A REPORT

ON

MACHINE LEARNING ALGORITHMS FOR STRUCTURAL HEALTH MONITORING

BY

THUKRAL SHAURYA MANISH	2018AAPS0554G
SHAH VISHWA VIPULKUMAR	2018A7PS0109G
SIDDHARTH S	2018A7PS0265G

AT

CENTRAL ELECTRONICS ENGINEERING RESEARCH INSTITUTE, PILANI

A PRACTICE SCHOOL – 1 STATION OF

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

JUNE 2020

A REPORT

ON

MACHINE LEARNING ALGORITHMS FOR STRUCTURAL HEALTH MONITORING

BY

THUKRAL SHAURYA MANISH	2018AAPS0554G	B.E.(Hons). Electronics and Communication Engineering
SHAH VISHWA VIPULKUMAR	2018A7PS0109G	B.E.(Hons) Computer Science Engineering
SIDDHARTH S	2018A7PS0265G	B.E.(Hons) Computer Science Engineering

Prepared in partial fulfilment of the

Practice School-I Course Nos.

BITS C221/BITS C231/BITS C241

AT

CENTRAL ELECTRONICS ENGINEERING RESEARCH INSTITUTE, PILANI

A PRACTICE SCHOOL – 1 STATION OF

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

JUNE 2020

ACKNOWLEDGEMENTS

We would like to acknowledge and convey our appreciation to all the people who have supported us and guided us during our time at one of India's most esteemed research institutes, CEERI, Pilani.

Firstly, we would like to extend our thanks to Dr D.K. Aswal, Director of CSIR-CEERI, Pilani. For having facilitated the collaboration between BITS Pilani University and CEERI. We would also like to thank Mr Vinod Verma, coordinator of the PS program at CEERI, as, without his efforts, this would not have been possible.

We thank Dr Kota Solomon Raju of CEERI, Pilani for giving us the opportunity to work on this project and his valuable mentorship. We also thank Dr Gaurav Purohit for his help and guidance and support.

Lastly, we would like to thank our PS Instructors, Mr Pawan Sharma and Dr Abhijit Rameshwar Asati for guiding us through the project.

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI

(RAJASTHAN)

Practice School Division

Station: CEERI, Pilani. **Centre:** Pilani.

Duration: 18th May- 28th June, 6 weeks **Date of Start:**18th May, 2020

Date of Submission: 24th June, 2020

Title of the Project: Machine Learning Algorithms for Structural Health Monitoring.

ID No./Name(s)/ Discipline(s)/of the student(s):

THUKRAL SHAURYA MANISH	2018AAPS0554G	B.E.(Hons). Electronics and Communication Engineering
SHAH VISHWA VIPULKUMAR	2018A7PS0109G	B.E.(Hons) Computer Science
SIDDHARTH S	2018A7PS0265G	B.E.(Hons) Computer Science

Name(s) and designation(s) of the expert(s):

Dr. Kota Solomon Raju - Senior Principal Scientist

Dr. Gaurav Purohit - Scientist

Name(s) of the PS Faculty: Mr. Pawan Sharma, Dr Abhijit Rameshwar Asati.

Key Words: Deep Learning, CNNs, Structural Health Monitoring, Machine Learning, RNNs, ANNs, Neural Networks, Tensorflow, Keras.

Project Areas: Structural Health Monitoring, Machine Leaning

Abstract: Rapid SHM systems that leverage the power of deep learning would help first responders prioritise structures which are most likely to fail in the aftermath of a natural calamity. This would help save lives. In this project, we achieve ML for SHM through two approaches. We denoise the noisy signal and determine the structural health of the building from it. We also determined the structural health of the structure using ML directly from the noisy relative acceleration data. For this report, we first summarize the paper upon which we based our ML architecture. Then we discuss some of the theoretical aspects of our project such as signal denoising, followed by the objectives and the description of our project. We conclude with the results and some of the limitations of the project.

Signature(s) of Student(s) **Signature of PS Faculty**

Date **Date**

TABLE OF CONTENTS

INTRODUCTION

Structural Health Monitoring (SHM) refers to the process of implementing a strategy for damage detection for engineering structures such as bridges, buildings and so on. The degradation and ageing of infrastructure are inevitable. The Structural health of buildings will deteriorate over time due to environmental factors or due to extreme natural calamities such as earthquakes or hurricanes. It is very important to periodically monitor the condition of a building. The section below lists gives incentives for doing so.

The motivation for SHM:

Considering that these structures, be it buildings or bridges are used on a daily basis and are exposed to the elements, the sudden failure of these structures could lead to the loss of lives. The integrity of these structures must be monitored regularly. Typically, the process of SHM is done by civil engineers who inspect structures visually and use instruments such as accelerometers to test the integrity of the structure, but this is a time-consuming process.

In the immediate aftermath of a natural calamity such as an earthquake or a hurricane, ideally, first responders, firefighters, and reconnaissance teams should have data pertaining to the structural health of various buildings so that they can address the most at-risk structures with the highest priority, and thereby avoid loss of lives and property to a large extent.

This is where fast SHM systems which leverage the power of Deep Learning can play a major role. Inexpensive and rapid SHM systems would also do away with the need for manual checking done by specialists and would be free of human error.

Thus, there is a lot to be gained by the development of robust, accurate, fast and affordable structural health monitoring systems.

In this project, the CNN classifier architecture that we will be designing for SHM will be based on the CNN architecture proposed in [1]. First, we give a summary of the relevant content in [1]. Then we look at some of the theoretical aspects of signal transformation before finally moving on to the detailed description of our project work.

PAPER SUMMARY

In [1], machine learning is used to identify the status of buildings based on accelerometer data. Unlike most of the prior work in this area, they have used Interstory Drift Ratios (IDRs) to indicate damage in a structure rather than modal parameters. Through documentation from government agencies, they relate IDR values to building damage level as shown below.

IDR %	Building State	Tag
$< 0.7\%$	Immediate occupancy (IO)	Green
$0.7\% - 5\%$	Life safety (LS)	Yellow
$> 5\%$	Collapse prevention (CP)	Red

Table 1: Relation between IDR and building state for steel moment frame buildings

- Immediate Occupancy (IO) –means that it is safe for people to inhabit the building.
- Life Safety (LS) – means that the building is possibly unsafe and requires inspection.
- Collapse prevention (CP) – means that the building has been severely damaged and is unsafe.

They then go on to create their data set by simulating a building's response to historical earthquakes and labelling the response of each floor as a sample according to the severity of the damage. They then artificially introduce noise to obtain both a noisy and a non-noisy dataset.

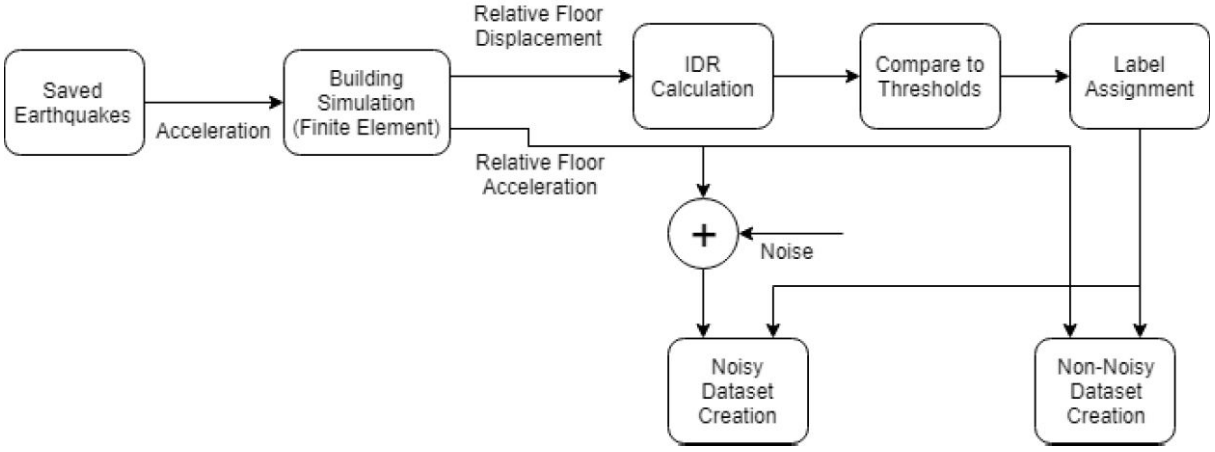

Figure 1: Block diagram of data set creation

The architecture of the CNN proposed by the authors is shown below, in Figure 2.

CNN NETWORK ARCHITECTURE

Layer	Type	Parameters	Activation
0	input	size = 5000	-
1	Conv1D	Kernel size=50 Filters=4	relu
2	Maxpooling	Pool size=20	-
3	Conv1D	Kernel size=10 Filters=10	relu
4	Maxpooling	Pool size=2	-
5	Flatten	-	-
6	Dense	Size=2	tanh
7	Dense	Size=3	tanh

Figure 2: Proposed CNN Architecture

The proposed CNN has 2 convolutional layers, 2 pooling layers and 2 fully connected layers. The final prediction uses one-hot encoding to signify the status of the structure. The paper assumes that the maximum monitoring duration would be 50 seconds. So, at a sampling rate of 100-Hz, the sensors would generate 5000 samples. Thus, the input layer to the network has size = 5000. This input is a 1dimensional array which represents the sensor output over the 50 second time interval.

Finally, the authors make 3 different classifiers using the following machine learning algorithms:

- Support Vector Machine
- K Nearest Neighbours
- The proposed CNN

These classifiers were trained using both the noisy and non-noisy data sets and were validated on the noisy datasets. Below are the normalized confusion matrices of the classifiers.

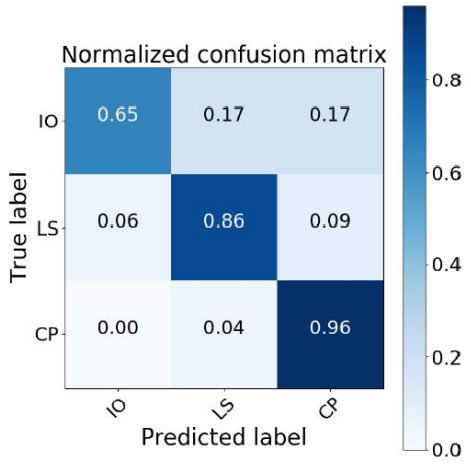

Fig 3: Confusion matrix of SVM trained using the noisy and non-noisy training data sets and validated using the noisy validation one.

