Mechanical Systems and Signal Processing 169 (2022) 108723

Contents lists available at ScienceDirect

Mechanical Systems and Signal Processing

journal homepage: www.elsevier.com/locate/ymssp

CNN and Convolutional Autoencoder (CAE) based real-time sensor fault detection, localization, and correction

Debasish Jana [a,1], Jayant Patil [a,1], Sudheendra Herkal [a,1], Satish Nagarajaiah [a,b,*], Leonardo Duenas-Osorio [a]

[a] Department of Civil and Environmental Engineering, Rice University, Houston, Texas, TX 77005, USA
[b] Department of Mechanical Engineering, Rice University, Houston, Texas, TX 77005, USA

ARTICLE INFO

Communicated by L. Jankowski

Keywords:
Faulty sensor
Fault detection
Fault classification
Data reconstruction
Convolutional neural network
Convolutional autoencoder
Deep learning

ABSTRACT

Increasing advances in sensing technologies and analytics have led to the proliferation of sensors to monitor structural and infrastructural systems. Accurate sensor data can provide information about structural health, aid in prognosis, and help calculate forces for vibration control. However, sensors are susceptible to faults such as loss of data, random noise, bias, drift, etc., due to the aging of sensors, defects, or environmental factors. Although traditional signal processing techniques can detect and isolate faults and reconstruct corrupt or missing sensor data, they demand significant human intervention. The continuous rise in computational power and demonstrated efficacy in numerous domains motivates the use of deep learning to minimize human-in-the-loop techniques. In this work, we introduce a novel, deep learning framework for linear systems with time-invariant parameters that identifies the presence and type of fault in sensor data, location of the faulty sensor and subsequently reconstructs the correct sensor data for fault detection, fault classification, and reconstruction. In our framework, first, a Convolutional Neural Network (CNN) is used to detect the presence of a fault and identify its type. Next, a suite of individually trained Convolutional Autoencoder (CAE) networks corresponding to each type of fault are employed for reconstruction. We demonstrate the efficacy of our framework to address both single and multiple sensor faults in synthetically generated data of a simple shear-type structure and experimentally measured data from a simplified arch bridge. While the framework is agnostic of fault-type, we demonstrate its use for four types of fault namely, *missing*, *spiky*, *random*, and *drift*. For both simulated and experimental datasets with a single fault, our models performed well, achieving 100% accuracy in faulty sensor localization, more than 98.7% accuracy in fault type detection, and more than 99% accuracy in reconstruction. Our framework can also address multiple concurrent faults with similar accuracy. We empirically demonstrate that our proposed framework performs better than other state-of-the-art techniques in terms of computational efficiency with comparable accuracy. Adoption of our framework in online structural health monitoring applications can lead to minimal disruption to monitoring processes, reduced downtime for structures and infrastructure while simultaneously reducing uncertainty and improving the quality of sensor data for historical records.

* Corresponding author at: Department of Civil and Environmental Engineering, Rice University, Houston, Texas, TX 77005, USA.
E-mail address: Satish.Nagarajaiah@rice.edu (S. Nagarajaiah).
[1] Authors' with equal contribution.

https://doi.org/10.1016/j.ymssp.2021.108723
Received 28 May 2021; Received in revised form 20 October 2021; Accepted 2 December 2021
Available online 12 January 2022

1. Introduction

In recent years, complex structural and infrastructural systems have seen the rise of instrumentation with sensors for system monitoring and management, operational control, maintenance, and security [1–4]. For structures, ambient excitation like wind gust induces a vibration response that can be recorded by sensors mounted over the entire structure. Structural Health Monitoring (SHM) techniques can monitor the response of structures to such ambient excitation and also estimate performance degradation. Various types of sensors provide critical data to build these SHM models. However, even reliable sensors are susceptible to failure due to environmental events or electronic defects arising from regular use. This results in loss of data or errors in data such as bias, drift, measurement freeze, and precision degradation [5,6]. Structural model construction using such corrupted data results in misinterpretation of structural performance and the possibility of unnoticed structural damage, which might lead to catastrophic failure. Thus, data validation is crucial to maintain the reliability of SHM processes that rely on collected sensor measurements. Sensor data reconstruction is essential to having complete historical records and making practical decisions more generally. It can also be used for the detection of an anomaly in sensor data by comparing reconstructed data with the measured data [5].

Sensors are susceptible to faults due to harsh environmental conditions, poor installation, inadequate maintenance, malfunction, etc. Ni et al. [2] describe the following types of faults that could occur in sensors: outlier, spike, 'stuck-at', high noise or variance, calibration, connection, hardware, low battery, environment out of range, and clipping. The latter five types are dependent on external factors and require physical intervention for rectification. The former five types can be computationally detected and approximately corrected. Outlier data points can be easily detected by specifying a threshold and using a rule-based method; or by using Principle Component Pursuit (PCP) [7]. Hence, we consider only the spike, 'stuck-at', high noise, and calibration fault types in our work and describe them using a slightly different taxonomy in Section 2. Ni et al. [2] define the *spike* fault type as multiple data points with a much greater rate of change than expected. *'Stuck at'* fault occurs when the variation in sensor values is zero for an unexpected length of time. *High noise* is defined as the existence of unexpectedly high variation or noise in the sensor values. Lastly, the *calibration* fault is characterized by the offsetting of reported sensor values from the ground truth.

Numerous methods exist to detect sensor faults and validate the measurement data. Faults in sensors attached in a system for health monitoring [8] as well as faults in actuators mounted in systems for control purposes [9] can be detected using analytical models. Model-based methods for sensor data validation use computational models to reconstruct data and use its difference with the measured sensor data to identify anomalies. Model based bank of AR Markov observers have also been proposed for fault detection [10]. These models usually exploit correlation or analytical relations between several variables measured at different times (temporal redundancy), or locations (spatial redundancy), or both (Spatio-temporal redundancy) [6]. Olsson et al. [11] suggest the use of interpolation methods for reconstructing faulty data in sensors. Mourad and Krajewski [12] use smoothing methods by applying criteria-based tests as a prevalidation step. However, their approach warrants manual intervention by an operator for the final validation. Ustoorikar and Deo [13] use genetic programming for the in-filling of missing data using spatial correlations, but the accuracy of their model decreases as the duration of missing data to be in-filled increases. Yoo et al. [14] validate and reconcile sensor data using a structured residual approach with maximum sensitivity (SRAMS) to detect, identify, and reconstruct single and multiple sensor faults. Their method is agnostic to the type and application of sensors. These methods require manual intervention for validation of identified faulty sensors but have room for increasing the accuracy of reconstruction.

Statistical machine learning techniques have also been applied for reconstructing sensor data based on the spatial correlations between the sensors [5,15,16]. Kerschen et al. [17] use principal component analysis for detecting a fault in sensor data and reconstructing the correct estimate. Yang and Nagarajaiah use Principal Component Pursuit (PCP) to detect outlier and noise in the signals as PCP finds out the low-rank structure of the signal [7]. Kullaa [18] leverage the covariance in sensor measurements and employ a minimization of mean squared error (MMSE) technique to reconstruct sensor data. Law et al. [19] use support vector regression (SVR) for sensor data reconstruction for bridge monitoring, and Cheng et al. [20] demonstrate the use of SVR and neighbor coordination for wireless sensor network. Compressive sensing [21,22] based data reconstruction techniques have been explored to address the data packet loss in wireless sensors [23,24]. Neural Networks (NN) have also been exploited for reconstruction in various studies [25]. Whereas these methods exploit spatial correlation to reconstruct the signal, they do not use the additional information captured in the temporal correlations between the variables. Techniques using spatiotemporal correlation can help achieve higher accuracy in sensor data reconstruction [26].

Recurrent Neural Networks (RNN) are NN models that can map sequential input data like time series to their output [27]. Unlike some NN architectures like feedforward neural networks (FFNN), RNNs can use previous output predictions as inputs for making new predictions with reasonable accuracy. This helps the network learn sequential information [28]. Long short-term memory (LSTM) [29] is a type of RNN architecture that can learn long-term features in sequential data using feedback connections. It is used in various applications with sequential or time-series data. A simple RNN model should theoretically be able to capture arbitrarily long-term features in sequential data. However, code implementations using finite precision numbers can suffer from the 'vanishing gradients' problem, in which the finite-precision gradient values tend to zero during successive back-propagation calculations during the training. LSTM addresses this problem by allowing at least one path along the back-propagation steps where the gradients will not vanish and hence, worth comparing with the newly proposed method in this paper. Bidirectional recurrent neural network (BRNN) is another type of RNN architecture that learns the temporal correlations in both forward and backward time direction [30]. Recent work by Jeong et al. [26] demonstrates the use of the Bidirectional Recurrent Neural Network-based framework for sensor data reconstruction. Their framework assumes that the faulty sensors are already identified, after which individual BRNN models corresponding to each faulty sensor are trained using historical, accurate data. We use the LSTM and BRNN models as benchmark methods for comparing the performance of our models.

The previously described frameworks do not have an automatic fault detection and localization mechanism and therefore can only be used in offline mode and with manual intervention. Online implementation of end-to-end fault detection provides a way for real time fault detection, classification and reconstruction but requires the automatic detection of sensor fault, which is the focus of this study. Whereas RNNs and BRNNs can be trained to address faulty sensor data if the faulty sensor is specified, they are not known to identify the type of fault itself. The detection of a sensor data fault and identification of fault type is a multiclass classification problem. Multiclass classification tasks in other domains have benefited from the advances in convolutional neural network (CNN) architectures [31] and could also be employed for detecting the presence and type of sensor data fault. In recent years, CNN models have been successful in various image classification tasks [31,32]. Zhao et al. [33] demonstrate the use of CNN for time series classification. CNN has also been used to solve sequential problems in natural language processing [34] and also for series prediction such as stock price prediction [35,36]. CNNs can automatically identify better multi-layer hierarchical patterns than human-engineered features, and the local receptive field enables detecting small features, while weights-sharing and pooling reduce the computational cost [37].

Autoencoder (AE) is a NN model trained to output a low-dimensional representation of the input [38]. During training, the data is transformed into lower-dimensional representations. In convolutional autoencoders (CAEs), first proposed by Jarrett et al. in 2009 [39], the CNN architecture is combined with AE. Convolution layers are used for encoding and deconvolution layers for decoding instead of the fully connected layers. It is well known that CNNs perform well on image processing tasks even on noisy, shifted, and corrupted image data [40]. CAEs have also been suggested to be better at image processing tasks, in comparison to autoencoders, as they leverage the characteristics of CNNs [41].

To take advantage of the Spatio-temporal correlations in sensor data and to develop a real-time setup that can address automatic fault identification and reconstruction for multiple concurrent faults, we propose a two-step framework based on CNN and CAE. In the first step, a CNN is used to identify the type of sensor fault. Based on the identified fault-type, the second step employs a CAE Network to reconstruct the data. Our work is scalable to a large number of sensors, and the end-to-end implementation enables autonomous fault-identification and reconstruction of faulty-data. The number of sensors of a system that can be included in our framework is primarily constrained by the computational cost associated with training the CNN and CAE models. Our proposed framework requires only r models for addressing r types of faults in a sensor network. If n, m, and r represent the total number of sensors, maximum permissible number of faulty sensors, and the number of types of sensor data faults, respectively, other state-of-the-art methods require $\sum_{i=1}^{m} \binom{n}{i} i^r$ models, which requires longer training time and larger computational cost, in comparison to our framework which can achieve a similar level of accuracy, but with lower training time for the models, and significantly lower computational cost as the overall number of models remains constant. This framework can, therefore, be integrated directly into the data acquisition system or implemented using edge computing. Along with a noticeable improvement in accuracy, the primary advantage of our approach is the ability to address faults in multiple sensors with an identical computational cost. This makes our framework particularly amenable for sensor networks with a large number of sensors. To the best of the authors' knowledge, no prior work exists that can reliably address detection, classification, and reconstruction all in a unified end-to-end framework with reasonable computational effort.

This paper is organized as follows: Section 2, provides a brief background on the major types of faults occurring in sensor data and the faults considered in this work. Section 3 describes our proposed framework and the neural network architectures used in our models. In Section 4, the framework is demonstrated using a numerical computer simulations benchmark of a 10 degree of freedom, linear time-invariant, shear-type structure. Next, in Section 5 modified data from physical experiments on a simplified model of an arch bridge is used to examine the efficacy of the framework. Finally, Section 6 concludes the paper with key discussions and suggested future work.

