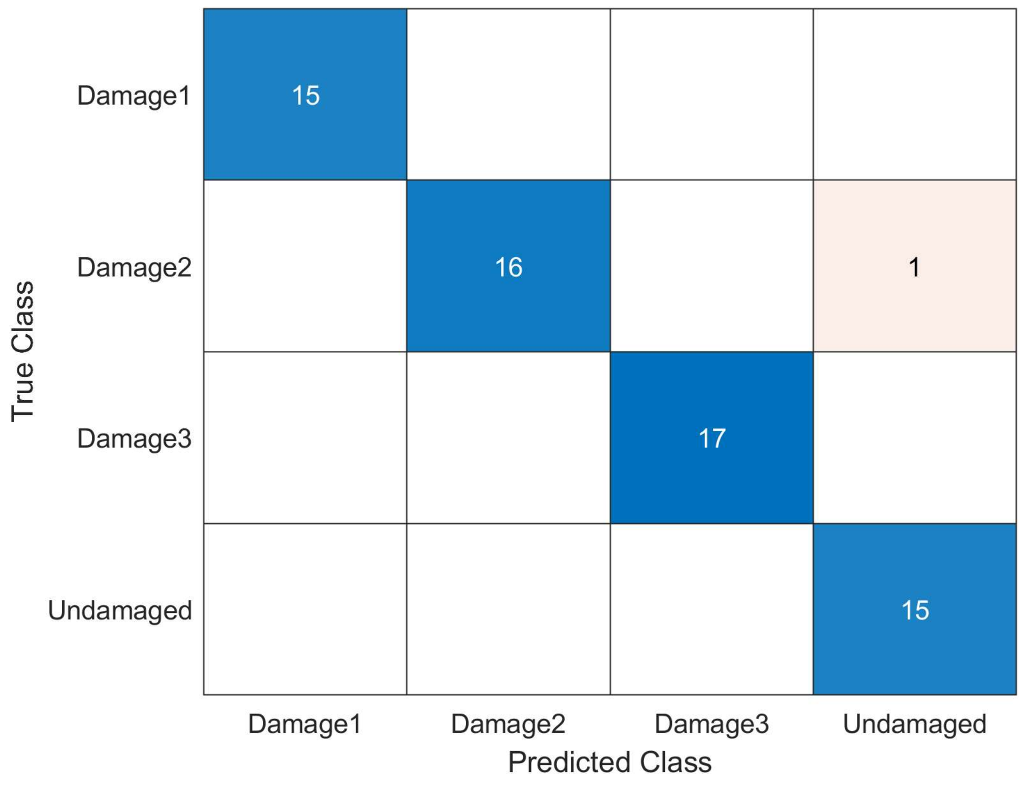


Figure 39: 15% Added Noise Confusion Matrix

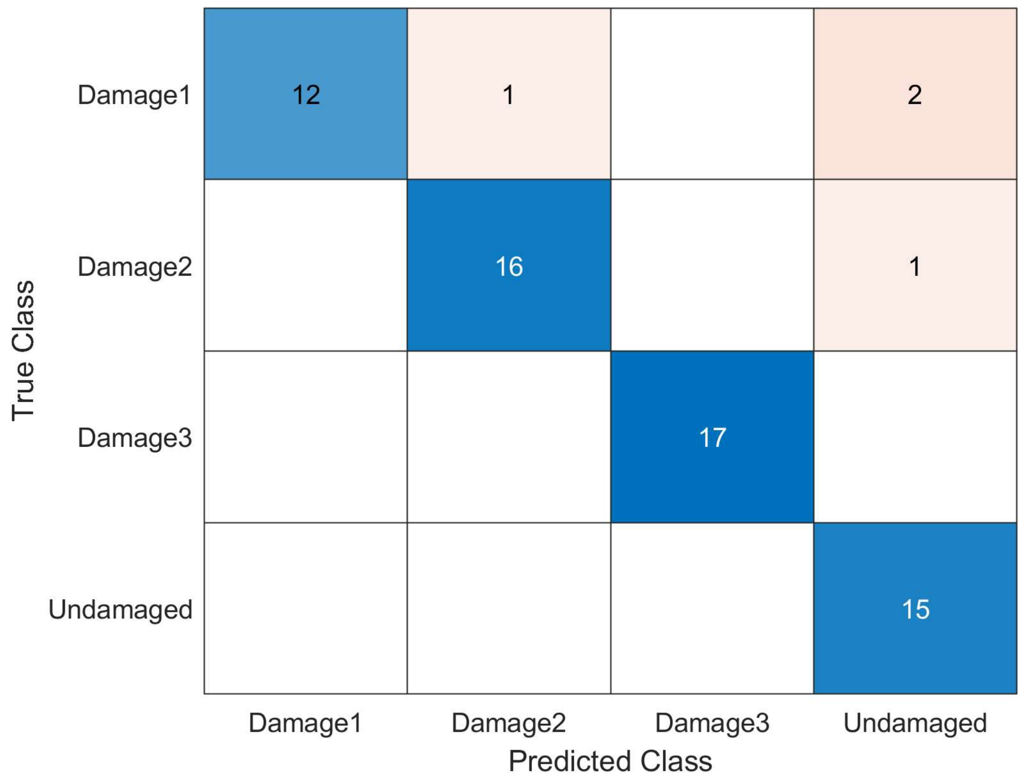


Figure 38: Reduced Sensor Confusion Matrix

# Chapter 7. Conclusion

## 7. 1 Summary of Work

ML algorithms have shown the potential to detect damage using instrumentation data from SHM systems with great accuracy. While many studies have furthered the development of ML algorithms there are some gaps in literature that hinder the efficacy of use in industry. Mainly the lack of full-scale experiments, lack of moving and variation loading, over complexity of data processing, and inability for methods to detect invisible damage. This study has shown that by using a 2D CNN a robust method for detecting damage under variational loading can be achieved with minimal data processing required.

The data used for testing the effectiveness of CNNs to detect damage consisted of strain gage data from 24 resistance-based gages. The gages were installed on a full-scale concrete bridge that was subject to various weighted vehicles that moved across the bridge in order to gather data. In this way the gathered data reflects the realistic responses that would be seen from an in-service bridge. During testing the bridge was also subject to different levels of damage varying from conditions that would be undetectable using classic visual inspection to highly damaged conditions, in order to test the CNNs ability to detect invisible damage. To reduce the required data processing time a method of representing strain gage data as an image was developed. This method could be executed in a simple MATLAB code and resulted in an input to the CNN that allowed for all gage reading for an individual run to be analyzed at once. Final input to network was a 2D greyscale image of dimensions 1000 by 24 by 1 where the time step, gage number, and strain value were represented as image height, width, and pixel value respectively. The total amount of images available was 321 that were divided into training, validation, and testing data.

The final CNN architecture and hyperparameters were determined and optimizes using the training and validation data. This was done as to avoid adding bias during the development of network. The architecture of the network was determined using an intuitive approach where the filter size and stride were determined based off gage location. The hyperparameters were tuned after architecture was finalized and optimized as to have the highest validation and testing accuracy possible.