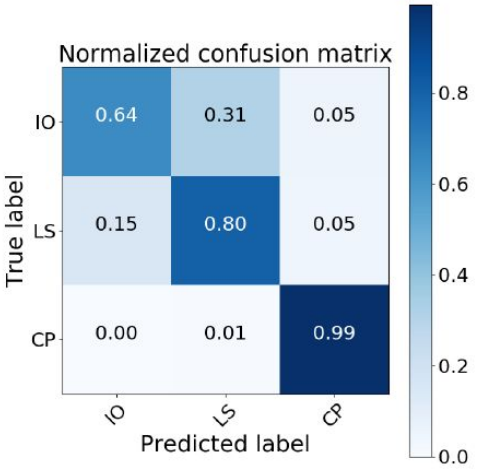

Fig 4: Confusion matrix of KNN trained using the noisy and non-noisy training data sets and validated using the noisy validation one.

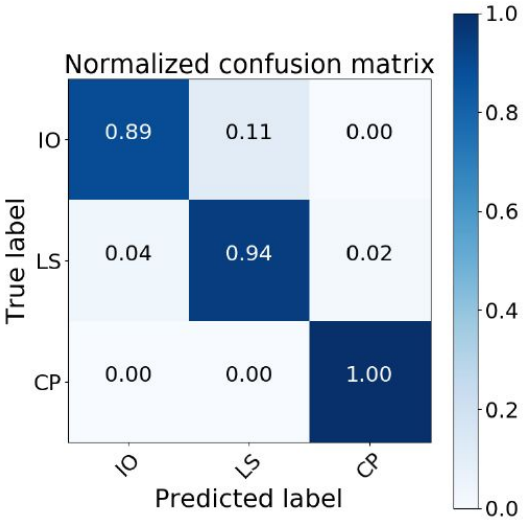

Figure 5: Confusion matrix of the proposed CNN when trained using the noisy and non-noisy training data sets and validated using the noisy validation one.

The main takeaways from the paper are:

- Based on the normalized confusion matrices, the CNN outperforms classifiers that use traditional ML algorithms such as SVM and KNN.
- The CNN can be trained with and make predictions on raw accelerometer data without the need for any data pre-processing.

REVIEW OF SIGNAL TRANSFORMATION

The short review that follows goes over the theoretical aspects and the mathematics behind transforming a signal from the time domain to the frequency domain. We have used these concepts in our project for the purposes of signal denoising.

Introduction to Signal Transformation

Signal decomposition can be utilized to split the signal into simpler components for which the output is already known or can be computed feasibly. After this, the output is synthesized to obtain the final output signal to the system. There are two main ways for transformations - Impulse decomposition and Fourier Transform. Accelerometers are useful for sensing vibrations in systems or for orientation applications. The main objective is to identify and locate any structural damage in real-time by processing the raw vibration signals acquired by a network of accelerometers. Vibration Analysis: Transforms and decompositions give us the spectrum of the constituting frequencies of the accelerometer i.e. what frequencies are present in your signal and in what proportions. If this matches the natural frequency of the structure, the structure is susceptible to damage due to resonance.

Impulse decomposition is one method of decomposition in case of signals. It is important because it allows signals to be examined one sample at a time. Similarly, systems are characterized by how they respond to impulses. By knowing how a system responds to an impulse, the system's output can be calculated for any given input. This relationship has been derived from the fact that x[n] can be represented as a delayed and weighted response of impulse signals at each time step. This impulse decomposition is done using the convolution property as shown below; h[n] is the impulse response here.

$$y[n] = x[n] .* \ h[n] = \sum_k x[k]h[n-k]$$

Fourier decomposition A Fourier series is a way of representing a periodic function as a (possibly infinite) sum of sine and cosine functions. For functions that are not periodic, the Fourier series is replaced by the Fourier transform. In mathematics, a Fourier transform (FT) is a mathematical transform which decomposes a function (often a function of time, or a signal) into its constituent frequencies - sin and cosine components. Below is a Fourier Series representation.

$$x(t) = \sum_{k=-\infty}^{\infty} a_k e^{jk\omega_0 t}$$

Discrete Fourier Transform

Any signal in the time domain can be deconstructed into a weighted sum of sinusoidal signals. Once deconstructed, we can analyse the different frequencies present in the signal. This facilitates the validation and troubleshooting of signals. Thus, in the frequency domain, it is easier to analyse whether a signal contains any kind of noise or jitter.

Here is where we leverage the power of Fourier Transforms to convert signals from the time domain to the frequency domain. These transforms are used on both continuous-time and discrete-time signals. However, since we are dealing with discrete samples of a signal, we specifically have a look at transforms which target discrete-time signals.

There are two such transforms: Discrete-time Fourier Transform (DTFT) and Discrete Fourier Transform (DFT).

The DTFT of a discrete-time signal x[n] is given by:

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}$$

We can obtain the original signal from the DTFT using the inverse DTFT with the help of the following equation:

$$x[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega n}d\omega$$

The DTFT of a discrete-time signal is a continuous function of frequency. As a result, it generates an infinite number of samples. However, because computers are equipped to deal with only a finite number of values, the DTFT has no practical implementation and just a theoretical application.

For the practical computation of the frequency content of real-world signals, the Discrete Fourier Transform (DFT) is used. The DFT transforms N discrete-time samples to the same number of discrete frequency samples and is defined as:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{N}kn}$$

We can obtain the original discrete-time samples from the discrete frequency samples by the inverse DFT:

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j\frac{2\pi}{N}kn}$$

The DFT computes exact samples of the DTFT at N equally spaced frequencies (2πk/N), thereby resolving the issue of infinite samples. This advocates the use of the DFT for practical purposes.

Fast Fourier Transform Algorithm

Discrete Fourier Transform (DFT) is a transformation that is performed widely in the field of Digital Signal Processing. Doing DFT by using the formula directly involves a lot of multiplications and additions. It is computationally expensive.

If we were to perform the DFT using the formula given below,

$$X[k] = \sum_{n=0}^{N-1} x[n]\ e^{-j2\pi kn/N} \quad , for\ \ n = 0, 1 \ldots N-1\ ,\ k = 0, 1 \ldots N-1$$

we would need to do N^2 multiplications and N^2 additions between complex numbers. For example, if N is 1000, we will have to do a million operations. As N increases, the computational complexity increases by N^2. This is where the Fast Fourier Transform algorithm (FFT) comes in.

The derivation below explains the principle behind the algorithm.

$$X[k] = \sum_{n=0}^{N-1} x[n]\ W_N^{nk} \qquad (W_N = e^{-j2\pi/N})$$

Note that W_N is the N^{th} root of unity.

$$X[k] = \sum_{n=even}^{N-1} x[n]\ W_N^{nk} + \sum_{n=odd}^{N-1} x[n]\ W_N^{nk}$$

Changing the local variable from n to r = n/2

$$X[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r]\ W_N^{2rk} + \sum_{r=0}^{\frac{N}{2}-1} x[2r+1]\ W_N^{(2r+1)k}$$

$$X[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r]\,(W_N^2)^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x[2r+1]\ (W_N^2)^{rk}$$

We know that the square of an N^{th} root of unity is an $(N/2)^{th}$ root of unity. Thus,

$$W_N^2 = W_{N/2}$$

Hence,

$$X[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r]\ W_{N/2}^{rk} \quad + \quad W_N^k \sum_{r=0}^{\frac{N}{2}-1} x[2r+1]\ W_{N/2}^{rk}$$

The individual summations are themselves DFTs on signals of length N/2.

We can write them as E[k] and O[k]. Thereby yielding,

$$X[k] = E[k] + W_N^k\ O[k]$$

By the above derivation, we can split a DFT of length N into 2 DFTs of length N/2 each. Evaluating the DFT through this method for $N = 8$ is captured by the below illustration (Figure 6) known as the butterfly diagram.

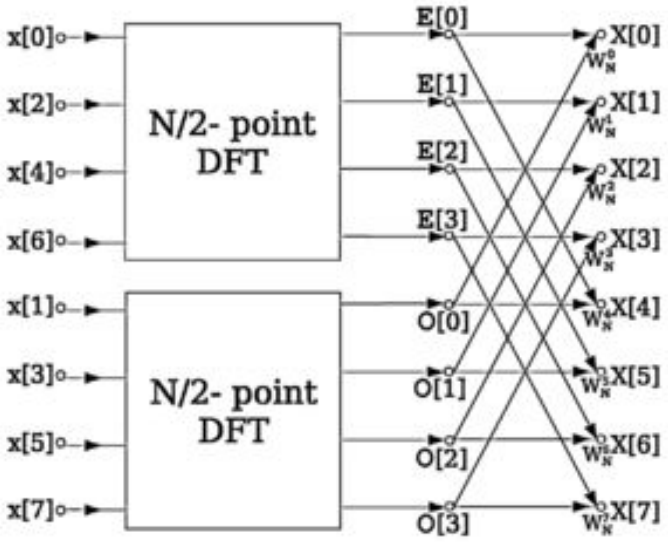

Figure 6: Butterfly diagram

Each N/2 DFT can be further broken down into 2 equally sized DFTs. We can continue this process, thereby enabling us to evaluate the DFT in a recursive fashion.

The FFT has a computational complexity of O(N logN), while calculating the DFT directly from the formula has a complexity of $O(N^2)$. This is a significant reduction in the amount of computation. To put this into perspective, say,

$$N = 2^{10} = 1024\ ,\ then$$

$$N^2 = 1048576 \approx 10^6\ ,\ but$$

$$N log_2^N = 10240 \approx 10^4$$

Thus, we would have to do 100 times the amount of multiplications if we choose to use the direct formula as opposed to the FFT.

In our code we have used the SciPy libraries,

scipy.fft.rfft and scipy.fft.irfft

which allow us to compute the 1-dimensional discrete Fourier Transform for real inputs and its inverse respectively [9].

PROJECT OBJECTIVES

The objective of this project is to use Machine Learning algorithms to determine the structural health of a structure. The objectives of this project align with the objectives of the paper [1]. The ultimate aim of this project is to take noisy accelerometer data from accelerometers placed on two adjacent floors and be able to accurately predict the status of the building. The status of the building can be one of 3 classes, Immediate Occupancy (IO), Life Safety (LS) or Collapse Prevention (CP) as explained in the paper summary of [1]. The status of the building is therefore represented as a 1-dimensional array of size 3 which uses one-hot encoding.