2. Sensor fault types

In this section, we discuss the most common sensor fault types that are encountered in structural health monitoring applications. For a comprehensive review of the types of faults in sensor data, we refer the reader to Ni et al. [2]. Tang et al. [42] discuss fault types with a particular focus on SHM. Among the common sensor faults [2,42], in this paper we deal with four sensor fault types – *Missing*, *Spiky*, *Random*, and *Drift*. Other sensor fault types like square waveform (where the sensor data takes square waveform), non-stationary noise, amplitude modification (where the amplitude of sensor data gets modified with an arbitrary multiple), precision degradation (the contamination of a signal with high ambient noise) can also be addressed by the framework we proposed – we only chose the aforementioned four fault types (*Missing*, *Spiky*, *Random*, and *Drift*) for demonstration. The approach we adopted to generate faults in this paper is inspired from the review paper on sensor data fault types by Ni et al. [2]. The numerically generated faults in this paper are based on the features detailed in the study by Tang et al. [42] which uses a computer vision-based technique to detect the type of sensor fault. In their work, vibration data was collected from a real, cable stayed bridge and the anomaly patterns were characterized. Their study is based on manual labeling of fault types by humans and relies on visually identifiable features. The main objective of our proposed framework is to automate the entire process of fault detection, localization, and reconstruction. Further, we have assumed that a sensor is affected by a single fault type at a given instance. We also consider combinations of two concurrent faults of different types occurring in two different sensors and discuss this in Section 4.9.2. However, as the concurrent occurrence of all 4 types of faults is very unlikely, we have not addressed it in this paper. Fig. 1 illustrates the fault types addressed in our work.

- **Missing data:** In a sensor with missing fault, significant time history is missing and has zero variation due to sensor blackout or packet loss in wireless sensors as shown in Fig. 1(a).

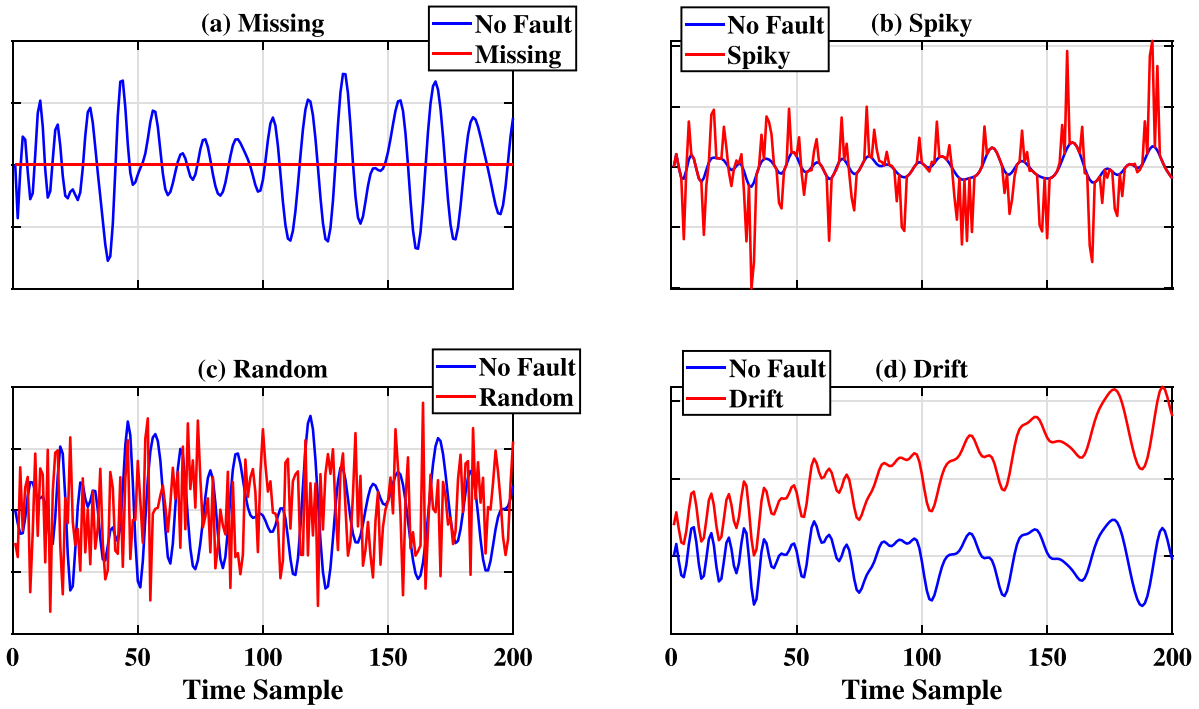

Fig. 1. Common types of sensor faults.

- **Spiky:** Spiky fault is characterized by time history containing multiple data points with a much greater rate of change than expected, as illustrated in Fig. 1(b). The spikes or jumps increase the mean absolute amplitude of the sensor data.
- **Random:** In a sensor with random fault, the time history response is replaced by random noise and does not have any information about the system. The random sensor fault is shown in Fig. 1(c). The sensor records random noises, which does not characterize any information about the frequency content of the system.
- **Drift:** The drift type of fault is a calibration fault that offsets the sensor values from ground truth by adding a linear or lower-order polynomial trend-line as displayed in Fig. 1(d).

While all four faults are common, *Spiky* fault, which is similar to outliers in data, is one of the most common sensor data faults. The detection of drift fault without human input is generally difficult [2].

3. Proposed CNN and CAE end-to-end framework

A good framework for sensor fault type classification, detection, and reconstruction should be robust against noise, type of structural systems, and external forcing functions. For structures with large number of Degrees of Freedom (DOFs), there is a need for large number of sensors to accurately estimate the structural properties for health monitoring [43]. Therefore, a reliable framework proposed for sensor data reconstruction should also be scalable with the number of sensors and should be able to reconstruct data in the presence of different types of faults. Additionally, the framework should detect and correct for faulty data autonomously so that it can be directly used with minimal human intervention for the task.

We propose an end-to-end framework as shown in Fig. 2 to address these problems. The main steps of our framework are the pre-processing step of vibration response data, the classification step, the reconstruction step, and the post-processing step. The classification model hereafter referred to as 'classifier', identifies the type of sensor data fault. Four individual reconstruction models are employed in our framework (RC_1, RC_2, RC_3, RC_4), hereafter referred to as 'reconstructors', corresponding to the four types of considered faults. We use basic signal processing techniques for pre-processing and post-processing of the data. Pre-processing packets real-time data into fixed time-width windows, which are passed as input to the classifier. The classifier and reconstructors utilize deep learning architectures, namely Convolutional Neural Network and Convolutional Autoencoder. The framework illustrated in Fig. 2 shows an example where the input has data corresponding to one sensor missing. The classifier identifies the fault as *missing* type and invokes reconstructor, RC_1, corresponding to the missing fault type as indicated by the first red arrow. Subsequently, the reconstructor predicts the sensor data, which is passed to the post-processing module. The post-processing module identifies the unique sensor with the fault, detects whether it is a sensor fault or a possible system fault, and yields the final output with the reconstructed data.

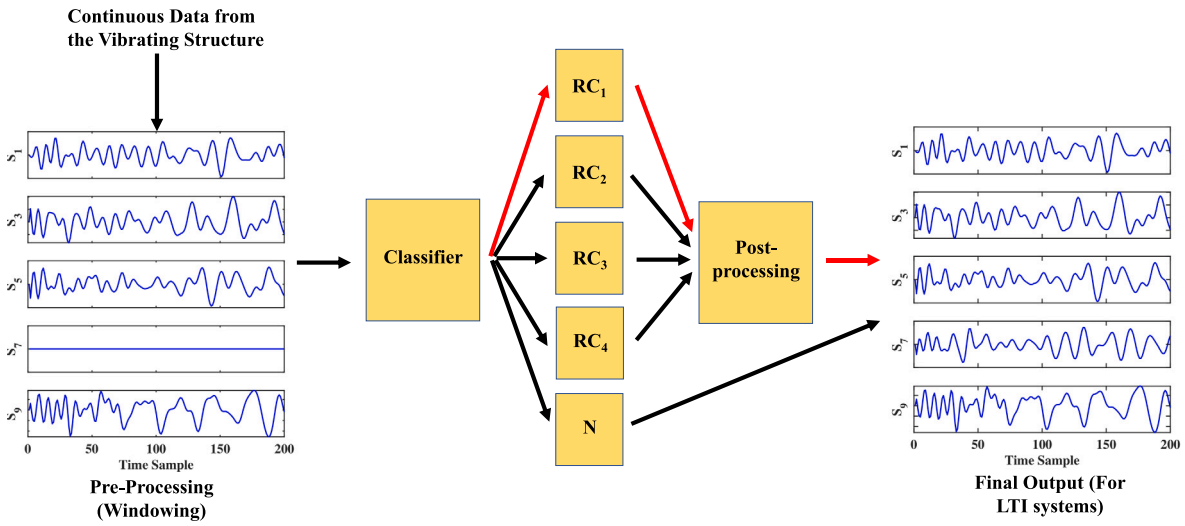

Fig. 2. Illustration of the proposed CNN/CAE end-to-end framework with an input example with data in sensor S_7 missing. Reconstructors corresponding to the *missing*, *random*, *drift*, and *spiky* faults are represented by RC_1, RC_2, RC_3, and RC_4, respectively. N corresponds to normal data without any faults and is passed without any processing.

The process of fault diagnosis – dealing with sensor faults – has four distinct stages: fault detection (presence of fault), fault isolation (location of fault), fault identification (type of fault), and fault accommodation (Data reconstruction) [44]. In our proposed framework, there are three main stages – (1) classifier (CNN) detects the presence of fault and classifies the fault type. According to fault type, the data is either sent to autoencoders (CAEs) or it is labeled as Normal data which does not need any processing. (2) Next, the fault accommodation is performed by the reconstructor (CAEs) corresponding to the fault type. (3) Finally, the fault isolation is performed by the post-processing module. This framework is applied on a sliding window of the real-time data, an example of which is shown in Fig. 3.

We select five instances of the sliding window — denoted as green, red, magenta, violet, and brown. These windows have normal data, missing data at S_4, random data at S_2, spiky data at S_3, and drift data at S_4 respectively. As instant when the responses in magenta window goes through the framework, first the CNN classifier detects this as a random fault and sends it to RC_2 which is only assigned with the task of random data reconstruction. In the post-processing stage the location of the fault is detected at S_2 and the final reconstructed data comes out as the output of the framework. In this figure we only show single fault, but this framework is also applicable for multiple fault as describe later in Section 4.9. Also, as the framework works on window, hence this approach is capable of addressing temporary faults as well.

3.1. Pre-processing of response data

The raw sensor measurements containing the vibration response of a structure are the input to the framework. The sensor data carries information about the system or structure, as well as characteristics of fault(s) if they are present. The real-time response is divided into small time-windows for monitoring the exact time of the fault occurrence. This processed data is the input to the classifier step. The input dimension to the classifier would be $T \times N_s$, where 'T' is the number of time steps in a single time window and 'N_s' represents the number of sensors. The vibration response of each sensor is individually standard normalized as each sensor data may have widely varying scales of response amplitude. Standard Normalization can be expressed as $z = (x - \mu)/\sigma$ – for each sensor, z is normalized data, x is raw data, μ and σ is the long-term mean and long-term standard deviation of the healthy sensor time history. For each sensor the μ and σ are constant values. Even with the faulty sensors the standard normalization is carried out with the constant μ and σ. In real scenarios/experimental cases, μ and σ can be obtained from the long-term health sensor time histories. This is also a common practice in deep learning applications and is done to achieve faster convergence during training [45,46].

3.2. Classifier

The classifier detects the 'type of fault' present in the data. If the classifier deduces that the response data is normal, the final output of the entire framework will output the normal data without any processing. However, if the classifier detects the presence of either one of the four faults, the sensor data will be passed to the unique reconstructor corresponding to the detected type of fault. Since, *Normal* is one the classes included in classification, the fault detection and classification is achieved in a single step by the classifier. We use a Convolutional Neural Network (CNN) model for classification.