The final method was tested in multiple ways as to determine how robust the model was. Under normal circumstances using 70% training data the model achieved an accuracy of 100%. With minimal trained data of 30% the model still achieved an accuracy of 95%. This indicated the model had the ability to be effectively trained using minimal data. The robustness was further tested by running an experiment where 15% white Gaussian noise was added to data and the amount of sensor data used for input was reduced to only 4. The accuracy for these cases was 98% and 93% respectively. This indicated that the method provided a robust approach to damage detection that could be implemented even if conditions were not ideal during data acquisition.

The final results show that the use of a 2D CNN with strain gage data offers a robust damage detection framework. The method was also able to detect damage present in the structure that would otherwise go unnoticed.  With the robustness of the model tested this approach has the potential to be implemented in practice, where idealized cases of data are not often available.

## 7. 2 Future Work

The results of this study indicated that a supervised approach for in-service structure using CNNs offers a promising damage detection framework. However, the use of a supervised approach has some drawbacks particularly the need to have labeled data for training. This is a drawback because in industry damaged data sets are not common due to safety concerns and limitations in resources. Two remedies are suggested, one is unsupervised ML and the other is Transfer Learning (TL).

In an unsupervised approach the need for damaged data for training is not necessary for this reason there has been a lot of attention around developing unsupervised approaches for damaged detection. One particular approach of interest is to use a Convolutional Autoencoder (CAE) to detect damage. The approach would utilize the same data set represented as an image but would seek to instead compress the data into a smaller dimension and then reconstruct the original image. The method is typically used in image denoising but can be applied in anomaly detection as well. The approach would be to train the network to reconstruct only undamaged images. Then after training, the network would try to reconstruct damaged images. In this way the reconstruction loss, which is taken as the MSE between original image and reconstructed image, can be found for the undamaged and damaged cased. If successful the network would have a higher MSE for damaged case, as well the different level of MSE may represent different levels of damage. This approach requires a considerably more complex architecture due to the need for not only data encoding but decoding. If implemented the use of these methods in industry would be much more likely since the need for labeled training data would not be necessary.