To put it simply, **the goal of the project is to take 2 noisy accelerometer readings and output a 1D array of size 3 which accurately represents the structural health of the structure.**

We also intend to use 2 different approaches, both of which independently satisfy the above goal of the project and within each approach we plan to use both CNN and ANN architectures. In the end, we identify which architecture performed better, as well as which approach was able to classify the noisy accelerometer readings into the IO, LS, or CP classes more accurately.

OUR APPROACHES TO THE PROJECT

We have taken two approaches to achieve the aforementioned goal:

A. **Denoising Approach:** In this approach, we create a Denoiser using Machine Learning algorithms and use it to denoise the noisy signals from the two accelerometers. Once we have obtained the denoised signals from adjacent accelerometers, we can use the acceleration data of each floor to get the displacement data of each floor using double integration. Once we have that, we can get the IDR values from relative floor displacement, and using that and the mapping given in Table 1, we can get the final classification between IO, LS and CP. Shown below is the block diagram (Figure 7) that outlines this approach.

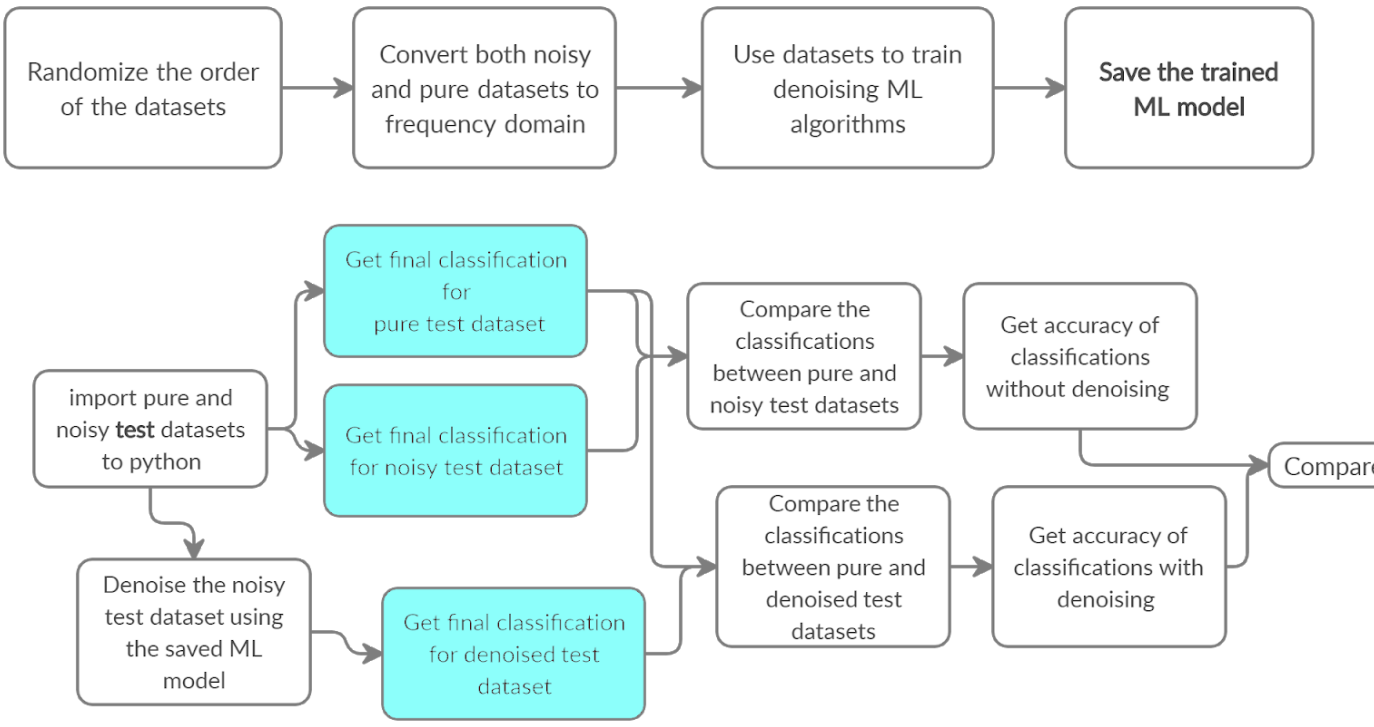

Figure 7: Block diagram of the denoising approach

B. **Direct Approach:** In this approach, we create a classifier which directly takes the relative acceleration between two adjacent floors and gives us a classification between IO, LS and CP. This approach is the same as that stated in [1]. Shown below is the block diagram (Figure 8) that outlines this approach.

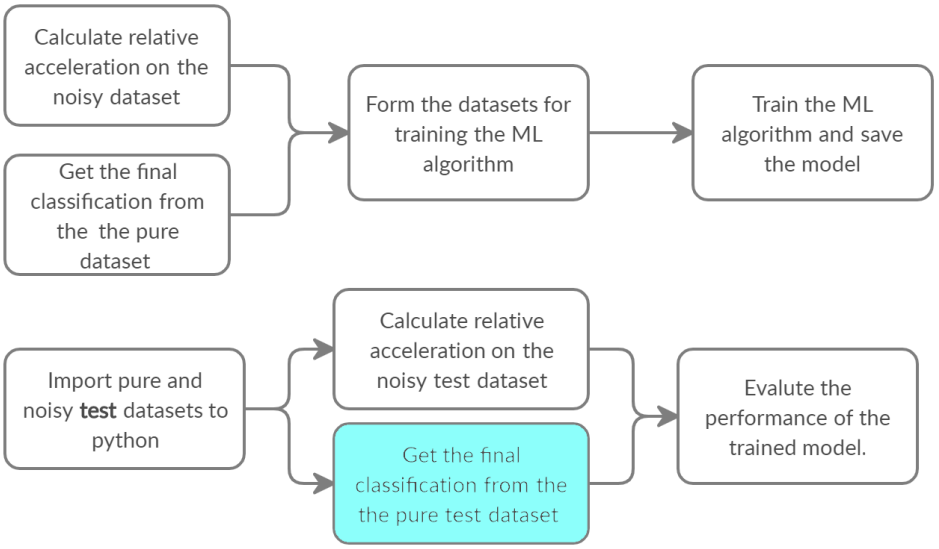

Figure 8: Block diagram of the direct approach

In both block diagrams in Figures 7 and 8, there are blocks shaded light blue. This is to indicate that these blocks can be expanded as shown below in Figure 9.

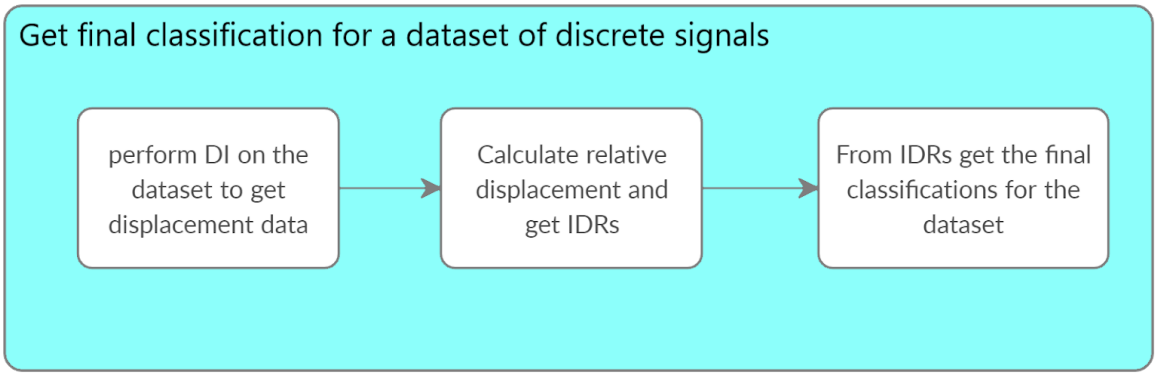

Figure 9: Get final classification for a dataset of discrete signals

We now discuss in detail each step of the two approaches. But before that, we explain how we created our datasets from scratch as well as the details of how we can get the final classification for a dataset through the process illustrated in Figure 9.

DATASET CREATION

For our Machine Learning algorithms, accelerometer readings from adjacent floors of a structure were used as input data. We simulated these readings as a combination of pure sine waves (signals) of different frequencies with varying degrees of noise using MATLAB. Additive white Gaussian noise was used to distort the signals as it mimics the effect of many random processes which occur in nature. The block diagram for the dataset generation is shown below in Figure 10.

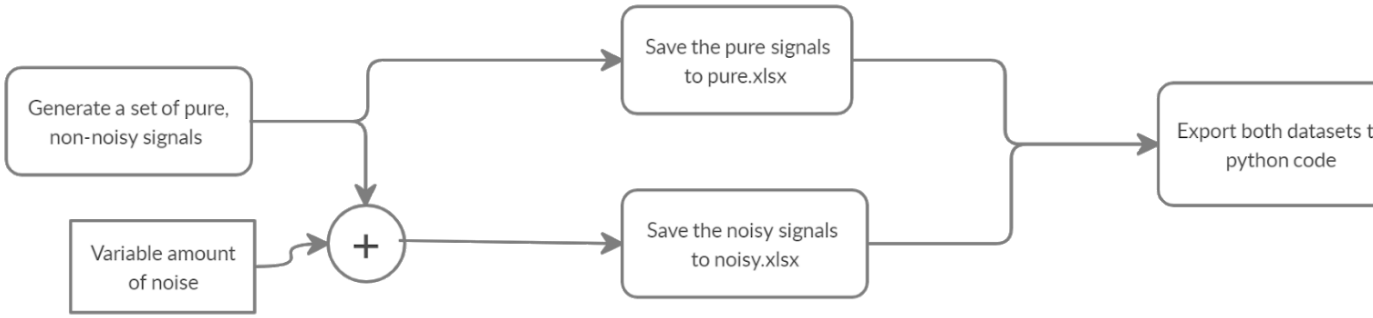

Figure 10: Dataset creation.

The extent of noise incorporated in the signal was manipulated by means of a Signal to Noise Ratio (SNR), which is a ratio of pure signal power to noise power. SNRs of 15, 7 and 1 were used for low-noise, medium-noise and harsh noise signals respectively.

We sampled the generated signals at equal time intervals of 0.01 seconds across 7 seconds to obtain 701 discrete-time samples. These samples were then added as rows to matrices (pure or noisy) depending on the type of signal.

These two matrices which had the pure and noisy samples were then saved to two different excel sheets pure.xlsx and noisy.xlsx respectively. Our python code then imported the pure and noisy datasets from these excel sheets.