CNN architectures have achieved high accuracy in numerous image classification tasks [31]. Pixels in an image are stacked measurements from a sensor detecting the intensity of light of a specific wavelength. When the data from multiple sensors are

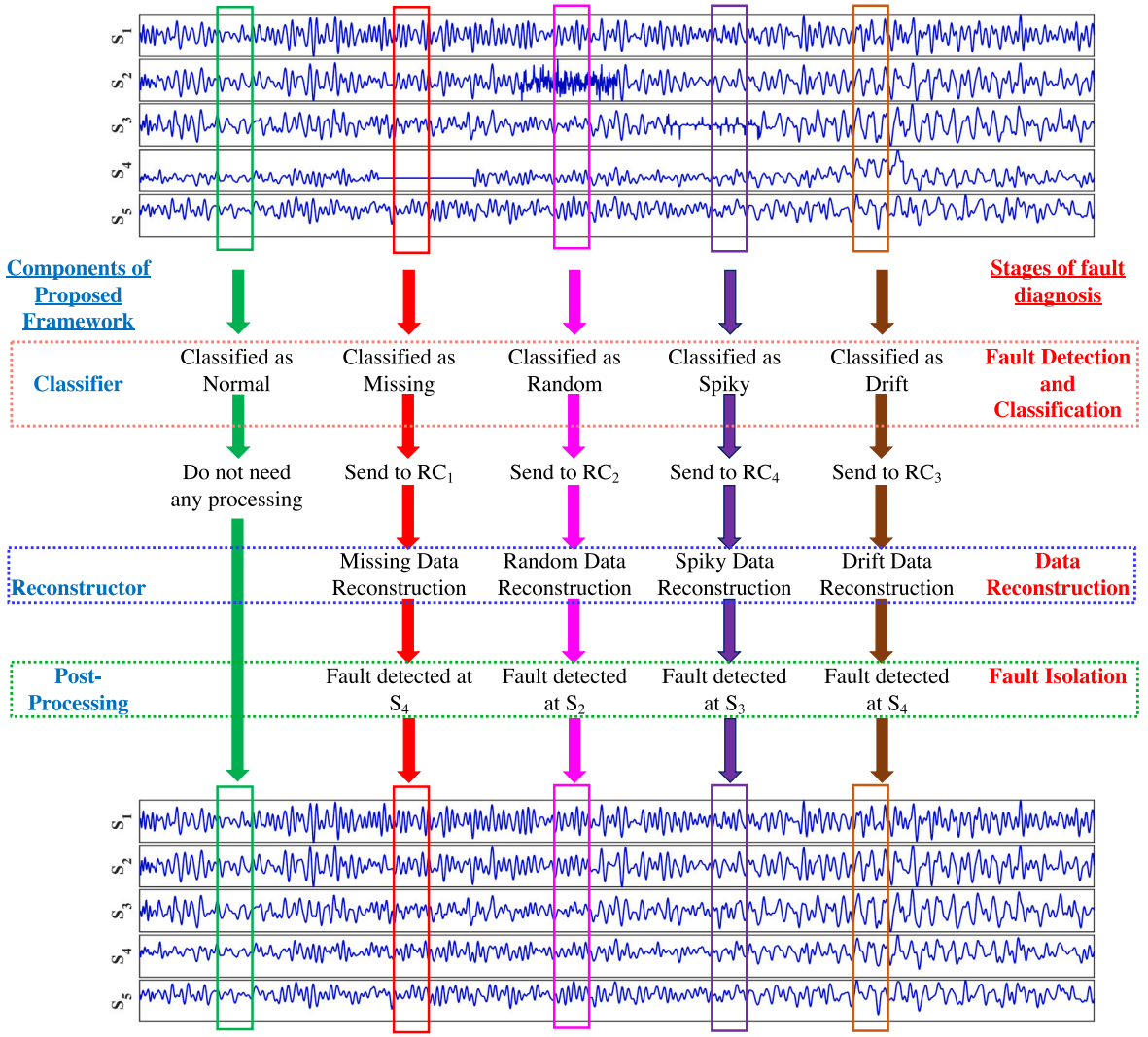

Fig. 3. Proposed framework addressing different stages of fault diagnosis (the fault time histories and the sliding window length are exaggerated for demonstration purpose).

stacked in an array, the sensor measurements are conceptually similar to pixels but with one dimension corresponding to the temporal evolution. When adequately trained, we expect the CNN model to similarly identify latent features from the input sensor data and correctly identify the presence of a fault and its type. We train the CNN model until the loss function converges. While common machine learning algorithms show reasonable accuracy in image classification tasks, they do not fully exploit features from the neighborhood of a particular point at multiple scales, which in our case are the spatial and temporal correlations in the input data.

A typical CNN architecture consists of a stack of convolutional layers that extract the feature maps, max- or median-pooling layers to keep only the relevant features, followed by fully connected layers and a softmax layer that uses the extracted features to perform classification. The convolution operation performs weighted averaging while the max-pooling operation enables downsampling. Our CNN model included two alternate layers of convolution and max pooling, followed by two fully connected layers, and finally, the softmax layer. This architecture was chosen to extract features across multiple time scales and use them for classification. As the faults addressed in this paper are primarily captured as temporal features (e.g., *missing* fault has zero reading along the temporal direction), the kernel size along the spatial direction was set to 1 to avoid feature averaging of data along the spatial dimension. Following the first convolution layer, a max-pooling layer is introduced to retain the most dominant features and to reduce the number of parameters in the network. Another convolutional layer is then added to extract features at higher time scales, which can be critical in some cases. For example, *drift* or *spiky* fault might not be easily perceivable at some smaller time-scale but can be identified at a longer time scale. Using cross-entropy loss as the objective function, we optimize the size of the kernel in the first convolution layer to (11×1) and in the second convolution layer to (5×1), while the filter size for both max-pooling layers is (2×1).

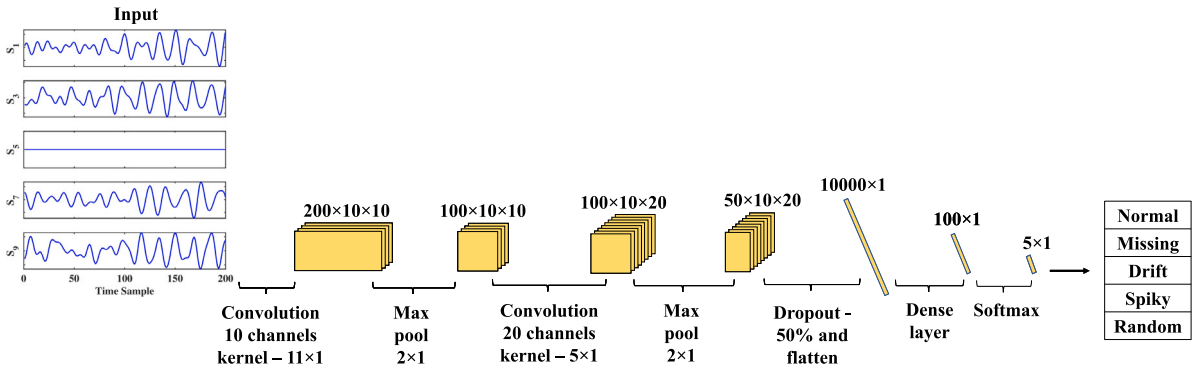

Fig. 4. Architecture of the proposed classifier module (2nd module of the pipeline in Fig. 2). This representative figure shows a windowed response of 10 sensors (only 5 are shown) and S_5 is a faulty sensor with missing fault type. The CNN is trained to classify the type of fault, in this case it will be 'missing'. The input size is (200×10), where 200 is the number of time steps in a window and 10 is the number of sensors. From the continuous input, a time window of 200 is chosen. The size of the time window influences the choice of the sampling frequency and is discussed in detail in Appendices B and C.

Finally, fully connected and softmax layers are added that use the identified features to perform classification. The training process involves the minimization of the categorical cross-entropy loss function [47].

3.3. Reconstructor

An autoencoder (AE) [38] is an unsupervised Deep Learning technique for reconstruction that tries to learn an approximation to the identity function, i.e. $x = f(x)$, where x is the input and $f(x)$ is the output. In the process, autoencoders learn the underlying data structure (often in compressed form) through the bottleneck layer, the encoding of the data into the compressed state through the encoder layers and its decoding to original form through the decoder layers. While AEs work well, they do not exploit the information from the neighborhood of the input data, similar to artificial neural networks. The convolution operation leverages the spatial and temporal data structure in identifying the correlations more easily in comparison to the stacking operation, which other NN architectures use. Convolutional autoencoders (CAE) [48] combine the two concepts, i.e., convolution and autoencoding to identify underlying features in the data while preserving the neighboring relations between the pixels. They have convolutional layers that perform the feature extraction followed by bottleneck layers that use the feature maps and perform the data compression. This is followed by the decoder, which performs the reconstruction. CAEs are therefore much more robust than AEs since they preserve the 2D input data structure and do not force redundancies by forcing each feature to be linked to the global scale, i.e., instead of using entire image information for reconstruction, CAEs can use the local patches for reconstruction [49]. In our work, we take inspiration from the neural network architectures for the problem of neural inpainting [50].

After the classifier identifies the type of sensor data fault, the data is passed through a reconstructor for reconstructing the faulty sensor data. The architecture of all reconstructors corresponding to different types of sensor faults is the same, as shown in Fig. 5. The size of the convolutional kernels is selected by minimizing the loss function and is discussed in Appendix D.3. All four CAE networks (RC_1, RC_2, RC_3, and RC_4) are independently trained for reconstruction. Each CAE model addresses a unique sensor fault type as using a single CAE model with the same architecture to address all faults will have a much lower accuracy. A much larger, single CAE model may also be able to address all types of faults, but the computational cost of training and using such a large model may not be feasible for all applications. The training process involves minimizing the mean-squared error defined as the difference between the ground truth and the network prediction [51].

3.4. Postprocessing

The output from the reconstructor step is a reconstructed data matrix for all the sensors. Although this approach does not warrant identifying the unique faulty sensor, it introduces unintended deviation in the data for all other sensors too. To circumvent this, we identify the faulty sensor using a difference metric and only replace the data for the identified faulty sensor, in the overall output. The difference metric, δ, is characterized in the following way:

$$\delta = \frac{\|\text{input} - \text{output}\|_2^2}{\|\text{output}\|_2^2} \tag{1}$$

where, $\|\cdot\|_2$ denotes the 2-norm and it is defined as $\|\mathbf{z}\|_2 = \sqrt{z_1^2 + z_2^2 + \cdots + z_n^2}$, if $\mathbf{z} = (z_1, z_2, \ldots, z_n)$.

The intuition for the metric develops by defining a threshold limit on the difference between the reconstructed data and the original data for a faulty sensor. This metric can be used to identify the faulty sensor for repair or replacement. The post-processing pipeline is shown in Fig. 6. First, reconstructed data from the reconstructor and the input data are used to calculate δ_i for each sensor, i. Faulty sensors are identified as those sensors with δ_i exceeding the threshold metric δ_{max}, i.e., $\delta_i > \delta_{max}$ for faulty sensors.

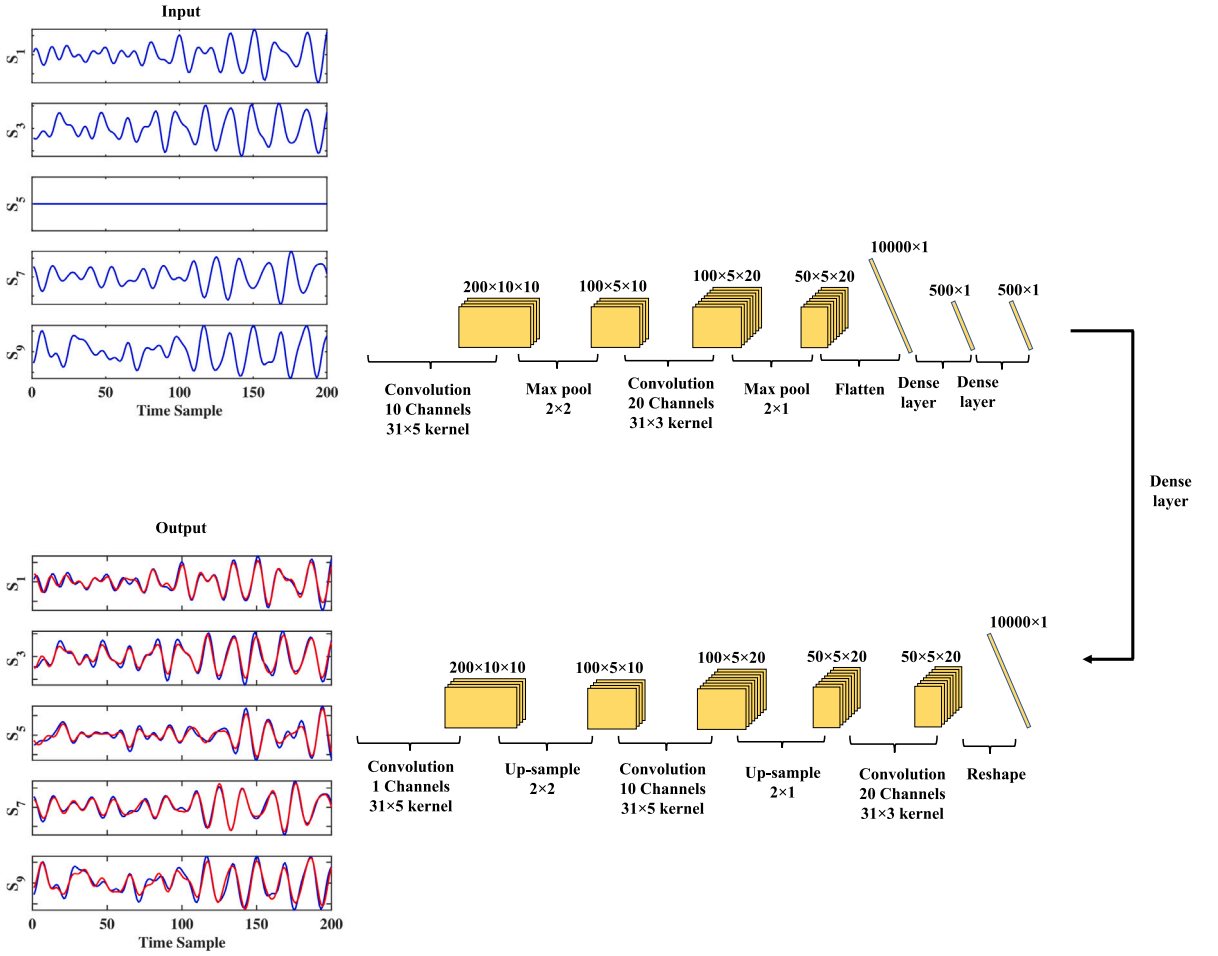

Fig. 5. Architecture of the proposed reconstructor module (3rd module of the pipeline in Fig. 2). This representative figure shows a windowed response of 10 sensors (only 5 are shown) and S_5 is a faulty sensor with missing fault type, which is same as shown previously in Fig. 4. After being classified as 'missing', the dataset is sent to the autoencoder RC_1 for reconstruction of the 'missing' input. As an output, it is shown that the S_5 is showing the response time history. Both the input and the output size is (200×10), where 200 is the number of time steps in a window and 10 is the number of sensors. From the continuous input, a time window of 200 is chosen. The size of the time window influences the choice of the sampling frequency and is discussed in Appendices B and C.