The drawback of unsupervised ML is that it only provides insights on existence of damage, meaning in some applications the extent of damage is not known. The extent, and characterization still would require a supervised approach. To this end, another field of research would be using TL to transfer the labels from a FE model to the real world. TL is unique in that it allows for what one algorithm learned in a different application during training to be applied to a new application. An example would be using the trained filters in the CNN discussed in this paper to detect damage from sensors on a similar structure. In this way the results from one study or application can be used multiple times. Cosliga et al., 2022 proposed a way of using TL depending on the similarity of structures between one another. The idea is to overcome the obstacle of limited data by allowing data to be shared between similar structures. Another application by Zheng et al., 2022 used TL along with a Bayesian model to produce a more accurate damage detection framework. The study showed that through the use of TL simulated data could be used to accurately train an algorithm and account for uncertainties between the responses seen in practice and simulated data. The advantage of TL is that it could utilize supervised results and provide a refined categorization of damage. For more information on future research see S. Ardani et al., 2022 and E. Akintunde et al., 2022.

The use of unsupervised ML and TL provide unique solutions for the lack of damaged data available for the development of damage detection frameworks. Through this application the efficacy of using ML in practice would be much more viable in that it would not be limited by available damaged data. The TL approach would be able to detect damage and level of damage as opposed to the unsupervised approach which may only being able to detect the presence of damage. Future work based on this paper would seek to not only to implement an unsupervised approach but to also research how TL can be used across multiple similar structures.

# REFERENCES

Abdeljaber, O., Avci, O., Kiranyaz, M. S., Boashash, B., Sodano, H., & Inman, D. J. (2018). 1-D CNNs for structural damage detection: Verification on a structural health monitoring benchmark data. *Neurocomputing*, *275*, 1308–1317. https://doi.org/10.1016/j.neucom.2017.09.069

Abdeljaber, O., Avci, O., Kiranyaz, S., Gabbouj, M., & Inman, D. J. (2017). Real-time vibration-based structural damage detection using one-dimensional convolutional neural networks. *Journal of Sound and Vibration*, *388*, 154–170. https://doi.org/10.1016/j.jsv.2016.10.043

Akintunde, E., Eftekhar Azam, S., Rageh, A., Ardani, S., & Linzell, D. (n.d.). *Damage Detection in Bridges using a Singular Value Decomposition based Novelty Index*.

Akintunde, E., Eftekhar Azam, S., Rageh, A., & Linzell, D. G. (2021). Unsupervised Machine Learning for Robust Bridge Damage Detection: Full-Scale Experimental Validation. *Engineering Structures*, *249*, 113250. https://doi.org/10.1016/j.engstruct.2021.113250

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, *8*(1), 53. https://doi.org/10.1186/s40537-021-00444-8

Ardani, S., Eftekhar Azam, S., Akintunde, E., & Linzell, D. (n.d.). *Application of Proper Orthogonal Decomposition to bridge damage detection—Field investigations*.

ASCE. (n.d.). *ASCE's 2021 report card marks the nation's infrastructure progress*. Retrieved May 23, 2022, from https://www.asce.org/publications-and-news/civil-engineering-source/civil-engineering-magazine/issues/magazine-issue/article/2021/03/asce-2021-report-card-marks-the-nations-infrastructure-progress

Azimi, M., Eslamlou, A., & Pekcan, G. (2020). Data-Driven Structural Health Monitoring and Damage Detection through Deep Learning: State-of-the-Art Review. *Sensors*, *20*(10), 2778. https://doi.org/10.3390/s20102778

Bao, Y., Tang, Z., Li, H., & Zhang, Y. (2019). Computer vision and deep learning–based data anomaly detection method for structural health monitoring. *Structural Health Monitoring*, *18*(2), 401–421. https://doi.org/10.1177/1475921718757405

Feng, C., Zhang, H., Wang, S., Li, Y., Wang, H., & Yan, F. (2019). Structural Damage Detection using Deep Convolutional Neural Network and Transfer Learning. *KSCE Journal of Civil Engineering*, *23*(10), 4493–4502. https://doi.org/10.1007/s12205-019-0437-z

FHWA. (n.d.). *Status of the Nation's Highways, Bridges, and Transit: Conditions and Performance, 24th Edition*. https://doi.org/10.21949/1521794