For the ML algorithm used in the denoising approach, all we need are the noisy signals and their corresponding pure signals as the training dataset.

However, for the ML algorithm used in the direct classification approach, the dataset used for training consists of the noisy signals and their corresponding classification to the 3 classes. We can get the classification of the noisy signals by performing double integration and IDR calculation on the pure signals that correspond to the noisy signals. This process has been explained in the following section.

GETTING FINAL CLASSIFICATIONS FOR A DATASET

The process of classifying 2 accelerometer signals into one of the three classes can be done by a three-step process, as illustrated in Figure 9. In this section, we explain each step of the process.

Note that this process does not use any ML algorithms and is only used to verify the accuracy of the classification done in the denoising approach and is used to create part of the training and testing dataset in the direct approach. This is shown in Figures 7 and 8 by the blocks shaded in light blue.

Double Integration (DI):

In practice, acceleration is something that we measure at different points in time by using an accelerometer. By performing integration on this discrete acceleration function, we can obtain a discrete velocity function. Similarly, by integrating the discrete velocity function, we can obtain a discrete displacement/position function.

In this project, we have integrated discrete functions using the trapezoidal rule. Shown below are the formulae that we have used to calculate velocity and position.

$$v2 = v1 + \frac{a1+a2}{2}\ (t2 - t1)\ ,\ \text{and}$$

$$p2 = p1 + \frac{v1+v2}{2}\ (t2 - t1)$$

Where $v2$ and $v1$, $a2$ and $a1$, and $p2$ and $p1$ are velocities, accelerations and positions at times $t2$ and $t1$ respectively.

We can also assume that at the time when accelerometer starts taking recordings of note, that the accelerometer was stationary and is in some mean position at that time instant. Thus, we can say that $v1 = 0$, and $p1 = 0$. Thereby enabling us to calculate velocities and positions at all the other time instants.

IDR calculation:

Interstory Drift Ratio (IDR) is defined as the ratio of the relative floor displacement to the floor height [1].

Remember that each row in the dataset corresponds to a signal from an accelerometer. We consider adjacent rows of the dataset to be signals from accelerometers that are positioned at adjacent floors. Once we have calculated the corresponding displacement function from each acceleration function in the dataset, we can evaluate the relative displacement between floors.

Note that the size of the dataset of relative displacements will be half the size of the dataset of accelerometer signals as it takes two accelerometer signals to get one relative displacement signal.

Now we have the relative displacement between floors as a function of time. To get the IDR values as a function of time, we simply divide the relative displacement function by the floor height which we have considered to be 2.75 meters. This is the standard floor height in India [12].

Getting final classification from IDR values:

In the previous section, we had calculated IDR as a function of time. For each IDR value, we can obtain a classification from the mapping given in Table 1. From the IDR function and the mapping in Table 1, we get a function of classifications over time.

However, we wish to get a single classification. To convert this function of classifications to a single classification, we consider the most severe classification which has at least one datapoint in the function as the single classification. The CP classification is considered to be the most severe, while the LS classification is considered to be the least severe.

Thus, we have converted a single classification from 2 accelerometer signals.

THE DENOISING APPROACH

Motivation for Using Neural Networks for Denoising

The motivation behind this method is the fact that deep neural networks can be used for denoising varying sensor signals effectively. Some examples of this are stated below.

As stated in [2], compared with traditional seismic noise attenuation algorithms that depend on signal models and their corresponding prior assumptions, removing noise with a deep neural network is trained based on a large training set in which the inputs are the raw data sets and the corresponding outputs are the desired clean data. After the completion of training, the deep-learning (DL) method achieves adaptive denoising with no requirements of (a) accurate modelling of the signal and noise, or (b) optimal parameters tuning. This is called intelligent denoising. In random and linear noise attenuation, the training set is generated with artificially added noise. In the multiple attenuation step, the training set is generated with the acoustic wave equation. The stochastic gradient descent algorithm is used to solve for the optimal parameters for the CNN. The runtime of DL on a graphics processing unit for denoising has the same order as the $f(x)$ deconvolution method. Synthetic and field results indicate the potential applications of DL in automatic attenuation of random noise (with unknown variance), linear noise, and multiples.

Another similar approach is found in [3] where Complex Deep Learning Models are used for Denoising of Human Heart electrocardiogram (ECG) signals. Effective and powerful methods for denoising real ECG signals are important for wearable sensors and devices. Deep Learning models have been used extensively in image processing and other domains with great success. In this paper, they present several DL models namely Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM), Restricted Boltzmann Machine (RBM) together with the more conventional filtering methods (low pass filtering, high pass filtering, Notch filtering) and the standard wavelet-based technique for denoising EEG signals. The results show the CNN model is a performant model that can be used for off-line denoising ECG applications where it is satisfactory to train on a clean part of an ECG signal from an ECG record, and then to test on the same ECG signal, which would have some high level of noise added to it. However, for real-time applications or near-real-time applications, this task becomes more cumbersome, as the clean part of an ECG signal is likely to be very limited in size. Therefore, the solution put forth in this work is to train a CNN model on 1 second ECG noisy artificial multiple heartbeat data (i.e. ECG at effort). Afterwards, it would be possible to use the trained CNN model in real-life situations to denoise the ECG signal.

Our Implementation

Initial data pre-processing:

We had created our dataset from MATLAB as explained earlier. To prevent overfitting of the model to the training data, we randomized the matrices (pure and noisy) by randomly shuffling their rows. This helped break any kind of patterns existing in the dataset, thereby improving generalizability of the model and ensuring good performance against new, unseen data. The shuffled matrices were then used as our final dataset (individual rows of which represented accelerometer sensor readings of a floor in a structure).

For our first approach, where we denoised signals using CNNs, we converted the generated readings from the time domain to the frequency domain using the Fast Fourier Transform algorithm before feeding them to the ML algorithm. This was done to make the process of analysing signals for noise elimination easier.

After denoising, these signals were transformed back to the time domain by using the Inverse Fast Fourier Transform algorithm. Finally, we use the denoised signals in the time domain to get the final classification using DI and IDR calculation as explained in the previous section.

We now discuss the ANN and the CNN used for denoising.

ANN used for denoising:

The ANN architecture that we have used has 6 layers, all the layers are fully connected layers (also known as dense layers).

The input layer takes a 1-dimensional array which is the noisy signal in the frequency domain. During the training process, the ANN compares its output to the pure signal which corresponded to the noisy signal which was fed to it. This pure signal is also in the frequency domain. Based on the error between the ANN output and the pure signal, the weights and biases of the ANN are altered.

The layers of the ANN have sizes and properties as shown in the following table.

Layer	Type	Parameters	Activation
0	Input Layer	Size = 701 (size of input signal)	Linear
1	Dense	Size = 4096	Linear
2	Dense	Size = 8192	Linear
3	Dense	Size = 4096	Linear
4	Dense	Size = 2048	Linear
5	Dense	Size = 701	Linear

Figure 11: ANN architecture for denoising

As the output of the ANN is a denoised signal in the frequency domain, the output layer has the same size as that of the input layer.

We decided the number of layers of each type based on trial and error. We trained multiple models with slight variations in the number of layers or the size of a layer and so on and checked their accuracies on the testing dataset. We decided on this architecture as it gave slightly better performance compared to the others that we had tested.

All the layers have a linear activation function. Generally, the linear activation function can take the form, $y = ax + b$, for some constants a and b, but in TensorFlow, the linear activation function simply returns the input without doing any modifications to it.

This is important as our ML algorithm does not classify data into multiple classes. Instead, it produces a denoised discrete signal. If we had used other activations such as ReLU, our values would be non-negative or if we used something like sigmoid or tanh, our outputs will only be between 0 and 1 or -1 and 1. This is not what we want when generating a denoised signal. Thus, linear activation is ideal for our purpose. For the same reasons, we have used the linear activation function for the CNN as well.

To train this ANN, we used the SGD optimizer, with a learning rate of 10^{-3}, and we trained it on our noisy training data set after converting the noisy signals to the frequency domain. We ran 100 epochs with a batch size of 12. Our cost function was a simple mean square error cost function.

CNN used for denoising:

The CNN, like the ANN, also takes input as a discrete signal in the frequency domain.

Our CNN has quite a few more layers than our ANN. The architecture of the CNN is summarized by the following table.

Layer	Type	Parameters	Activation
0	Zero-Padding	Padding = 4	-
1	Convolutional 1D	Filters = 16 Kernel size = 7	Linear
2	Zero-Padding	Padding = 8	-
3	Convolutional 1D	Filters = 32 Kernel size = 3	Linear
4	Convolutional 1D	Filters = 32 Kernel size = 3	Linear
5	Convolutional 1D	Filters = 32 Kernel size = 3	Linear
6	Convolutional 1D	Filters = 16 Kernel size = 3	Linear
7	Convolutional 1D	Filters = 16 Kernel size = 3	Linear
8	Convolutional 1D	Filters = 16 Kernel size = 3	Linear
9	Flatten	-	-
10	Dense	Size = 16	Linear
11	Dense	Size = 701 (Size of input signal)	Linear

Figure 12: CNN architecture for denoising

Similar to the case of the ANN, the architecture was decided on through trial and error and after a lot of testing. Again, all layers have a linear activation function for reasons explained earlier.

This time, we used the Adam optimizer rather than the SGD optimizer to train the model with a smaller learning rate of 3×10^{-4}. For training, we use a batch size of 16 and let the model train for 100 epochs.

Results and Comparisons Between the Denoisers

ANN:

On our testing data set, it gave a root mean square error of 0.5127. Shown below are some of the results of the ANN denoiser on noisy input data. The following images have been generated using matplotlib [11].