The choice of δ_{max} is dependent on the problem and the level of expected noise in the sensor. We conservatively select $\delta_{max} = 0.2$ for SHM applications based on engineering judgment. The threshold δ_{max} should be increased if the sensor data is expected to have a high level of noise. This is because the output from the CAE will not contain the same level of noise, resulting in higher δ_i even for a *Normal* sensor. We also define a heuristic upper-bound on the number of sensors with concurrent faults that the framework can address, $N_{faulty,max}$, and is set to 40% of the total number of sensors. This is because if more than half the total number of sensors are faulty, then the structural property may have changed, as it is highly improbable that half of the sensors will have faults simultaneously. It is advisable that a detailed investigation be performed to determine the actual cause. At the same time, reconstructing data for more than one-half of the faulty sensors from the remaining *Normal* sensors may lead to inaccurate reconstruction. If the number of faulty sensors, N_{faulty} exceeds $N_{faulty,max}$, or if δ_i is very high for multiple sensors, it is likely that the structural property has changed. In such a scenario, damage detection is recommended.

4. Numerical simulation

The proposed framework can be used for any type of simple or complex time-invariant system. For time-variant systems, the classifier and reconstructor models are not expected to perform well, as the models are trained on a finite dataset not corresponding to the system properties that change with time. A simple 10 DOF linear time-invariant (LTI) shear-type model as illustrated in Fig. 7 is selected as a benchmark to test the efficacy of the framework.

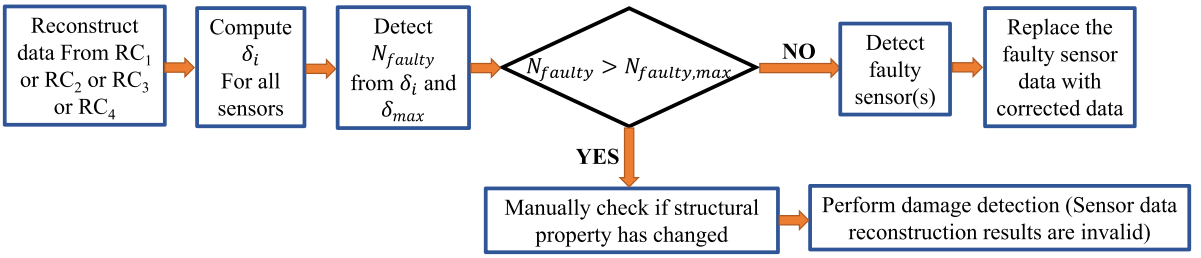

Fig. 6. Postprocessing pipeline. The acceptable number of faulty sensors, $N_{faulty,max}$, and the threshold metric, δ_{max}, are specified by the user according to the type of application.

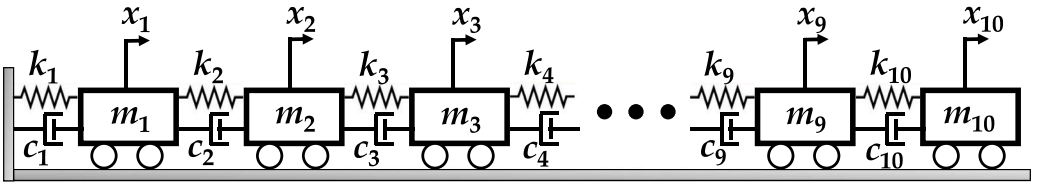

Fig. 7. 10 Degrees of freedom shear type linear time-invariant structural system.

4.1. System description

The equation of motion of a N_d-DOF linear, classically damped system can be described by the following differential equation:

$$\mathbf{M}\ddot{\mathbf{x}}(t) + \mathbf{C}\dot{\mathbf{x}}(t) + \mathbf{K}\mathbf{x}(t) = \mathbf{P}_I\mathbf{u}(t) \tag{2}$$

where **M**, **C**, and **K** are the mass, damping, and stiffness matrices, respectively; and $\mathbf{x} = \{x_1 \ x_2 \ \cdots \ x_{N_d}\}^T$, $\dot{\mathbf{x}}$, and $\ddot{\mathbf{x}}$ are the displacement, velocity, and acceleration responses in physical coordinates respectively. Here, x_i is the displacement corresponding to the ith DOF, m_i ($m_i \in \mathbf{M}$) is the mass of ith DOF, k_i ($k_i \in \mathbf{K}$), and c_i ($c_i \in \mathbf{C}$) are the stiffness and damping of the element joining the ith and $(i-1)$th DOF, with the 0th DOF being the fixed end in the left of Fig. 7. Suppose that there are n_I inputs, $\mathbf{u}$ is the $n_I \times 1$ vector of input forces, and $\mathbf{P}_I$ is a $N_d \times n_I$ binary input location matrix. For any input, the corresponding column of P_I has all elements zero except the element corresponding to the DOF where the input acts; for example, if the ith input is located at DOF p then the ith column of $\mathbf{P}_I$ has all elements equal to zero, except the pth element which is one.

The structural system vibrates when subjected to an external load like ambient wind force or earthquake ground motion. When appropriately instrumented, the sensors associated with the structure's DOFs record the vibration response and can be used to identify the system properties like m_i, k_i, and c_i. Monitoring these variables provides information about the health of the structure.

4.2. Dynamic properties

The mass at each DOF is 100 kg, and the stiffness of each element is 10^6 N/m. The damping ratio of the first two modes is assumed as 1% with classical Rayleigh damping approximation. The values of the system parameters are chosen as a case study, and any other values could also be used instead. The natural frequencies of the system are: 2.37, 7.08, 11.63, 15.92, 19.85, 23.33, 26.3, 28.68, 30.41, and 31.47 Hz. To simulate ambient vibration from wind loads in a tall building, a random force of low frequency (0–20 Hz) is applied to each degree of freedom. Note that the time history of the forcing function is different for each of the DOFs, to simulate the actual ambient force scenario. These ambient random forces have zero mean and a standard deviation of 10 N.

4.3. Data preparation

The responses of each of the DOFs are obtained by numerically solving the differential Eq. (2) for the structural parameters, and the forcing function just described. The data acquisition sampling frequency is 200 Hz and the selected time range/window for single input sample to the framework is 1 s. Hence, the size of a single sample to the framework is 200 × 10 (number of time instances × number of sensors) – we consider all the DOFs are instrumented, with one instance shown in Fig. 9(a). Ten thousand such samples are generated such that there is enough data for training, validation, and testing.

Sensor data faults as described in Section 2 are introduced in the numerically simulated data. For each sample in the *Missing* dataset, a sensor is selected uniformly at random, and its reading is replaced by zeros. A realization of *Missing* dataset is shown in Fig. 9(a) – with faulty sensor at S_6. Similarly, for each sample in *Random* dataset, a sensor is selected uniformly at random, and its value is replaced with random noise. A realization of *Random* dataset is shown in Fig. 14(a) – with faulty sensor at S_4.

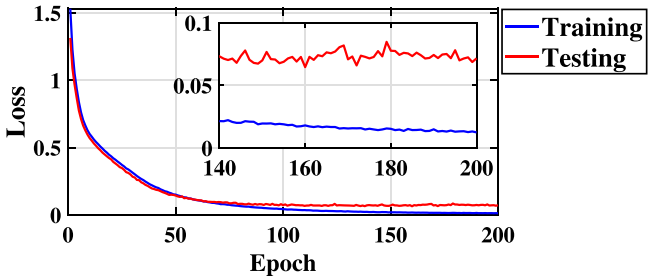

Fig. 8. Evaluated cross entropy loss vs. Epoch during training for training and testing data.

Table 1
Confusion matrix of the CNN classifier (in %).

		Predicted class				
		Normal	Missing	Random	Drift	Spiky
Actual class	Normal	100	0	0	0	0
	Missing	0	100	0	0	0
	Random	0	0	100	0	0
	Drift	4.3	0.2	0.15	95.35	0
	Spiky	1.85	0	0	0	98.15

For constructing the *Drift* dataset, the bias and slope of the mean trendline are determined by sampling a random number between -1 and 1. Such sample *Drift* data is shown in Fig. 15(a), where the faulty sensor is sensor number S_{10}.

Finally, for the *Spiky* dataset, the number of spikes in the time series can be random. This implies that the probability of a spike occurring is uniformly random between 0 to 1 at a time step for a given sensor. The amplitude of the spike is also assumed to be uniformly random between 0 to 10. A sample *Spiky* data is shown in Fig. 16(a), where the faulty sensor is sensor number S_6.

4.4. Classification

The entire dataset containing 50,000 samples represents four fault type classes and the normal data type class, with each class having 10,000 data samples, and it is used for five class classification. In most classification problems with real datasets, there may be a class imbalance issue. As the data selected for training needs to be balanced to prevent bias [52], all classes must in similar proportion. If this results in the selected dataset becoming too small, data augmentation techniques can be used to increase the available training data [53,54]. In the case of fault detection and classification, data augmentation can be done by manually adding corruption to normal data to produce corrupted data of different fault types. A CNN model with architecture as described in Section 3.2 is used. The data is randomly divided into 72% for training, 8% for validation, and 20% for testing, which is a common split ratio for deep learning [55,56].

The weights of all hidden units are initialized using Xavier initialization [57]. This is a common technique that initializes the weights of the neural network such that the variance of the activations in every layer is the same. The equal variance for every layer prevents the exploding or vanishing gradient problem. We have used the Adam Optimizer [58] for training the CNN. Adam or Adaptive Moment Estimation is a variant of stochastic gradient descent optimizer and it is very commonly used in deep learning. Categorical cross-entropy is used as the loss function. We fix the number of epochs in training to 200 and optimize the learning rate, batch size, and dropout hyperparameters of the CNN training using grid search implemented with the Talos [59] package. Talos is a python package that automates hyperparameter tuning and model evaluation. We used Talos to implement the grid search technique of hyperparameter optimization. The hyperparameter optimization yields an optimal learning rate of 0.0001, batch size of 500, and a dropout ratio of 40% (Appendix D.1). Fig. 8 shows the evolution of the evaluated loss, with each epoch during training, for the training, and testing datasets.

The confusion matrix for the testing data is shown in Table 1. The diagonal values of the confusion matrix indicate the classification accuracy tested on 2000 samples. These diagonal values are used in calculating the average classification accuracy as 98.70%. For *Normal*, *Missing* and *Random* data classes, the accuracy is 100% which suggests that these classes have unique, distinguishable features that make them easy to classify. Some test samples from *Spiky* and *Drift* classes are misclassified as *Normal* data. As described in Section 4.3, the occurrence and amplitude of spikes and the bias in *drift* are random. This results in some samples being similar to and indistinguishable from *Normal* data.