FHWA. (2020). *Bureau of Transportation Statistics U.S. Vehicle-Miles*. https://www.bts.gov/content/us-vehicle-miles

Figueiredo, E., Moldovan, I., Santos, A., Campos, P., & Costa, J. C. W. A. (2019). Finite Element–Based Machine-Learning Approach to Detect Damage in Bridges under Operational and Environmental Variations. *Journal of Bridge Engineering*, *24*(7), 04019061. https://doi.org/10.1061/(ASCE)BE.1943-5592.0001432

Gosliga, J., Hester, D., Worden, K., & Bunce, A. (2022). On Population-based structural health monitoring for bridges. *Mechanical Systems and Signal Processing*, *173*, 108919. https://doi.org/10.1016/j.ymssp.2022.108919

Gulgec, N., Takáč, M., & Pakzad, S. (2017). *Structural Damage Detection Using Convolutional Neural Networks* (pp. 331–337). https://doi.org/10.1007/978-3-319-54858-6_33

Hakim, S. J. S., Abdul Razak, H., & Ravanfar, S. A. (2015). Fault diagnosis on beam-like structures from modal parameters using artificial neural networks. *Measurement*, *76*, 45–61. https://doi.org/10.1016/j.measurement.2015.08.021

*Happy 50th Anniversary—National Bridge Inspection Standards—General Highway History—Highway History—Federal Highway Administration*. (n.d.). Retrieved May 23, 2022, from https://www.fhwa.dot.gov/highwayhistory/national_bridge_inspection_standards.cfm

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. https://doi.org/10.48550/ARXIV.1502.01852

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An Introduction to Statistical Learning: With Applications in R*. Springer US. https://doi.org/10.1007/978-1-0716-1418-1

Jeong, J. (2019, July 17). *The Most Intuitive and Easiest Guide for CNN*. Medium. https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480

Kaspar, K., Santini-Bell, E., Petrik, M., & Sanayei, M. (2020). Comparison between a Linear Regression and an Artificial Neural Network Model to Detect and Localize Damage in the Powder

Mill Bridge. *Transportation Research Record: Journal of the Transportation Research Board*, *2674*(8), 394–404. https://doi.org/10.1177/0361198120920631

Khodabandehlou, H., Pekcan, G., & Fadali, M. S. (2019). Vibration-based structural condition assessment using convolution neural networks. *Structural Control and Health Monitoring*, *26*(2), e2308. https://doi.org/10.1002/stc.2308

Kingma, D. P., & Ba, J. (2014). *Adam: A Method for Stochastic Optimization*. https://doi.org/10.48550/ARXIV.1412.6980

Kiranyaz, S., Avci, O., Abdeljaber, O., Ince, T., Gabbouj, M., & Inman, D. J. (2021). 1D convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing*, *151*, 107398. https://doi.org/10.1016/j.ymssp.2020.107398

Krauss, M. (2022, January 31). What Pittsburgh's Fern Hollow Bridge collapse can tell us about Pennsylvania bridges. *WITF*. https://www.witf.org/2022/01/31/what-pittsburghs-fern-hollow-bridge-collapse-can-tell-us-about-pennsylvania-bridges/

Lazo, L. (2022). D.C. to replace collapsed pedestrian bridge over Route 295, Bowser says. *Washington Post*. https://www.washingtonpost.com/transportation/2021/07/09/dc-pedestrian-bridge-collapse/

Madhavan, S. (2021). *Introduction to convolutional neural networks*. IBM Developer. https://developer.ibm.com/articles/introduction-to-convolutional-neural-networks/

Malekloo, A., Ozer, E., AlHamaydeh, M., & Girolami, M. (2021). Machine learning and structural health monitoring overview with emerging technology and high-dimensional data source highlights. *Structural Health Monitoring*, 147592172110368. https://doi.org/10.1177/14759217211036880

Markou, M., & Singh, S. (2003). Novelty detection: A review—part 2: *Signal Processing*, *83*(12), 2499–2521. https://doi.org/10.1016/j.sigpro.2003.07.019

Pathirage, C. S. N., Li, J., Li, L., Hao, H., Liu, W., & Ni, P. (2018). Structural damage identification based on autoencoder neural networks and deep learning. *Engineering Structures*, *172*, 13–28. https://doi.org/10.1016/j.engstruct.2018.05.109