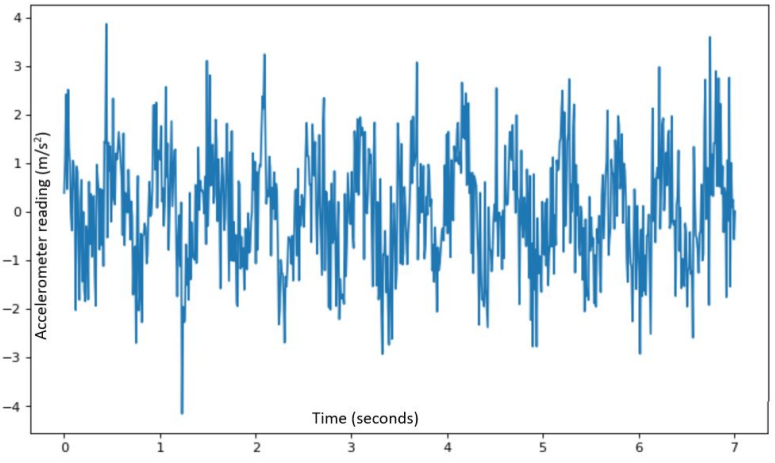

Figure 13(a): noisy signal

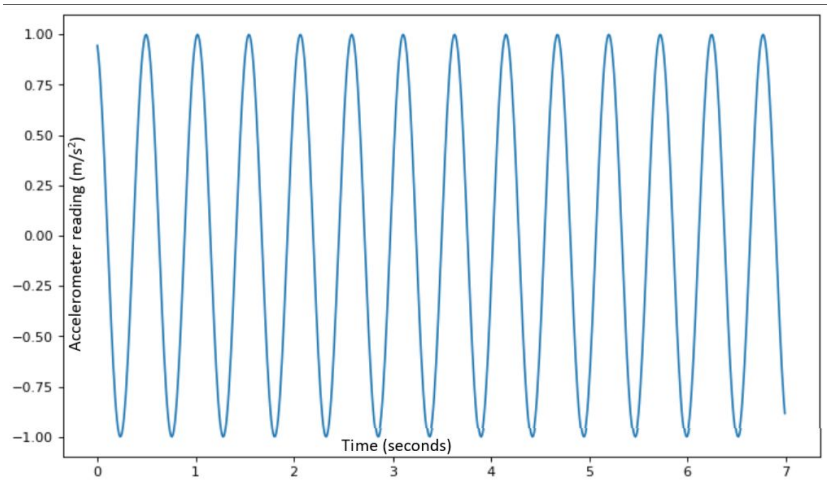

Figure 13(b): pure signal

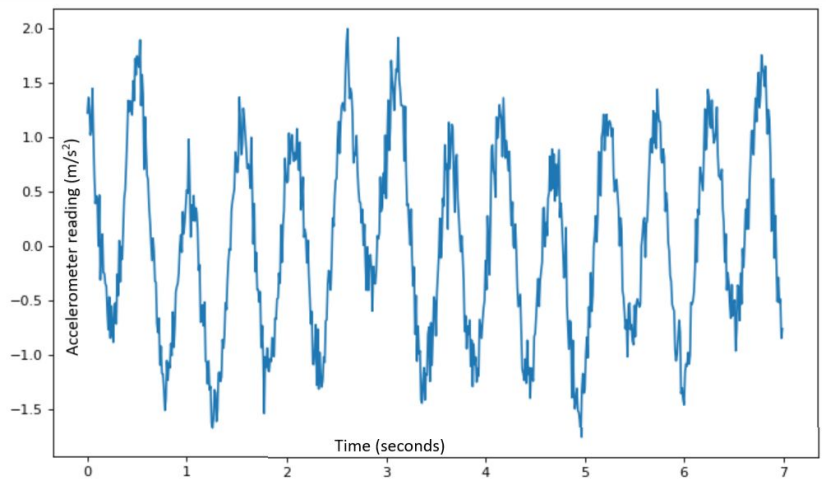

Figure 13(c): denoised signal

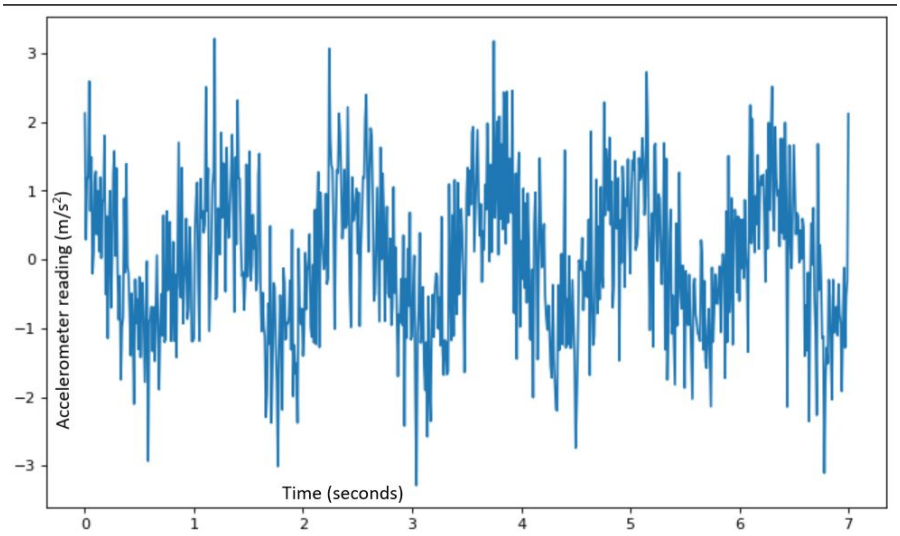

Figure 14(a): noisy signal

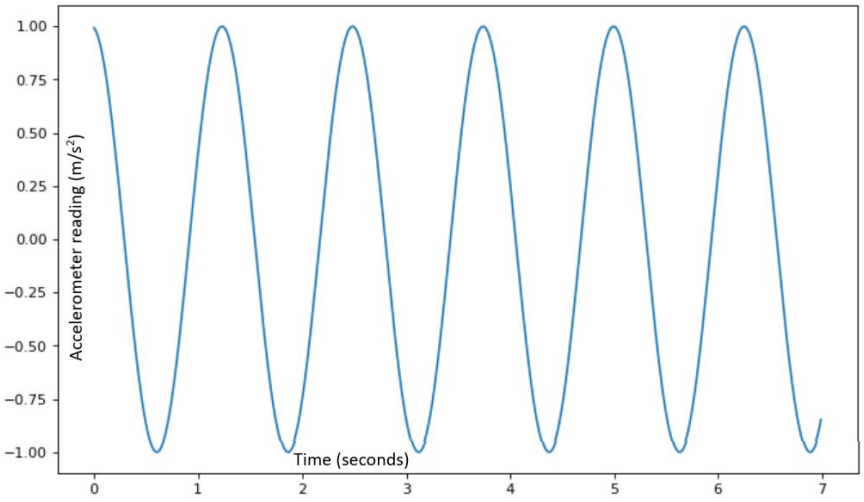

Figure 14(b): pure signal

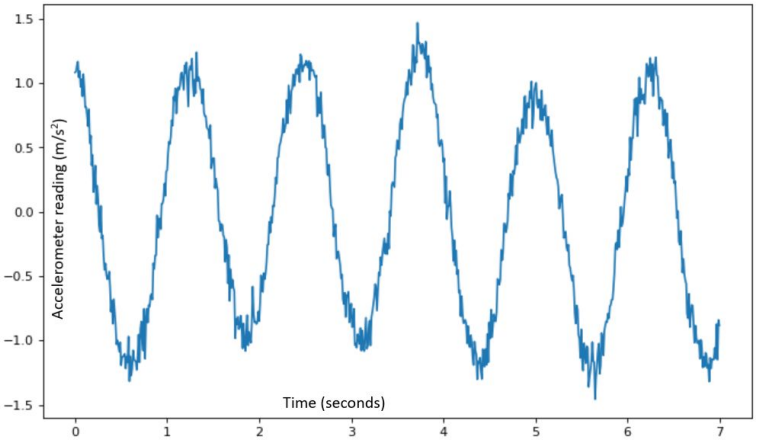

Figure 14(c): denoised signal

As can be seen from some of the results of denoising, the ANN does not perform up to the mark. The denoising is not smooth and there is a noticeable amplitude reduction. The ANN performs worse when the signal is a superposition of sinusoids of varying frequencies.

When we compared the accuracy of classification of signals that had been denoised by the ANN into the IO, LS, and CP classes, to the classification if no denoising was done at all, we saw a trivial 5% increase. The ANN yielded a classification accuracy of 57.45%. This is inadequate. Thankfully the CNN performs much better when it comes to denoising.

CNN:

With this CNN architecture, we were able to get a root mean square error of 0.5547 on the test data. Shown below are some of the results of the CNN denoiser.