4.5. Reconstruction

For each type of sensor fault, a separate CAE model is trained to achieve the highest possible reconstruction accuracy. Similar to the CNN model, all of the data is randomly divided into 72% for training, 8% for validation and 20% for testing according to a

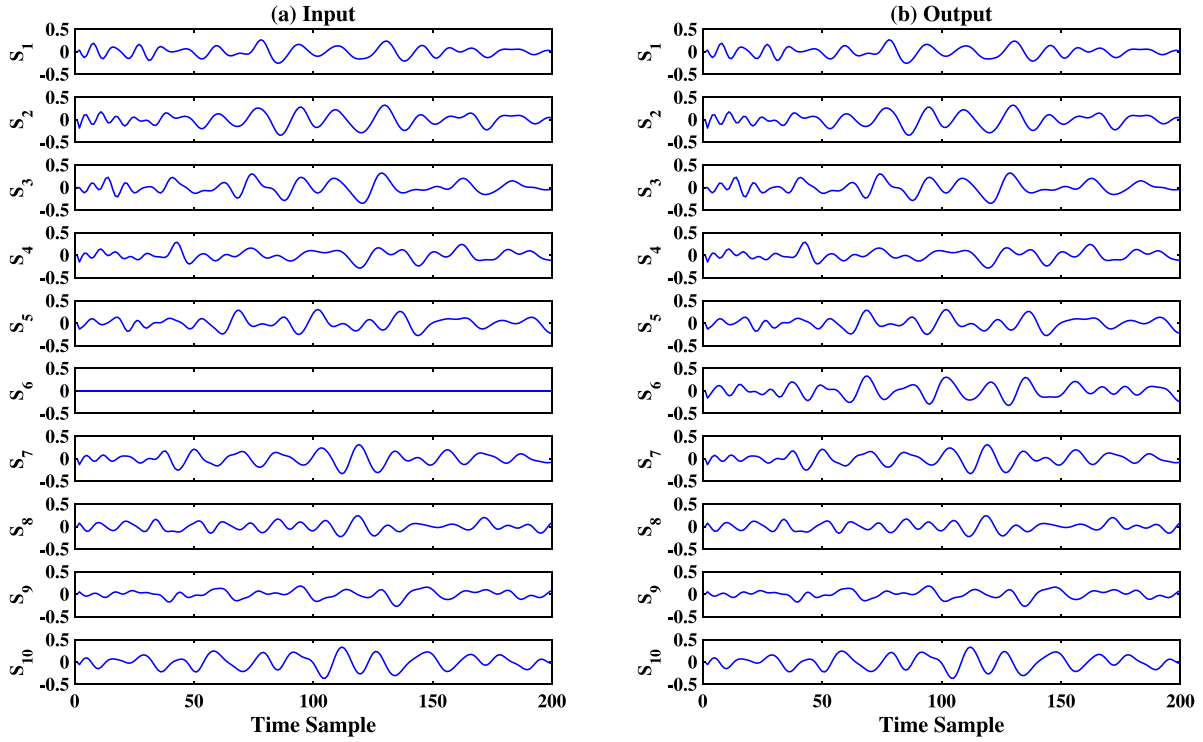

Fig. 9. Sample training data (acceleration responses in m/s^2) for CAE-RC_1 — (a) Input (*Missing* Fault in Sensor S_6); (b) Output (ground truth data — with actual time history available for Sensor S_6).

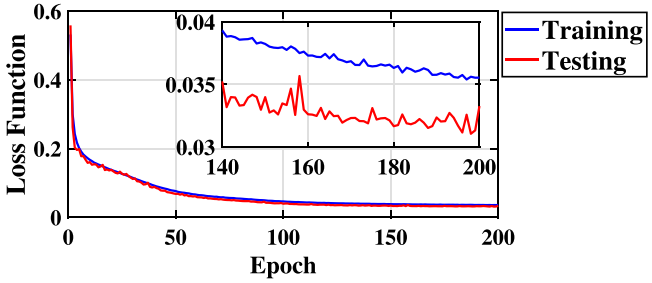

Fig. 10. Evaluated Mean Squared Error (MSE) loss function of CAE-RC_1 vs. Epoch during training for training and testing data.

common convention in deep learning practice [55,56], viz., eight thousand training samples (80% of the total data) representing eight thousand seconds of data are used for training each individual model. The result for reconstructing *missing* fault is discussed below. The process and results for other types of faults show similar trends. The input and output corresponding to other faults are discussed in Section 4.8. The architecture for the CAE models is described in Section 3.3.

4.5.1. Missing sensor data reconstruction

The input to the CAE reconstructor model for missing data is a (200×10) matrix where one of the columns has missing data (or zero). The output has the same dimension as the input and is the ground truth from the *Normal* dataset. The sensor with *Missing* fault is selected uniformly at random from one of the 10 DOFs. One instance of such input and output with a fault in sensor number 6, S_6, is shown in Fig. 9(a) and (b) respectively.

As with the classification task, learning rate, batch size, and dropout ratio were selected as the hyperparameters of the CAE training to be optimized using grid search implemented using the Talos [59] package, and these details are discussed in Appendix D.1. The weights and biases of the hidden units are initialized using Xavier initialization [57] and updated using the Adam Optimizer [58] during training. Hyperparameter tuning yielded a learning rate of 0.0001, batch size of 50, and dropout ratio of 40%. Mean squared error is used as the loss function to be minimized during training, and the evolution of the reconstructor loss function is shown in Fig. 10.

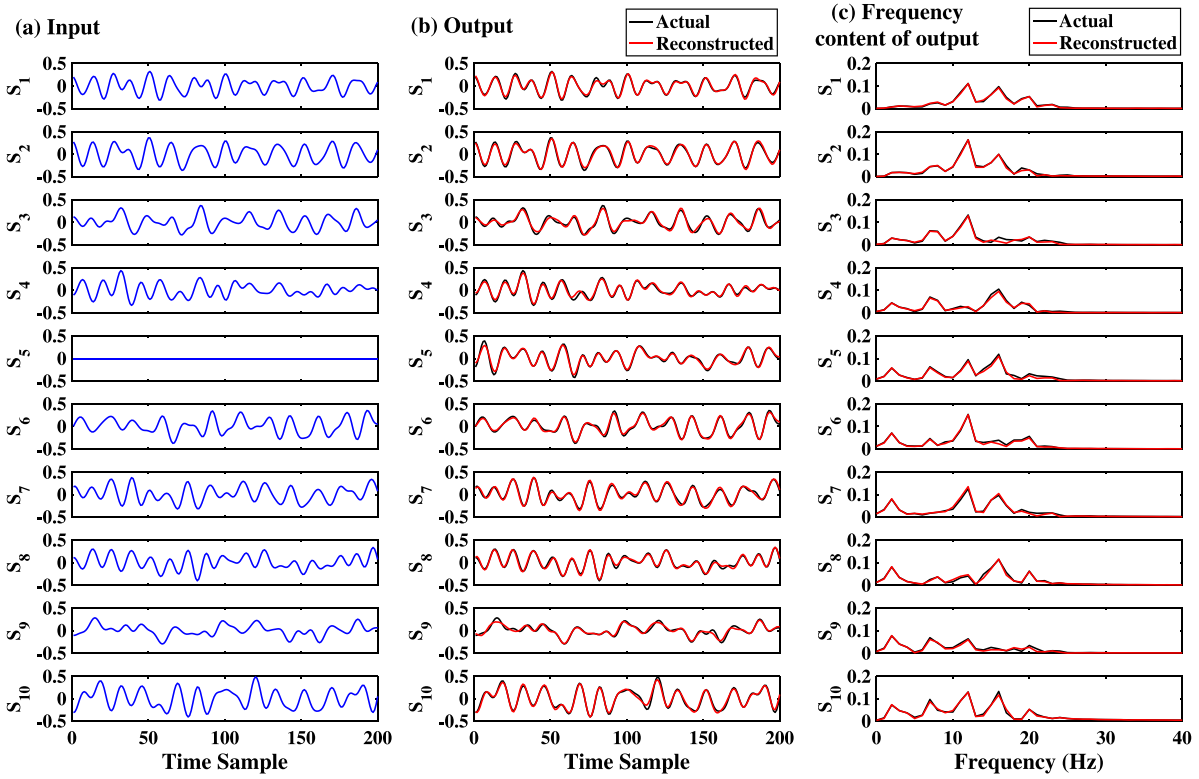

Fig. 11. Sample testing data (acceleration responses in m/s^2) for CAE-RC_1 – (a) Input (*Missing* Fault in Sensor S_5); (b) Output (reconstructed time history for Sensor S_5) - as the reconstructed time history of the healthy sensors is unwarranted, it will be replaced with the original, unprocessed sensor data in the post-processing step; (c) Frequency content (m/s) of the actual and reconstructed data in Fig. 11(b).

Post-training, the CAE is tested on the unseen testing data. One sample from the unseen testing data is shown in Fig. 11(a) where the response data from sensor 5, S_5, is missing. The CAE reconstructor predicts the response data for all sensors, as shown in Fig. 11(b) in red color. The actual ground truth is also shown for comparison in the same figure in blue color. The frequency content of the reconstructed data and the actual data (ground truth) also reasonably match, as shown in Fig. 11(c).

4.6. Post-processing

As seen in Fig. 11, the CAE reconstructs the time history of the faulty sensor as well as the healthy sensors. However, as the reconstructed time history of the healthy sensors is unwarranted, it should be replaced by the original, unprocessed sensor data. To this end, the difference metric, δ as defined by Eq. (1) is computed for each sensor and is used to identify the faulty sensor(s) data and the fault-free sensors. By computing this difference metric for the sample shown in Fig. 11(a) and (b), Sensor S_5 was found to contain the *Missing* fault. The difference metric δ for each sensor is shown in Fig. 12. As δ for S_5 exceeds the threshold limit of $\delta_{max} = 0.2$, the time history of only the faulty sensor S_5 is replaced with the reconstructed time history, while the time history of the remaining sensors is kept intact. The final output after this is shown in Fig. 13(b). The accuracy of localizing the faulty sensor for the entire testing dataset was 100%.

4.7. Reconstruction error of the framework

The relative error is defined as the ratio of the 2-norm of the error matrix and the 2-norm of the true measurement matrix for each training or testing sample. The 2-norm is calculated as

2-norm of matrix **X**: $\|\mathbf{X}\|_2 = \max(\text{svd}(\mathbf{X})) = \sqrt{\lambda_{max}(\mathbf{X}'\mathbf{X})}$

Here, $\text{svd}(\cdot)$, $\max(\cdot)$, and $\lambda_{max}(\cdot)$ denote the Singular Value Decomposition (SVD), maximum value, and the maximum eigenvalue, respectively. The dimension of reconstructed data matrix X represents (number of time steps × number of sensors). The dimensions of training X matrix are [(200 × 8000) × 10] or [1,600,000 × 10] while the dimensions of testing X matrix are [(200 × 2000) × 10] or [400,000 × 10]. This is because the lengths of the time windows for training and testing are 8000 and 2000 respectively.

The error matrix is generated by subtracting the reconstructed data matrix from the actual ground truth data matrix. If **E** represents the error matrix and **A** represents the actual ground truth matrix, the 2-norm of the relative error is given by

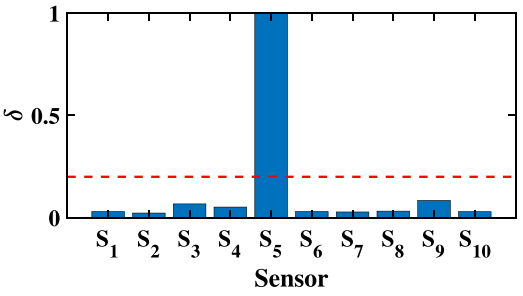

Fig. 12. Faulty sensor localization using difference metric δ.

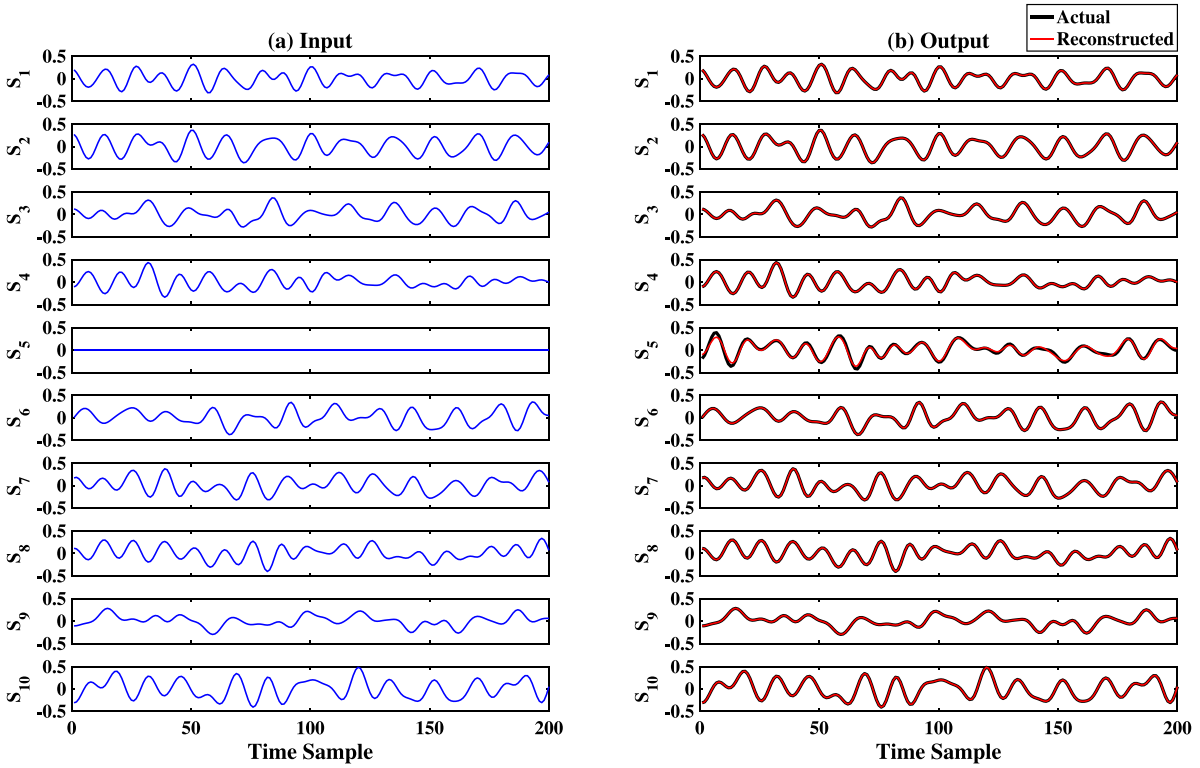

Fig. 13. (a) Sample input to the framework (m/s^2); (b) Final output of the framework (m/s^2) – it detects the presence of fault, the type of fault as *missing*, the location of fault at S_5, and reconstructs the time history.