Phares, B. M., Washer, G. A., Rolander, D. D., Graybeal, B. A., & Moore, M. (2004). Routine Highway Bridge Inspection Condition Documentation Accuracy and Reliability. *Journal of Bridge Engineering*, *9*(4), 403–413. https://doi.org/10.1061/(ASCE)1084-0702(2004)9:4(403)

Rastin, Z., Ghodrati Amiri, G., & Darvishan, E. (2021). Unsupervised Structural Damage Detection Technique Based on a Deep Convolutional Autoencoder. *Shock and Vibration*, *2021*, 1–11. https://doi.org/10.1155/2021/6658575

Sen, S. Y., & Ozkurt, N. (2020). Convolutional Neural Network Hyperparameter Tuning with Adam Optimizer for ECG Classification. *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*, 1–6. https://doi.org/10.1109/ASYU50717.2020.9259896

*ST350 Strain Transducer—BDI Structural Testing & Monitoring Systems*. (n.d.). Retrieved June 13, 2022, from https://bditest.com/product/st350-strain-transducer/

*Strain Gauge and Wheatstone Bridge—MATLAB & Simulink*. (n.d.). Retrieved June 13, 2022, from https://www.mathworks.com/help/physmod/sps/ug/strain-gauge-and-wheatstone-bridge.html#responsive_offcanvas

Teng, S., Chen, G., Gong, P., Liu, G., & Cui, F. (2020). Structural damage detection using convolutional neural networks combining strain energy and dynamic response. *Meccanica*, *55*(4), 945–959. https://doi.org/10.1007/s11012-019-01052-w

Teng, Z., Teng, S., Zhang, J., Chen, G., & Cui, F. (2020). Structural Damage Detection Based on Real-Time Vibration Signal and Convolutional Neural Network. *Applied Sciences*, *10*(14), 4720. https://doi.org/10.3390/app10144720

TNDOT. (2021). *Interstate 40 Hernando DeSoto Bridge*. https://www.tn.gov/tdot/projects/region-4/i-40-hernando-desoto-bridge.html

Vagnoli, M., Remenyte-Prescott, R., & Andrews, J. (2018). Railway bridge structural health monitoring and fault detection: State-of-the-art methods and future challenges. *Structural Health Monitoring*, *17*(4), 971–1007. https://doi.org/10.1177/1475921717721137

Weinstein, J. C., Sanayei, M., & Brenner, B. R. (2018). Bridge Damage Identification Using Artificial Neural Networks. *Journal of Bridge Engineering*, *23*(11), 04018084. https://doi.org/10.1061/(ASCE)BE.1943-5592.0001302

Yang, Q., Shen, D., Du, W., & Li, W. (2021). A Deep Learning-Based Framework for Damage Detection With Time Series. *IEEE Access*, *9*, 66570–66586. https://doi.org/10.1109/ACCESS.2021.3076436

Zhang, Z., Sun, C., & Guo, B. (2022). Transfer-learning guided Bayesian model updating for damage identification considering modeling uncertainty. *Mechanical Systems and Signal Processing*, *166*, 108426. https://doi.org/10.1016/j.ymssp.2021.108426

Zhao, R., Yan, R., Chen, Z., Mao, K., Wang, P., & Gao, R. X. (2019). Deep learning and its applications to machine health monitoring. *Mechanical Systems and Signal Processing*, *115*, 213–237. https://doi.org/10.1016/j.ymssp.2018.05.050

Zhong, K., Teng, S., Liu, G., Chen, G., & Cui, F. (2020). Structural Damage Features Extracted by Convolutional Neural Networks from Mode Shapes. *Applied Sciences*, *10*(12), 4247. https://doi.org/10.3390/app10124247

Zhou, X., Di, J., & Tu, X. (2019). Investigation of collapse of Florida International University (FIU) pedestrian bridge. *Engineering Structures*, *200*, 109733. https://doi.org/10.1016/j.engstruct.2019.109733

Zhu, Y., Brettin, T., Xia, F., Partin, A., Shukla, M., Yoo, H., Evrard, Y. A., Doroshow, J. H., & Stevens, R. L. (2021). Converting tabular data into images for deep learning with convolutional neural networks. *Scientific Reports*, *11*(1), 11325. https://doi.org/10.1038/s41598-021-90923-y