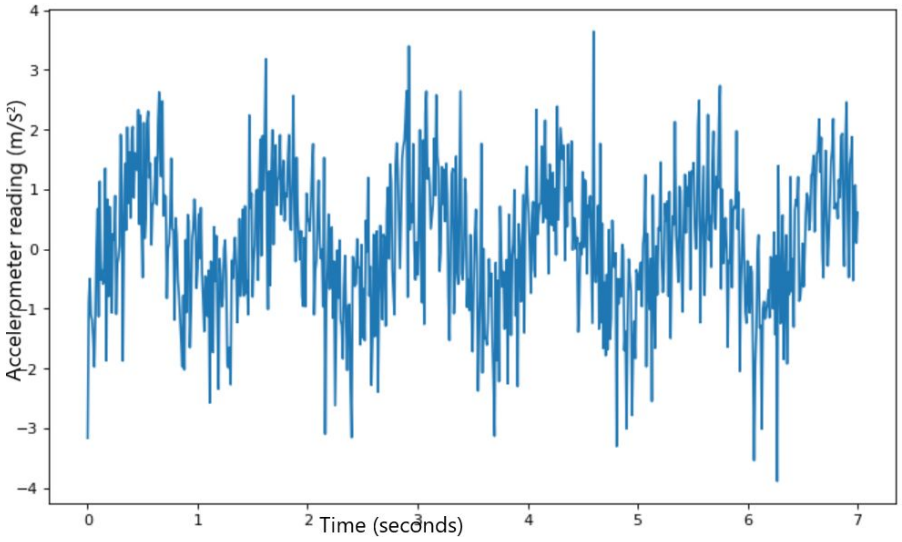

Figure 15(a): noisy signal

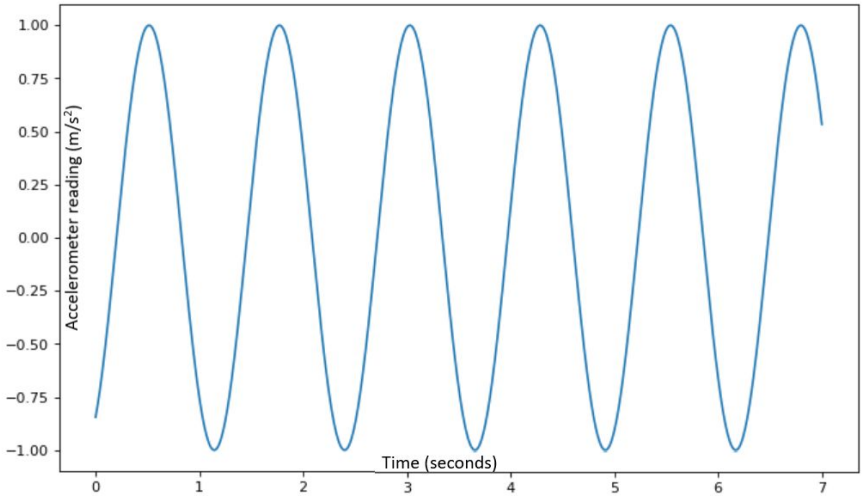

Figure 15(b): pure signal

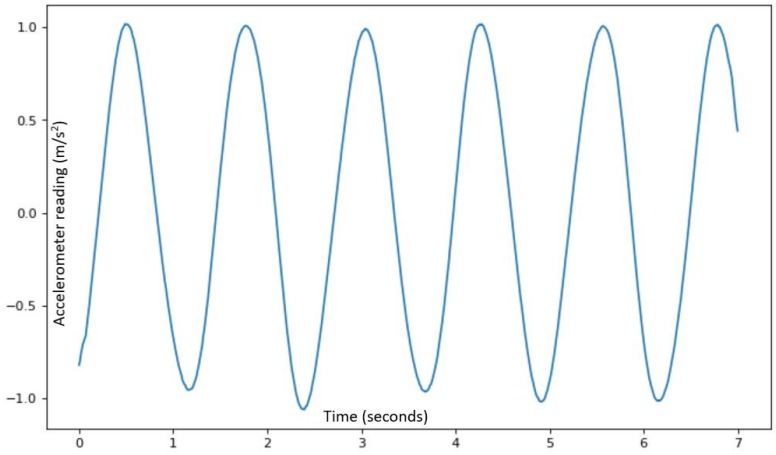

Figure 15(c): denoised signal

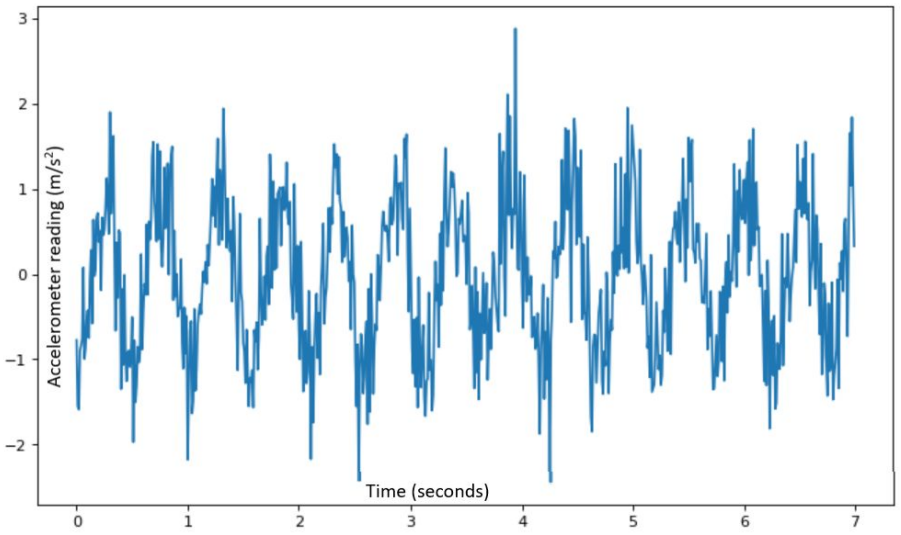

Figure 16(a): noisy signal

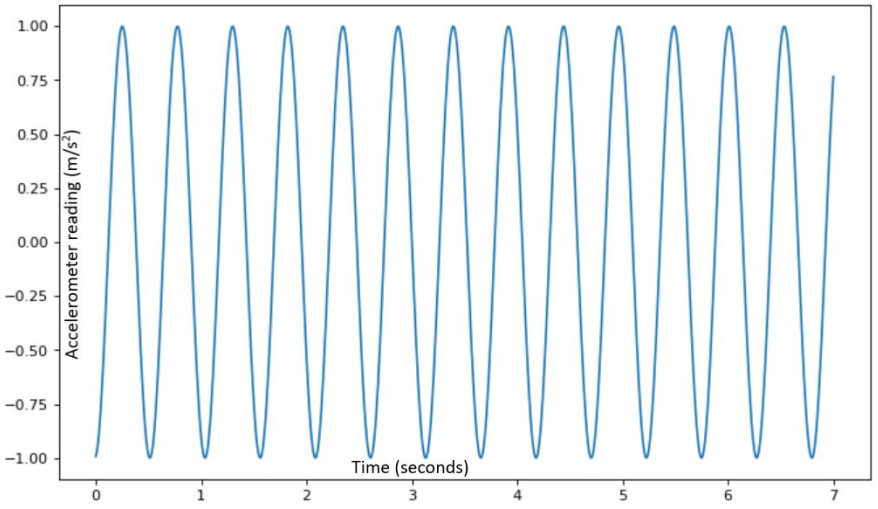

Figure 16(b): pure signal

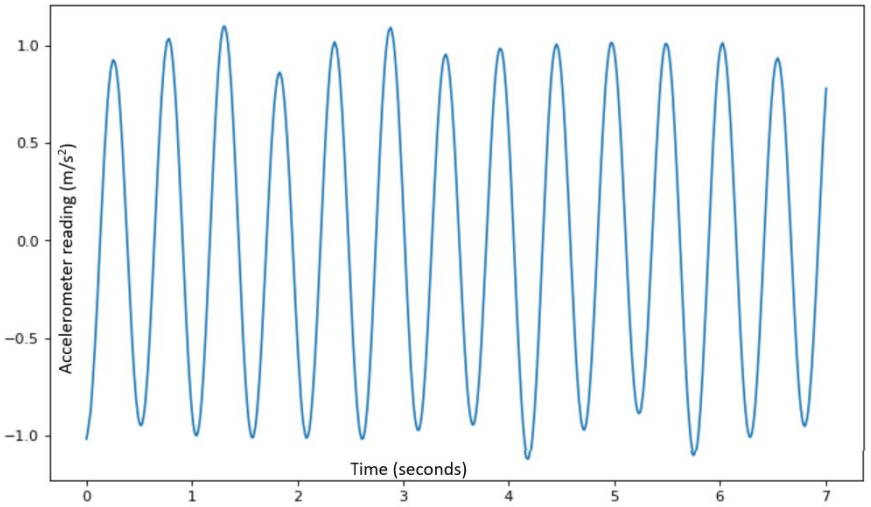

Figure 16(c): denoised signal

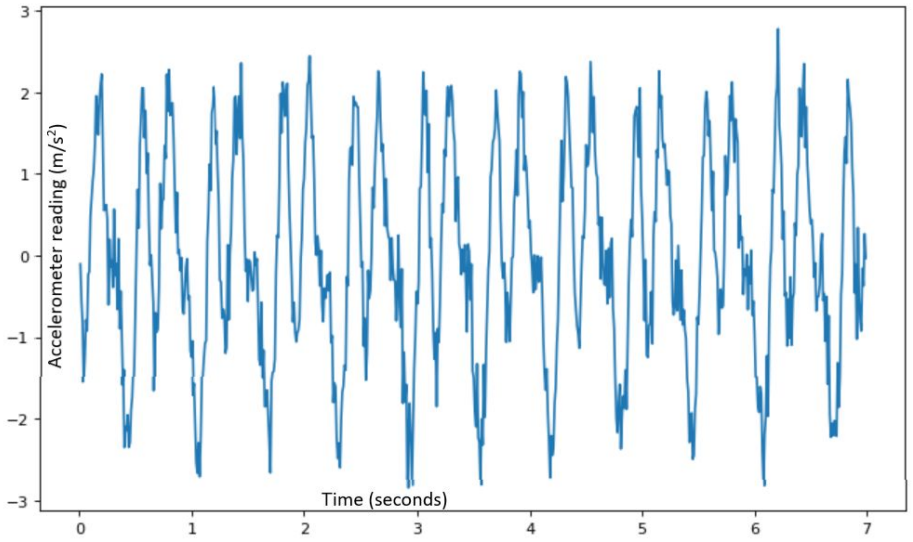

Figure 17(a): noisy signal

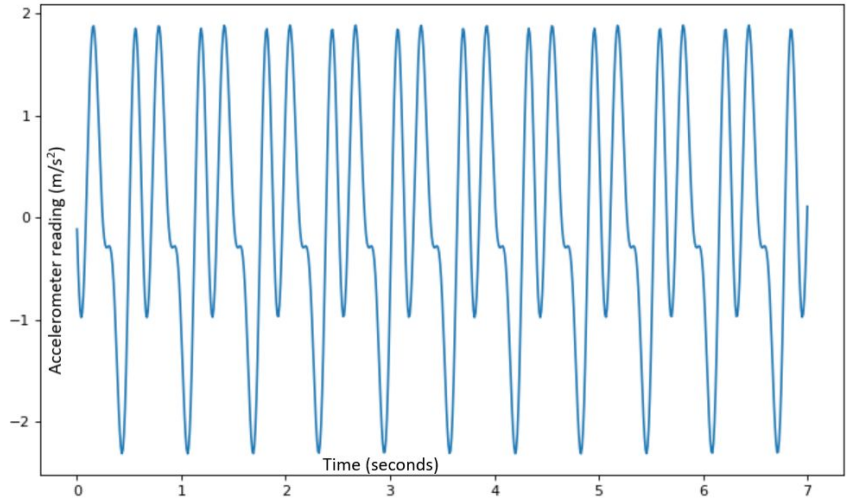

Figure 17(b): pure signal

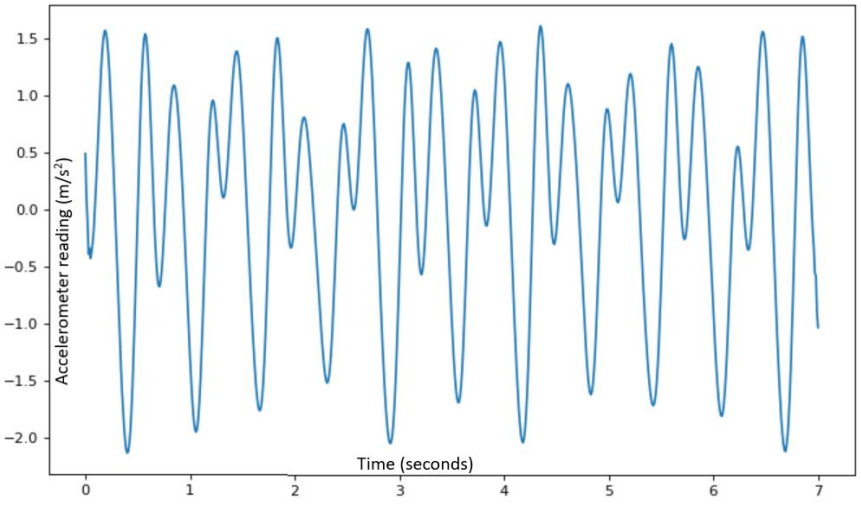

Figure 17(c): denoised signal

The CNN gives much smoother denoised signals compared to the signals denoised by the ANN. Also, the CNN does not have the problem of amplitude reduction. The CNN denoiser clearly outperforms the ANN denoiser. However, like the ANN, the CNN performs poorly when it comes to denoising signals that are a superposition of sinusoids of multiple frequencies.