$\|RE\|_2 = \frac{\|\mathbf{E}\|_2^2}{\|\mathbf{A}\|_2^2} \times 100$. The error metric used is Mean Squared Error (MSE), which represents the fraction of the energy of the error in the signal compared to the energy of the original signal. The 2-norm of the training and testing errors are summarized in Table 2. The testing error for the missing fault was 0.2657%.

4.8. Comparison of reconstruction error of the framework for different fault types

Using the proposed framework, the faulty sensor data is reconstructed for *Random*, *Drift*, and *Spiky* fault types and the results are shown in Figs. 14–16 respectively.

Table 2 summarizes the reconstruction errors for all the sensor reconstruction cases. The average test error over all the fault types is 0.2305%. As expected, the training errors are marginally better than the testing errors, and both are of the same order, which supports the generalization of the network. We observe that errors in *Missing* and *Random* fault types are larger than *Spiky* and *Drift* fault types. *Missing* and *Random* fault-types are more difficult to reconstruct as these fault-types have no information regarding the true signal. *Spiky* fault type has limited information about the true sensor data as there are jumps at arbitrary temporal points. *Drift* is easiest to reconstruct as it has all the information regarding the system but also contains additional trends and bias. For all cases, the reconstruction error is very small, indicating a sufficiently accurate signal reconstruction.

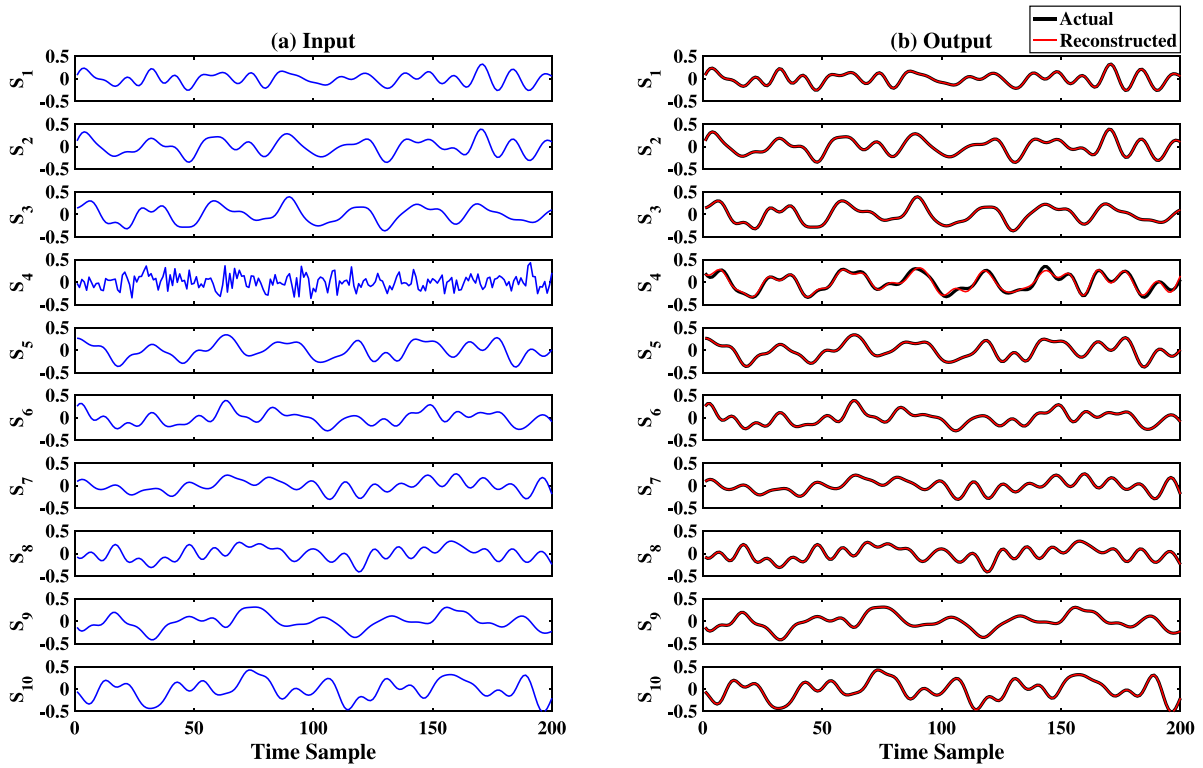

Fig. 14. (a) Sample input to the framework (m/s^2); (b) Final output of the framework (m/s^2) – it detects the presence of fault, the type of fault as *random*, the location of fault at S_4, and reconstructs the time history.

Table 2
$||RE||_2$ for each type of sensor data fault.

Fault type	Training error (%)	Testing error (%)
Missing	0.2121	0.2657
Random	0.1884	0.2408
Drift	0.1335	0.1811
Spiky	0.1942	0.2346

If an input sample to the framework gets misclassified as *Normal* it does not get corrected and is directly passed as output. If it gets misclassified to have any other type of fault, then the framework attempts to correct it using the reconstructor corresponding of the misclassified fault-type. As Table 1 indicates, this issue of misclassification arises primarily for the *Drift* and *Spiky* fault-types in our benchmarks. As Table 2 shows, the overall accuracy of reconstruction, which includes the misclassified samples, is still quite good.

4.9. Performance of the framework for multiple faults

Common sources of sensor damages are due to shock impacts, excessive temperature, electro-static discharge, and mispowering [60]. The most common cause of sensor failure is excessive temperature. As sensor fault occurs due to some ambient situation, it is most likely that multiple sensors will experience similar type of fault. Therefore in this paper, first we have addressed multiple and concurrent sensors faults but of similar types. We have shown the cases where the number of faulty sensors is up to 5 among 10 sensors. We have also addressed two concurrent sensor faults with two different fault type — which is itself a highly improbable event. We did not explore more than two concurrent faults with different fault types as probability of occurrence of that event is negligible.

It is important to consider that we use models trained on data with single fault types to also address possible multiple faults. While new models can also be used to address each combination of faults, this combinatorially increases the number of models to be trained. This tradeoff between computational effort and accuracy can be individually adjusted by a user in their implementation of our framework.

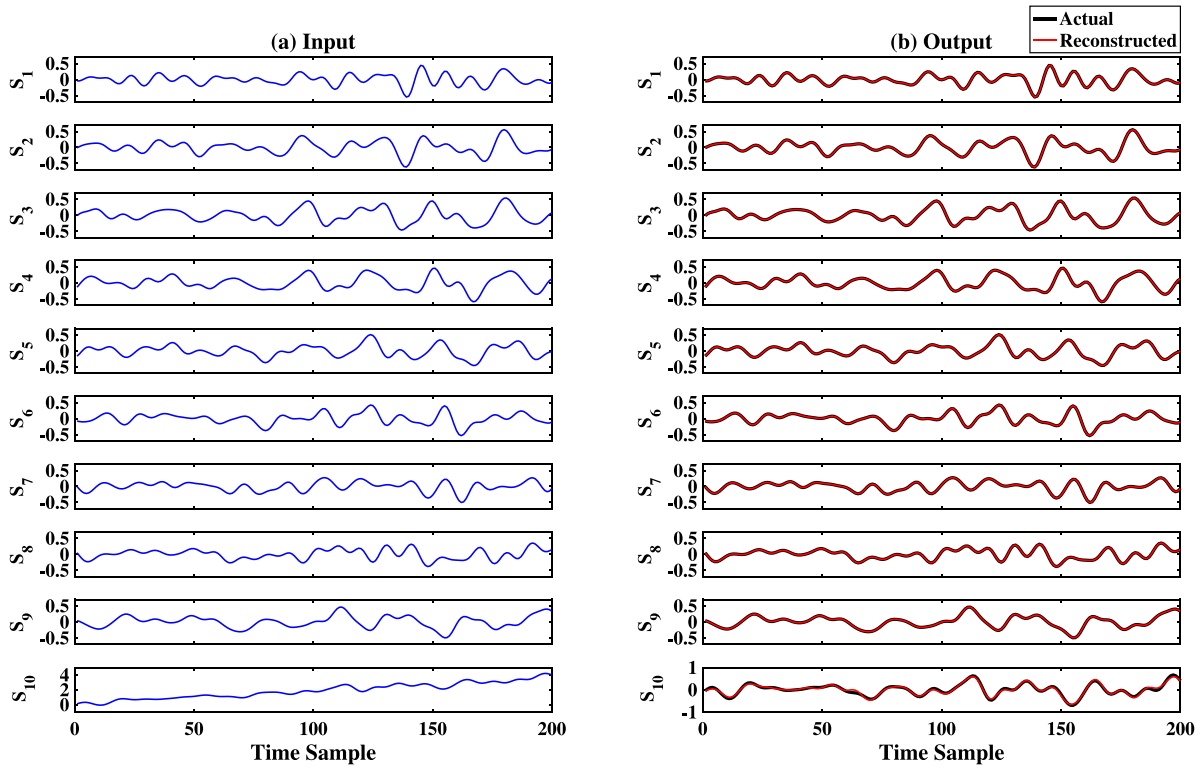

Fig. 15. (a) Sample input to the framework (m/s^2); (b) Final output of the framework (m/s^2) – it detects the presence of fault, the type of fault as $drift$, the location of fault at S_{10}, and reconstructs the time history.

Table 3
$||RE||_2$ for same-type multiple faults (in %).

Number of faults	Trained on 1 missing fault and tested on multiple missing faults	Trained on 1 random fault and tested on multiple random faults	Trained on 1 drift fault and tested on multiple drift faults	Trained on spiky fault and tested on multiple spiky faults
1	0.2570	0.3319	0.1626	0.1959
2	0.5852	0.7966	0.2510	0.4547
3	1.5351	1.6945	0.4414	1.0243
4	3.4664	3.7605	0.6340	2.0291
5	7.1720	8.1913	0.8669	4.2884

4.9.1. Multiple faults of the same types

While all the models in the framework are trained to reconstruct sensor data with the fault in a single sensor, we test the ability of the models to address multiple, concurrent faults of the same type. For generating data with faults in multiple sensors, sensors are selected uniformly at random, and the faults are introduced. For example, for introducing the missing fault in two sensors, they are selected at random out of the ten total sensors, and the corresponding sensor readings are replaced by zeros. For a combination of fault types, say one missing and one random, two sensors are selected at random, one sensor reading is replaced with zeros, and the other sensor reading is replaced by random noise.

The proposed framework is able to perform reasonably well as the CAEs preserve the latent representation of the sensor data. The errors for various cases with 1 to 5 faults of different types are reported in Table 3. As expected, the reconstruction error increases as the number of faults increase. However, the average error for the entire data matrix for 2, 3, 4 faults out of 10 sensors are 0.5218%, 1.1738%, and 2.4725%, respectively. The classifier detects multiple faults of the same type with 100% accuracy even though it was trained on a single fault. This is particularly useful in large, real-world applications because it circumvents training a large number of models that can address faults in specific combinations of sensors.