# Appendix A: Data Conversion MATLAB Script

```
%% This script is to save all snapshot matrices into pictures. These pictures
will be gray scale and
%This script is to save all snapshot matrices into pictures. These pictures
%will be gray scale and have a dimension mxn where n is the number of
%sensors and m is the smallest series length.

% Load Data
clc;clear;
load('F:\Research\StrainGageCNN\SnapshotData\Data.mat');
names = [D1_DTE, D1_DTF ,D1_SM05, D1_SM10,D1_SM15,...
    D2_DTE, D2_DTF ,D2_SM05, D2_SM10, D2_SM15,...
    D3_DTE, D3_DTF ,D3_SM05, D3_SM10, D3_SM15,...
    UN_DTE, UN_DTF ,UN_SM05, UN_SM10, UN_SM15];
%%
SeriesLengths =
[length(D1_DTE{1,1}),length(D1_DTF{1,1}),length(D1_SM05{1,1}),length(D1_SM10{
1,1}),length(D1_SM15{1,1}), ...

length(D2_DTE{1,1}),length(D2_DTF{1,1}),length(D2_SM05{1,1}),length(D2_SM10{1
,1}),length(D2_SM15{1,1}), ...

length(D3_DTE{1,1}),length(D3_DTF{1,1}),length(D3_SM05{1,1}),length(D3_SM10{1
,1}),length(D3_SM15{1,1}), ...

length(UN_DTE{1,1}),length(UN_DTF{1,1}),length(UN_SM05{1,1}),length(UN_SM10{1
,1}),length(UN_SM15{1,1})];

MinSeriesLength = min(SeriesLengths);
MaxSeriesLength = max(SeriesLengths);
%Typically image data is resized to the minimum on so in our case all
%images should be 1000x24

%%
%Base code for making image and saving it
for i = 1:length(names)
    maxVal(1,i) = max(names{1,i},[],"all");
    minVal(1,i) = min(names{1,i},[],"all");
end
maxAll = max(maxVal,[],"all");
minAll = min(minVal,[],"all");

%%
%normData = normalize(D1_DTE{1,2},'zscore')
imwrite(normData,'Testimage.png','png','Transparency',.5);
ExImage = imread("Testimage.png");
figure(3);imshow('Testimage.png');
figure(1);surf(ExImage)
figure(2);surf(normData)
%%
figure()
plot(D1_DTE{1,1})
adnoise = awgn(D1_DTE{1,1},15,'measured');
figure()
plot(adnoise)
%%
for i = 1:length(UN_DTE)
    normData = (UN_DTE{1,i} - minAll)/(maxAll - minAll);
    itext = num2str(i);
    imwrite(normData,['UN_DTE_' itext '.png']);
    for i2 = 1:length(UN_DTE)

        Image = imread(["UN_DTE_"+itext+".png"]);
        ResizedImage = imresize(Image,[1000 24]);
        imwrite(ResizedImage,['UN_DTE_' itext '.png']);
    end
end

%%
for i = 1:length(UN_DTF)
    normData = (UN_DTF{1,i} - minAll)/(maxAll - minAll);
    itext = num2str(i);
    imwrite(normData,['UN_DTF_' itext '.png']);
    for i2 = 1:length(UN_DTF);

        Image = imread(["UN_DTF_"+itext+".png"]);
        ResizedImage = imresize(Image,[1000 24]);
        imwrite(ResizedImage,['UN_DTF_' itext '.png']);
    end
end
%%
for i = 1:length(UN_SM05)
    normData = (UN_SM05{1,i} - minAll)/(maxAll - minAll);
    itext = num2str(i);
    imwrite(normData,['UN_SM05_' itext '.png']);
    for i2 = 1:length(UN_SM05);

        Image = imread(["UN_SM05_"+itext+".png"]);
        ResizedImage = imresize(Image,[1000 24]);
         imwrite(ResizedImage,['UN_SM05_' itext '.png']);
    end
end
%%
for i = 1:length(UN_SM10)
    normData = (UN_SM10{1,i} - minAll)/(maxAll - minAll);
    itext = num2str(i);
    imwrite(normData,['UN_SM10_' itext '.png']);
    for i2 = 1:length(UN_SM10);

        Image = imread(["UN_SM10_"+itext+".png"]);
        ResizedImage = imresize(Image,[1000 24]);
         imwrite(ResizedImage,['UN_SM10_' itext '.png']);
    end
end
%%
for i = 1:length(UN_SM15)
    normData = (UN_SM15{1,i} - minAll)/(maxAll - minAll);
    itext = num2str(i);
    imwrite(normData,['UN_SM15_' itext '.png']);
    for i2 = 1:length(UN_SM15);

        Image = imread(["UN_SM15_"+itext+".png"]);
        ResizedImage = imresize(Image,[1000 24]);
        imwrite(ResizedImage,['UN_SM15_' itext '.png']);
    end
end
```