When we compared the accuracy of classification of signals that had been denoised by the CNN into the 3 classes, to the classification if no denoising was done at all, we saw a significant 25.55% increase, from 52.45% in the case of no denoising to 78.00% in the case of denoising with the CNN. This a significant improvement from the accuracy increase shown by the ANN.

THE DIRECT APPROACH

In this approach, we create a classifier which directly takes the relative acceleration between two adjacent floors and gives us a classification between IO, LS and CP. This approach is similar to that given in [1]. Figure 8 gives a block diagram which outlines this approach.

Motivation for Using Neural Networks

Machine Learning Classifiers have significant potential in processing noisy data. As mentioned in [13] and [14], a CNN is an appropriate method to capture damage patterns in the response of buildings to vibration.

As stated in [13], Structural damage detection has been an interdisciplinary area of interest for various engineering fields. While the available damage detection methods have been in the process of adapting machine learning concepts, most machine learning-based methods extract "hand-crafted" features which are fixed and manually selected in advance. Their performance varies significantly among various patterns of data depending on the particular structure under analysis. Convolutional neural networks (CNNs), on the other hand, can fuse and simultaneously optimize two major sets of an assessment task (feature extraction and classification) into a single learning block during the training phase. This ability not only provides an improved classification performance but also yields a superior computational efficiency. 1D CNNs have recently achieved state-of-the-art performance in vibration-based structural damage detection.

As proposed in [14], the damage detection method operates directly on the raw ambient vibration condition signals without any filtering or pre-processing. This ability is cost-effective and practical in Wireless Sensor Networks considering the hardware systems have been occasionally reported to suffer from limited power supply in these networks. To display the capability and verify the success of the proposed method, large-scale experiments conducted on a laboratory structure equipped with a state-of-the-art Wireless Sensor Networks were reported.

Most of these experiments make use of CNNs, we have also made the use of ANN to check the accuracy of deep neural networks in general.

For both the implementations we have used the same data for training and testing to cross verify and obtain the better accuracy of the two.

Train Data size: (3600,701)

Test Data size: (900, 701)

We have sampled the dataset from 0 to 7 seconds at a rate of 100Hz, hence each accelerometer discrete time series is of length 701. The labelling of the dataset has been done using the most frequent label that has been observed in the entire discrete-time dataset.

Our implementation

Our training data for the ANN and the CNN architectures were obtained as shown in the block diagrams in Figures 8 and 10. Relative accelerometer data obtained from the MATLAB simulations were given as input to both the ANN and the CNN.

Implementation of the Artificial Neural Network:

Table 2 indicates the implementation and parameters of each layer. The input size is the size of the discrete-time signal (time width) that we are observing, the output is a 3-class classification. By repetitive analysis and checking the test accuracy the number and size of dense layers were determined. This is not a linear mapping but a classification algorithm. For our algorithm to learn, a non-linear activation is required, hence we make use of ReLU. Since this is a 3-class classification problem, the last layer is a probabilistic distribution of the 3 classes using softmax.

Layer	Type	Parameters	Activation
0	Input	size = 701(size of input signal)	-
1	Flatten	-	-
2	Dense	size= 512	relu
3	Dense	size= 512	relu
4	Dense	size= 512	relu
5	Dense/Output	output size= 3	softmax

Table 2: ANN Network Architecture

The number of layers and size of each layer is decided based on trial and error and accuracy on the test data. We use categorical cross-entropy as a loss function in this case as this is a classification problem. The metric we have used here is the percentage accuracy, as we evaluate the model directly based on actual and predicted classes. The Optimizer that we have used here is the stochastic gradient descent optimizer as it converges and updates parameters faster. We have trained them for 5 epochs, to avoid overfitting.

Implementation of Convolutional Neural Network:

Table 3 indicates the implementation and parameters of each layer. This is not a linear mapping but a classification algorithm. Therefore, for our algorithm to learn a non-linear activation we make use of the ReLU activation function. The input size is the size of the discrete-time signal (time width) that we are observing, the output is a 3-class classification. The Conv1D layer filters and extracts features from the 1-dimensional time series data we have in this case. MaxPooling1D allows dimensionality reduction and bins out only the essential or highest contributing features. Since this is a 3-class classification problem, the last layer is a probabilistic distribution of the 3 classes using softmax.

Layer	Type	Parameters	Activation
0	Input	size = 701(size of input signal)	-
1	Conv1D	No. of filters = 32 kernel size = 32 stride = 1	relu
2	Conv1D	No. of filters = 32 kernel size = 32 stride = 1	relu
3	MaxPooling1D	pool size = 8	-
4	Flatten	-	-
5	Dense	size = 512	relu

6	Dense	size = 512	relu
7	Dense/ Output	output size = 3	softmax

Table 3: CNN Network Architecture

The number of layers and size of each layer is decided based on trial and error and accuracy on the test data. We use categorical cross-entropy in this case as this is a classification problem. The metric we have used here is the percentage accuracy as we evaluate the model directly based on actual and predicted classes. The optimizer that we have used here is stochastic gradient descent as it converges and updates parameters faster. We have trained them for 5 epochs, to avoid overfitting.

Comparing accuracies

Label	Building State (As mentioned earlier)
0	IO
1	LS
2	CP

Figure 18: label to building state mapping

Shown below are the accuracies on the noisy test data for both cases:

1. Convoluted Neural Networks- loss: 0.3551 - accuracy: 0.8511
2. Artificial Neural Networks- loss: 0.6103 - accuracy: 0.7733

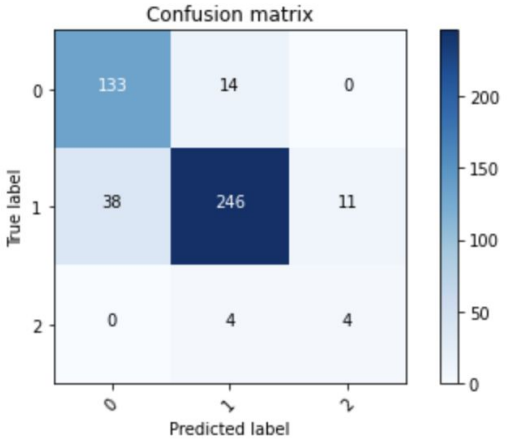

Fig 19(a): CNN confusion Matrix

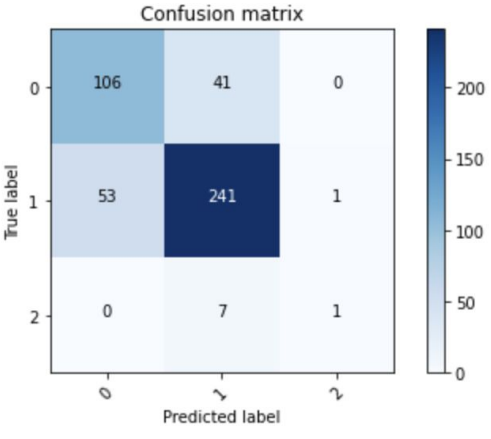

Fig 19(b): ANN confusion Matrix

Both of these confusion matrices have been obtained after modelling on training noisy data and testing on the test validation data [17].

Both Algorithms have pretty good accuracies in terms of direct classification. We can see here that the total number of predictions corresponding to their labels are 383/450 samples in the CNN architecture and 348/450 in the case of the ANN architecture. The CNN method achieves higher accuracy, regardless of filtering the training data. So irrespective of noise or pure data, the algorithm provides a robust method for classification of Building State.

RESULTS

Now that we've gone through both approaches to using ML in SHM, let us compare the results of these approaches.

In both approaches, the CNN outperformed the ANN architectures. In the denoising approach, the CNN managed to achieve a classification accuracy of 78.00% while the ANN only achieved a classification accuracy of 57.45%. In the direct approach, the CNN achieved an accuracy of 85.11% while the ANN achieved an accuracy of 77.33%.

The CNN used in the direct approach achieved an accuracy of 85.11% while the CNN used in the denoising approach achieved an accuracy of 78%. It is clear that the direct approach is more accurate. On top of this, we can say that the algorithm used in the direct approach is faster due to the following reasons:

1. The CNN architecture used in the direct approach has fewer layers compared to the one used for denoising.
2. In the denoising approach, we had to convert the noisy signal to the frequency domain and convert it back to the time domain after denoising. These processes are bypassed in the direct approach.
3. In the denoising approach, we perform DI on the denoised signal and calculate IDRs. This process is computationally expensive. These computations are avoided in the direct approach.

The only computation involved in the direct approach besides the CNN computations itself is the calculation of relative acceleration from the two accelerometer readings. This is done in linear time. Being able to reduce the number of computations is very important as a single structure can have tens of thousands of sensors.

Between the two approaches, the direct approach is far superior, both due to a 7.11% greater classification accuracy and due to the fact that it is faster.

Limitations/Shortcomings of the project:

Below, we list the shortcomings and limitations of our project.

- We had initially set out to achieve an accuracy of over 90%. However, we were unable to achieve this.
- The CNN used in [1] managed to achieve an accuracy of about 95%. The direct approach that we used was based on [1] and we were aiming to surpass it. However, we fell short of this target.
- We had initially decided on building an RNN architecture as well, along with the ANN and CNN architectures. However, the RNN posed a few difficulties while implementing it and the accuracies that we got were quite poor.
- Another limitation this project has is that the ML algorithms did not perform well on accelerometer signals that were a superposition of sines on multiple frequencies, which is a more accurate representation of actual accelerometer readings.
- Another limitation is that the dataset we had worked with was generated artificially in MATLAB. The algorithms have not been tested on actual accelerometer data generated by placing accelerometers on actual structures.