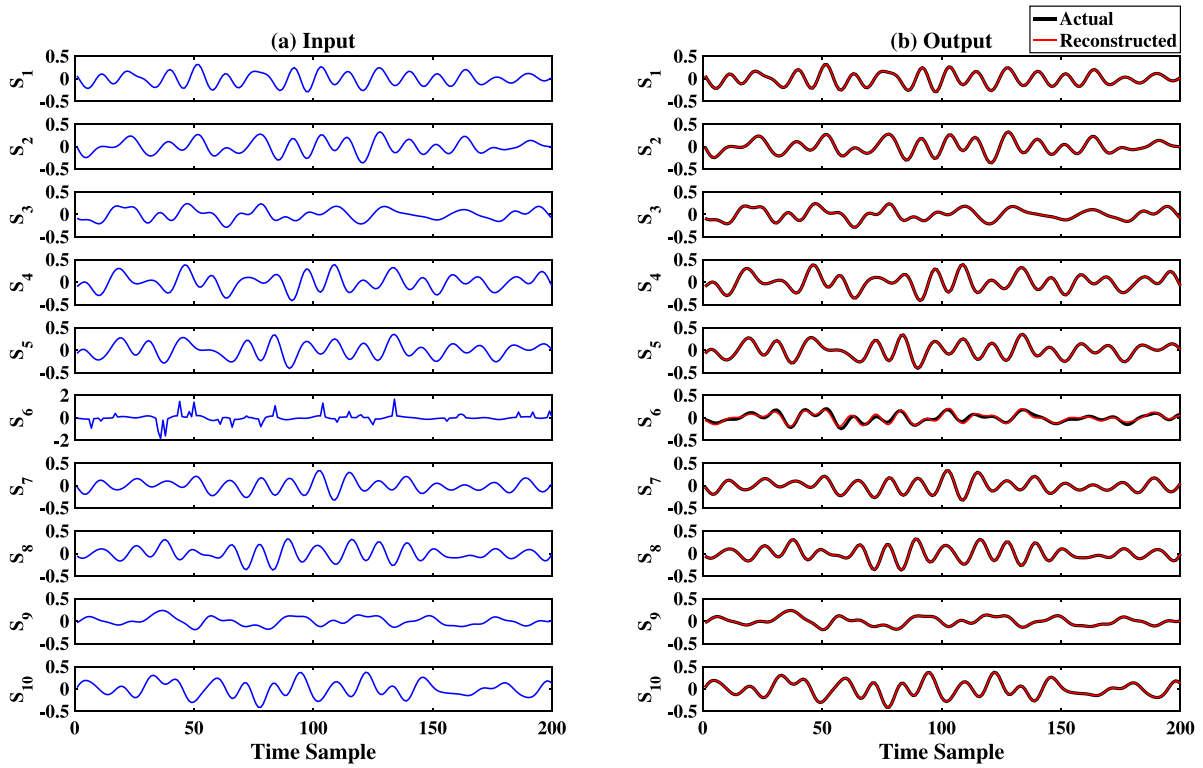

Fig. 16. (a) Sample input to the framework (m/s^2); (b) Final output of the framework (m/s^2) – it detects the presence of fault, the type of fault as *spiky*, the location of fault at S_6, and reconstructs the time history.

Table 4
Classification of combination of multiple faults (in %).

Combination of fault types	Predicted fault types			
	Missing	Random	Drift	Spiky
Missing & random	0	98.7	0	1.3
Missing & drift	6.9	0	93.1	0
Missing & spiky	0	85	0	15
Random & drift	0	99.3	0	0.7
Random & spiky	0	84.6	0	15.4
Drift & spiky	0	1.4	9.3	89.3

4.9.2. Multiple faults of different type

In a real-world application, multiple sensors may have faults of different types. We examine such scenarios using an example case of 2 sensor faults of different types. The results of classification are summarized in Table 4 and the results of reconstruction are reported in Table 5.

For each type of fault combination, the classifier will predict a single, distinct fault type, thus invoking the corresponding reconstructor. Table 4 demonstrates that for each combination, there is a corresponding type of fault that is statistically more likely to be selected. In Table 5, the $\|RE\|_2$ for these likely to be predicted fault types in each row are in bold. For example, as per Table 4, the classifier predicts that the Missing and Drift combination as a *Drift* fault, for a majority 93.1% of the cases. Therefore, in Table 5 the error corresponding to the drift reconstructor, 0.8338%, is in bold. From Table 4, we also observe that fault combinations containing the *random* fault, are classified as *random* fault in the majority cases. And for these combinations, the *random* reconstructor reconstructs the sensor data with an average error of 2%. These results demonstrate that our end-to-end framework works well for any combination of multiple faults.

4.10. Comparison of the proposed framework with benchmark methods

We compare the performance of our framework with other benchmark methods for sensor data reconstruction. Principal Component Regression (PCR) and Partial Least Squares Regression (PLSR) are both multivariate regression methods [61]. In the Multiple Linear Regression (MLR) method, the solution for $\mathbf{Y} = \mathbf{XB} + \epsilon$ is given by $\mathbf{B} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$. However, $\mathbf{X}^T\mathbf{X}$ is often singular

Table 5
$||RE||_2$ for combination of multiple fault (in %).

Combination of 2 different faults	Trained on 1 missing fault	Trained on 1 random fault	Trained on 1 drift fault	Trained on 1 spiky fault
Missing & random	0.8963	**0.8019**	1.7837	1.9995
Missing & drift	8.5117	2.7629	**0.8338**	1.4263
Missing & spiky	4.7083	**1.7012**	2.7306	2.0130
Random & drift	8.4075	**3.1774**	0.8936	1.6019
Random & spiky	4.9250	**1.9591**	1.4894	0.9031
Drift & spiky	11.1806	4.0644	1.3723	**0.9152**

Table 6
Comparison with other available technique.

Algorithm	Training error (%)	Testing error (%)
Partial Least Square Regression (PLSR)	3.4794	3.5022
Principal Component Regression (PCR)	1.587	1.5774
Support Vector Regression (SVR)	1.4077	1.5048
LSTM–RNN	1.7027	1.7722
Bidirectional RNN	2.1404	2.1666
Proposed Convolutional Autoencoder	2.4481	3.2027

due to collinearities or when the number of datapoints are less than the number of variables. PCR and PLSR decompose $\mathbf{X}$ as $\mathbf{X} = \mathbf{TP}$ where $\mathbf{T}$ represents the orthogonal scores and $\mathbf{P}$ represents the loadings. The regression is then performed on the first a columns of $\mathbf{T}$ instead of $\mathbf{X}$. This avoids the problem of $\mathbf{X}^T\mathbf{X}$ being singular. In PCR, $\mathbf{T}$ is given by the left singular vectors of $\mathbf{X}$ scaled by the singular values while $\mathbf{P}$ are the right singular vectors of $\mathbf{X}$. Various versions of PLSR capture information from $\mathbf{X}$ as well as $\mathbf{Y}$ in calculating $\mathbf{T}$ and $\mathbf{P}$. We use the SIMPLS version [62] of PLSR which chooses $\mathbf{T}$ and $\mathbf{P}$ in such a way that describes the covariance between $\mathbf{X}$ and $\mathbf{Y}$ as much as possible [61].

Introduced by Drucker et al. [63], Support Vector Regression (SVR) is another regression technique which aims to minimize the coefficients of the mapping between input $\mathbf{X}$ and output $\mathbf{Y}$ such that the prediction error is within a specified limit [64]. If x_i is the input, β_i are the coefficients corresponding to each (x_i, y_i) pair, $k(\cdot)$ is the Radial Basis Function (RBF) kernel given by $k(u,u') = \exp\left(-\frac{\|u-u'\|^2}{2\sigma^2}\right)$, and b are the biases, then the RBF based SVR function, $f(\cdot)$, is given by:

$$f(x) = \sum_{i=1}^{N} \beta_i k(x_i, x) + b \tag{3}$$

We also implement an LSTM and a BRNN model, described in the introduction, for comparison. We have chosen the aforementioned techniques as the benchmarks because BRNN is the state-of-the-art technique for sensor data reconstruction, and LSTM–RNN is a conventional and popular deep learning architecture that can be used for such reconstruction. Also, the machine learning algorithms PLSR, PCR, and SVR are traditional techniques for regression that can be used for reconstruction. These state-of-the-art (SOTA) techniques were used as benchmark methods for reconstruction.

The error metric used to quantify the performance for all the techniques is calculated as below.

$$\epsilon = \frac{\|\text{Exact Response} - \text{Reconstructed Response}\|_2^2}{\|\text{Exact Response}\|_2^2} \times 100 \tag{4}$$

Among the ten sensors associated with the 10 DOF of the structural system, the 5th sensor is faulty with say *drift* fault. Since none of the benchmark algorithms can identify the faulty sensor or type of fault directly, this information is assumed given for the algorithms to reconstruct the correct signal from the data from the other nine sensors. The reconstruction errors are summarized in Table 6. The reconstruction errors are comparable to the SOTA methods, but they need more models to achieve similar accuracy as our proposed method. These SOTA methods require separate models for combinations of faulty sensors, a total of $\sum_{i=1}^{m} \binom{n}{i} i^r$ models need to be trained, where n is the total number of sensors, m is the maximum number of faulty sensors, and r is the maximum number of types of faults considered. This combinatorially large number of models warrants a large amount of data and computational cost for building the models. Machine learning methods like PCR and SVR are agnostic to fault type and will thus require a total of $\sum_{i=1}^{m} \binom{n}{i}$ models to address all possible combinations of faults. In contrast to these methods, our framework requires only r models to address multiple faults. Further, other SOTA methods require the detection and identification of faulty sensor apriori but with this framework, we do not need to identify the faulty sensor.

5. Experimental validation

5.1. System description

We further validate the proposed deep learning framework using experimental data from an arch bridge model. This arch bridge model is a simplified model of the arch bridge across US highway 59 and Mandell street in Houston, Texas. An image of this bridge

Fig. 17. Arch bridge across US 59 and Mandell St [65].

is shown in Fig. 17. The steel arch has nine pairs of steel hanger cables to support the bridge deck. The deck is made of concrete which is supported by underlying steel girders, and it is simply supported at both abutments.

The simplified physical arch bridge model is built using PASCO advanced structures model sets ME 6992 [66]. The model is shown in Fig. 18. The steel arch and the concrete deck are modeled using plastic components, while the steel cables are modeled using nylon strings. The elevation of the plastic bridge model is shown in Fig. 19. The distance between the abutments is 1.27 m.

A single MB Dynamics (MODAL 50A) mass shaker [67] is attached to the node next to the right abutment to produce a wide range random noise to simulate the forcing function as shown in Figs. 18 and 19. Crown CE 2000 power amplifier is used to vary the amplitude of the forcing function, which results in varying response amplitude. Eight accelerometers of PCB Model 333B32 [68] are attached to the plastic deck at the points where the hanger cable supports the deck. National Instrument's (NI) Data Acquisition System (DAQ) Signal Express [69] is used to collect the sensor data.

5.2. Data acquisition and training

For this simplified plastic arch bridge model, the natural frequency of the system is quite high, with the first three natural frequencies being 71.72, 123, and 174.2 Hz. Hence the frequency content of the random forcing function for excitation was selected to be between 0–200 Hz. The data acquisition sampling frequency of each accelerometer was 2000 Hz, and the selected time window was 200 time samples or 0.1 s. Thus, the input data matrix for our proposed framework has a size of 200×8, where 8 is the total number of sensors with one sensor placed at each DOF. Ten thousand such samples were collected and labeled as *Normal* Data as all the sensor measurements were free from any error.

We numerically simulated sensor data faults for this experimental dataset using the same approach that was used in the numerical simulations as described in Section 2 and Section 4.3. Each type of dataset (*Normal*, *Missing*, *Random*, *Spiky*, *Drift*), had 10 000 samples. The only difference between the classifier and reconstructor models for the experimental study, and the models for the numerical study, is the number of sensors, which also correspond to the degrees of freedom. The shear model used in the numerical study has 10 DOFs, while the experimental arch bridge has 8 DOFs.

5.3. Classification

The architecture for the CNN classifier is the same as described in Section 3.2. The hyperparameters of the CNN as well as CAE training were tuned using the Talos [59] package and are summarized in Table D.15 of Appendix D.4. The evolution of the loss function with epochs is shown in Fig. 20.