CONCLUSION

No structure can be engineered to last forever. They are subject to periodic usage and are exposed to the elements. Sudden failure of these structures are undesirable and ideally, we would want to know well beforehand if a structure is about to fail. This is why SHM is done in the first place. However, manual inspection is quite time consuming and suffers from potential human error. Thus, successful and affordable implementation of swift and smooth systems which make use of robust and accurate ML algorithms with minimal human intervention have vast applications.

In this project report, we discussed how we artificially created our dataset using MATLAB. We talked about the data pre-processing stages, which consisted of the randomization of our dataset and the conversion of the signals to the frequency domain. We talked about how we can obtain the IO, LS and CP classifications using DI, IDR calculation and the mapping given in Table 1.

Then we moved onto the 2 approaches that we took for ML in SHM. We discussed the architectures of the CNN and the ANN that we built for the purposes of denoising while giving justifications for certain design choices. We displayed the denoising capabilities of these algorithms by showing its effect on some artificial noisy signals and compared the accuracies of these two architectures. We also discussed the architectures of the CNN and ANN in the direct approach. We then compared these two architectures by comparing the accuracies as well as the confusion matrices.

Finally, we discussed the overall results of the project and argued that the CNN architecture outperformed the other architectures as it was the most accurate while also being fast.

APPENDICES

Appendix A: Python Source Code

Full Python Code: Neural Networks Python Code

CNN for denoising:

```
model = keras.Sequential([
  keras.layers.ZeroPadding1D(padding=3),
  keras.layers.Conv1D(16, 7, strides=1, activation='linear'),
  keras.layers.ZeroPadding1D(padding=8),
  keras.layers.Conv1D(32, 3, strides=1, activation='linear'),
  keras.layers.Conv1D(32, 3, strides=1, activation='linear'),
  keras.layers.Conv1D(32, 3, strides=1, activation='linear'),
  keras.layers.Conv1D(16, 3, strides=1, activation='linear'),
  keras.layers.Conv1D(16, 3, strides=1, activation='linear'),
  keras.layers.Conv1D(16, 3, strides=1, activation='linear'),
  keras.layers.Flatten(),
  keras.layers.Dense(16, activation='linear'),
  keras.layers.Dense(pure_acc_freq.shape[1], activation=None)
])
optim = tf.keras.optimizers.Adam(3e-4)
model.compile(optimizer=optim,
            loss = 'mse',
            metrics=[tf.keras.metrics.RootMeanSquaredError('rmse')])
model.fit(noisy_acc_freq, pure_acc_freq, epochs=100, batch_size=16)
```

ANN for denoising:

```
model = keras.Sequential([
  keras.layers.Flatten(),
  keras.layers.Dense(4096, activation='linear'),
  keras.layers.Dense(8192, activation='linear'),
  keras.layers.Dense(4096, activation='linear'),
  keras.layers.Dense(2048, activation='linear'),
  keras.layers.Dense(pure_acc_freq.shape[1], activation=None)
])
optim = tf.keras.optimizers.SGD(1e-3)
model.compile(optimizer=optim,
            loss = 'mse',
            metrics=[tf.keras.metrics.RootMeanSquaredError('rmse')])
model.fit(noisy_acc_freq, pure_acc_freq, epochs=100, batch_size=12)
```

CNN for direct classification:

```
model = keras.Sequential([
  keras.layers.AveragePooling1D(1,1,input_shape = (701,1)),
  keras.layers.Conv1D(32, 32, strides=1, activation='relu'),
  keras.layers.Conv1D(32, 32, strides=1, activation='relu'),
  keras.layers.MaxPool1D(pool_size=8),
  keras.layers.Flatten(),
  keras.layers.Dense(512, activation='relu'),
  keras.layers.Dense(512, activation='relu'),
  keras.layers.Dense(3, activation='softmax')
])
model.compile(loss=keras.losses.categorical_crossentropy,
            optimizer=keras.optimizers.SGD(),
            metrics=['accuracy'])

model.fit(rel_acc_noisy, pure_classification2, epochs=5, batch_size=16)
```

ANN for direct classification:

```
model = keras.Sequential([
  keras.layers.Flatten(),
  keras.layers.Dense(512, activation='relu'),
  keras.layers.Dense(512, activation='relu'),
  keras.layers.Dense(512, activation='relu'),
  keras.layers.Dense(3, activation='softmax')
])
model.compile(loss=keras.losses.categorical_crossentropy,
            optimizer=keras.optimizers.SGD(),
            metrics=['accuracy'])
model.fit(rel_acc_noisy, pure_classification2, epochs=5, batch_size=4)
```

Classifying signals using DI and IDRs:

```
time = 1/360 #time taken between 2 readings. Sampling rate = 360Hz
pure_classification = np.zeros((int(test_pure_acc.shape[0]*0.5),3),np.float64)

v = np.zeros(test_pure_acc.shape,np.float64) #velocity
disp = np.zeros(test_pure_acc.shape,np.float64) #displacement
floor_height = 2.00

for i in range(0,test_pure_acc.shape[0]):
  for j in range(1,test_pure_acc.shape[1]):   #double integration.
     v[i][j] = v[i][j-1] + (((test_pure_acc[i][j-1]+test_pure_acc[i][j])/2) * (time))

for i in range(0,test_pure_acc.shape[0]):
  for j in range(1,test_pure_acc.shape[1]):   #double integration.
     disp[i][j] = disp[i][j-1] + (((v[i][j-1]+v[i][j])/2) * (time))
```

```
for i in range(0,disp.shape[0],2):
  idr = np.zeros(disp.shape[1],np.float64)

  for j in range(disp.shape[1]):
    idr[j] = ( np.abs(disp[i][j]-disp[i+1][j]) )/(floor_height)

  for k in range (idr.shape[0]):
    if idr[k]<0.007:
      scores[0]+=1
    elif idr[k]>0.05:
      scores[2]+=1
    else:
      scores[1]+=1

  #most severe score is considered for labelling the dataset
  if scores[2]>0:
    scores = [0,0,1]
  elif scores[1]>0:
    scores = [0,1,0]
  else:
    scores = [1,0,0]
  pure_classification[int(i/2)]=scores
  scores=np.array([0,0,0])
```

Appendix B: MATLAB Source Code

MATLAB code for dataset generation: Github Link

REFERENCES

[1] A. Ibrahim, A. Eltawil, Y. Na and S. El-Tawil, "A Machine Learning Approach for Structural Health Monitoring Using Noisy Data Sets," in IEEE Transactions on Automation Science and Engineering, vol. 17, no. 2, pp. 900-908, April 2020, doi: 10.1109/TASE.2019.2950958.

[2] Yu, S., Ma, J., & Wang, W. (2019). Deep learning for denoising. *Geophysics, 84*(6), 1-107. doi:10.1190/geo2018-0668.1

[3] Arsene, C. (2019). Complex Deep Learning Models for Denoising of Human Heart ECG signals. *EUSIPCO.2019,* 11-18. doi:10.31224/osf.io/3sdfa

[4] Stanford Lecture Collection | Convolutional Neural Networks for Visual Recognition (Spring 2017): https://www.youtube.com/playlist?list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv

[5] Digital Signal Processing (ECSE-4530) Lectures, Fall 2014: https://www.youtube.com/playlist?list=PLuh62Q4Sv7BUSzx5Jr8Wrxxn-U10qG1et

[6] Keras documentation: https://keras.io/api/

[7] Tensorflow documentation: https://www.tensorflow.org/api_docs/python/tf

[8] Pandas documentation: https://pandas.pydata.org/docs/

[9] SciPy documentation: https://www.scipy.org/docs.html

[10] NumPy documentation: https://numpy.org/doc/

[11] matplotlib documentation: https://matplotlib.org/contents.html

[12] General Building Requirements in India: http://mohua.gov.in/upload/uploadfiles/files/Chap-4.pdf

[13] O. Abdeljaber, O. Avci, M. S. Kiranyaz, B. Boashash, H. Sodano, and D. J. Inman, "1-D CNNs for structural damage detection: Verification on a structural health monitoring benchmark data," *Neurocomputing*, vol. 275, pp. 1308–1317, Jan. 2018

[14] O. Avci, O. Abdeljaber, S. Kiranyaz, M. Hussein, and D. J. Inman, "Wireless and real-time structural damage detection: A novel decentralized method for wireless sensor networks," *J. Sound Vib.*, vol. 424, pp. 158–172, Jun. 2018.

[15]https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31

[16]https://brilliant.org/wiki/fourier-series/#:~:text=A%20Fourier%20series%20is%20a,larger%20sum%20of%20trigonometric%20terms

[17] https://datascience.stackexchange.com/questions/40067/confusion-matrix-three-classes-python

GLOSSARY

Interstory Drift Ratios (IDRs): IDR is the ratio of the relative floor displacement to the floor height.

Accelerometer: An accelerometer is a device which is used to measure acceleration forces.

Modal parameters: Modal parameters are parameters related to the physical and mechanical properties of a structure, such as mass, stiffness, energy dissipation, resonance frequency, damping factor, vibration modes and so on.

Modal analysis: Modal analysis is the study of the dynamic properties of systems in the frequency domain.

Hand-crafted features: hand-crafted features refer to features derived using various deterministic, non-ML algorithms using the dataset.

Epochs: Epoch indicates the number of passes of the entire training dataset the machine learning algorithm has completed.

Learning Rate: The learning rate is a configurable hyperparameter which controls how quickly the model is adapted to the problem

Batch Size: The batch size refers to the number of training examples utilized in one iteration.

Optimizer: Optimizers are algorithms which are used to change the attributes of your neural network such as weights and biases in order to reduce the losses.

Stochastic Gradient Descent: It is a simple optimizing algorithm used to train Deep Neural Networks.

Adam algorithm: Adam algorithm is an adaptive learning rate optimization algorithm that's designed specifically for training deep neural networks.

Confusion matrix: It is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. Along the vertical axis true labels of the samples are shown, and along the horizontal column we have the predicted labels.

Signal-to-noise ratio (SNR): It is a measure that compares the level of a desired signal to the level of background noise.

ReLU: It is an activation function defined as $f(x) = max(0, x)$

Softmax: It is an activation function defined as $f_i(\vec{a}) = \frac{e^{a_i}}{\sum_k e^{a_k}}$ Where a is a vector which represents the values in a layer.