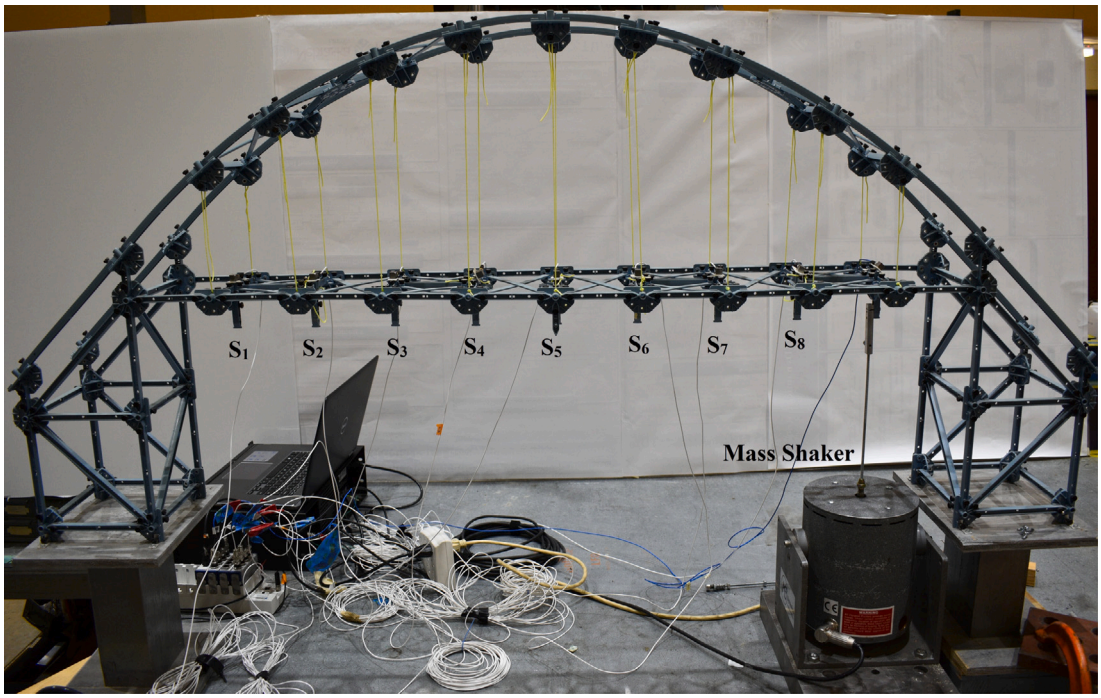

Fig. 18. Simplified experimental model: plastic arch bridge; the mass shaker produces random force to shake the structure.

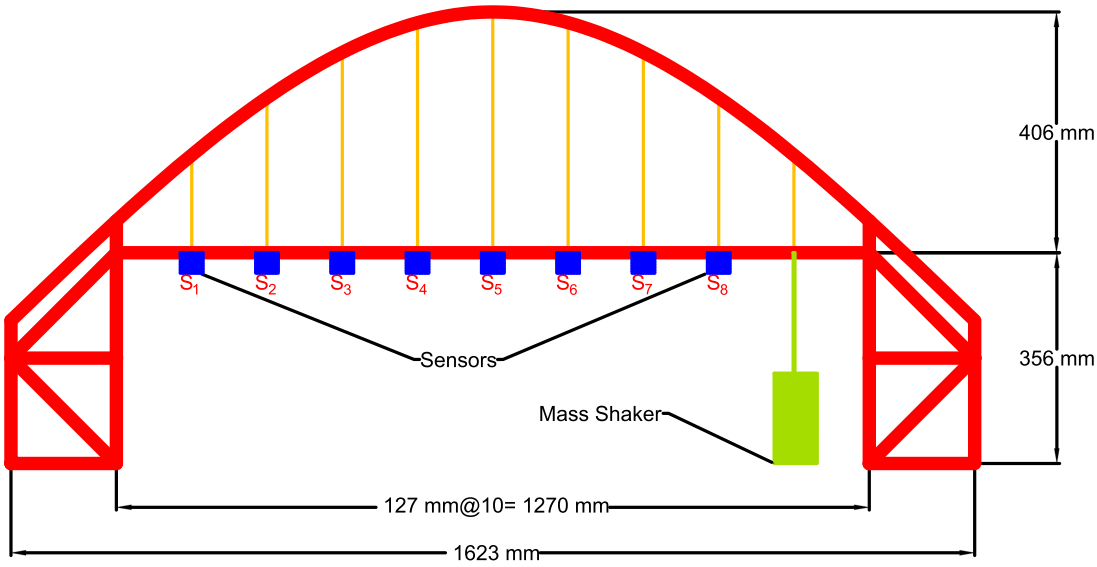

Fig. 19. Simplified experimental model elevation.

The confusion matrix for predictions on the testing data is shown in Table 7. The average classification accuracy is 98.74%, which is obtained by taking the mean of the percentage accuracy reported in the diagonal entries of the confusion matrix. The accuracy of classification is 100% for the *Missing* fault type, while *Normal* and *Random* faults also have a very high accuracy. Similar to the trends in the results from the numerical simulation benchmark, a small number of samples from the *Drift* and *Spiky* classes get misclassified as *Normal*. This may be due to the low-level of error in these samples, making them indistinguishable from *Normal* data.

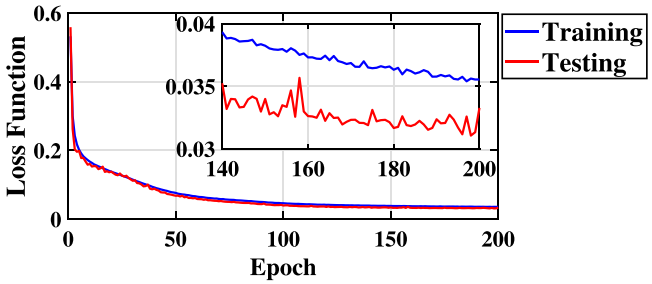

Fig. 20. Evaluated cross entropy loss vs. Epoch during training, for training and testing data.

Table 7
Confusion matrix of the CNN classifier (in %). The diagonal values represent the accurately classified labels while the off-diagonal values represents the mis-classified labels.

		Predicted class				
		Normal	Missing	Random	Drift	Spiky
Actual Class	Normal	99.85	0.15	0	0	0
	Missing	0	100	0	0	0
	Random	0	0	99.95	0.05	0
	Drift	3.7	0.05	0	96.25	0
	Spiky	2.55	0	0	0	97.45

5.4. Reconstruction

Similar to the numerical simulation benchmark, a separate CAE model (Section 3.3) is trained to reconstruct each type of sensor data fault. We discuss the results from the reconstruction of *random* faults by CAE-RC$_2$ – which is a designated reconstructor for *random* faults.

The inputs for the CAE-RC$_2$ for random sensor data faults are (200×8) matrices, where one of the columns contains random noise. An example input is shown in Fig. 21(a) where S_8 reports random data instead of the actual structural response. During training, the ground truth, which is the *Normal* data, as shown in Fig. 21(b), is used as the output for the CAE-RC$_2$.

The loss function for the reconstructor decreases with the number of epochs and converged to a minimum as shown in Fig. 22.

The trained CAE-RC$_2$ model is used to reconstruct the corrected data for the testing data containing the random sensor data fault. A sample from the test dataset is shown in Fig. 23(a) with *Random* sensor data fault at S_1. The CAE-RC$_2$ model generates the corrected output data as shown by the red curves in Fig. 23(b). The actual (ground truth) data of the sensors in blue, matches well with the reconstructed data. The frequency content of the reconstructed data and the actual data shown in Fig. 23(c) indicate that the frequency peaks and amplitude of the reconstructed data are also close to the actual data.

Using the difference metric δ described in Section 3.4, the faulty sensor is identified as S_1 as is evident from Fig. 24. Hence, only the data corresponding to S_1 is replaced with the reconstructed data while the time history of other sensors is unchanged. The final output of the framework after this step is shown in Fig. 25(b). The accuracy of faulty sensor data localization was 100%, similar to the results from the numerical simulation benchmark.

Relative error norms are calculated using the formulae described in Section 4.5.1 for each of the training and testing samples. The 2-norm error for training and testing data were 0.7503% and 0.9166%, respectively, which suggests that the reconstruction results were reasonably good.

5.5. Performance of the proposed framework for other fault types

We performed similar operations for the other types of sensor data faults (*Missing*, *Drift*, and *Spiky*). Sample cases from each fault type are shown in Figs. 26(a), 27(a), and 28(a) where the sensor fault is at S_5, S_6, and S_5 respectively. The corresponding reconstructed time histories using the proposed framework are shown in Figs. 26(b), 27(b), and 28(b) respectively. Reconstruction errors for all types of faults are calculated and presented in Table 8. Average testing error for all types of fault was 0.9386%. The small reconstruction error indicates good performance of the models.

6. Conclusions

We propose an end-to-end deep learning framework to detect the presence of a fault in sensor data, its type, and reconstruct the corrected data for faulty sensors. A convolutional neural network (CNN) was employed for the classification task, and reconstruction of corrected data was performed using a suite of convolutional autoencoder (CAE) models. Our framework is particularly useful for applications relying on real-time sensor data. For instance, continuous response data is necessary to monitor the structural health

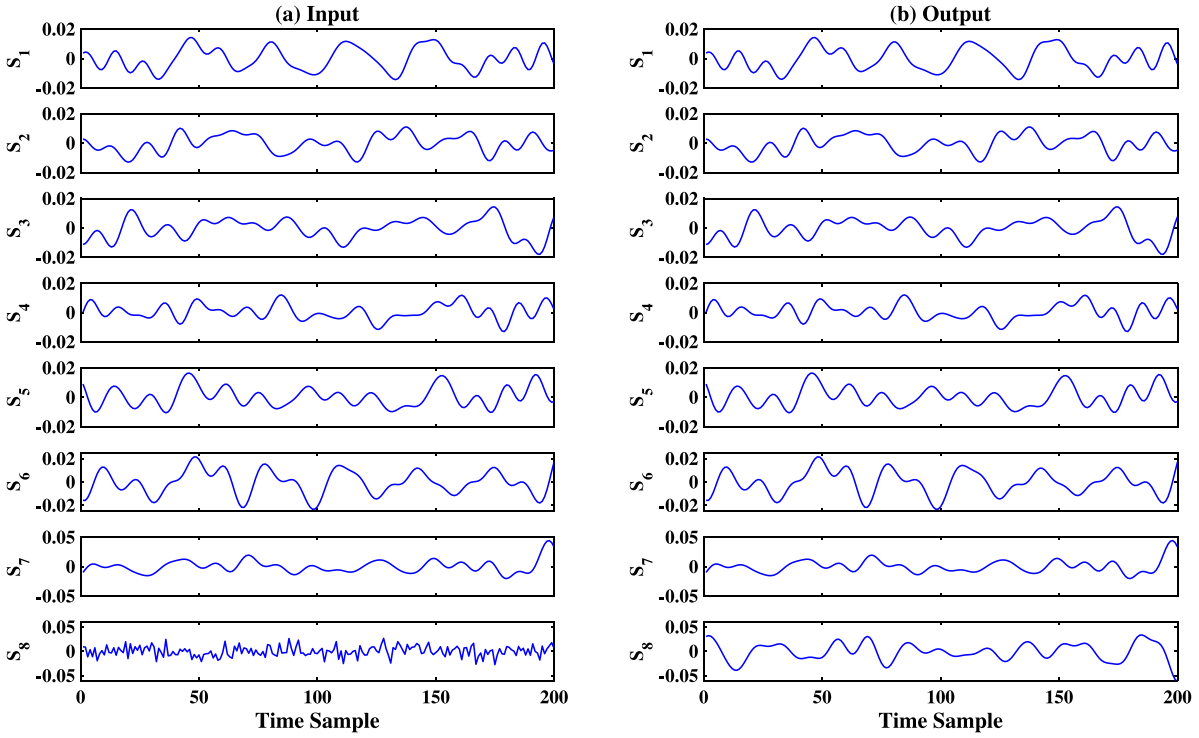

Fig. 21. Sample training data (acceleration responses in m/s^2) for CAE-RC_2 – (a) Input (*Random* Fault in Sensor S_8); (b) Output (ground truth data — with actual time history available for Sensor S_8).

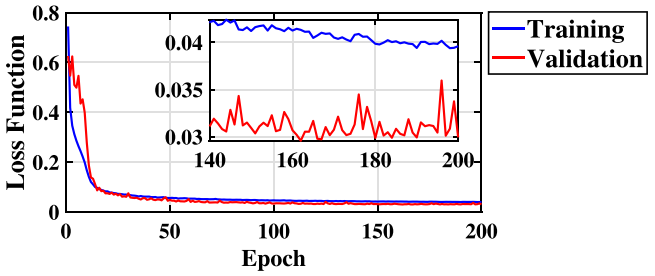

Fig. 22. Loss function of the CAE-RC_2, Mean Squared Error (MSE) Loss has converged.

Table 8
$||RE||_2$ of the framework for different fault types.

Fault type	Training error (%)	Testing error (%)
Missing	1.1810	1.4001
Random	0.9425	1.4069
Drift	0.1973	0.2341
Spiky	0.5605	0.7133

in near-real-time. The automatic detection of sensor faults and correction of faulty sensor data is critical for reliable, continuous operation of structural and infrastructural systems.

We validate the framework using data from numerical simulations of a 10 DOF shear-type system and data from physical experiments on a simplified model of an arch bridge. For the test datasets, our trained models yield a classification accuracy of 98.70% for the numerical simulations benchmark and 98.74% for the physical experiments benchmark. Amongst the different types of faults, *Spiky* fault and *Drift* fault were relatively difficult to classify as the data for both types of faults contained some samples with a relatively small magnitude of the fault. This resulted in these samples getting misclassified as *Normal* data. On the other hand, *Random* and *Missing* faults have clear, distinctive characteristics, which makes it relatively easier to characterize them and resulting in close to 100% classification accuracy.

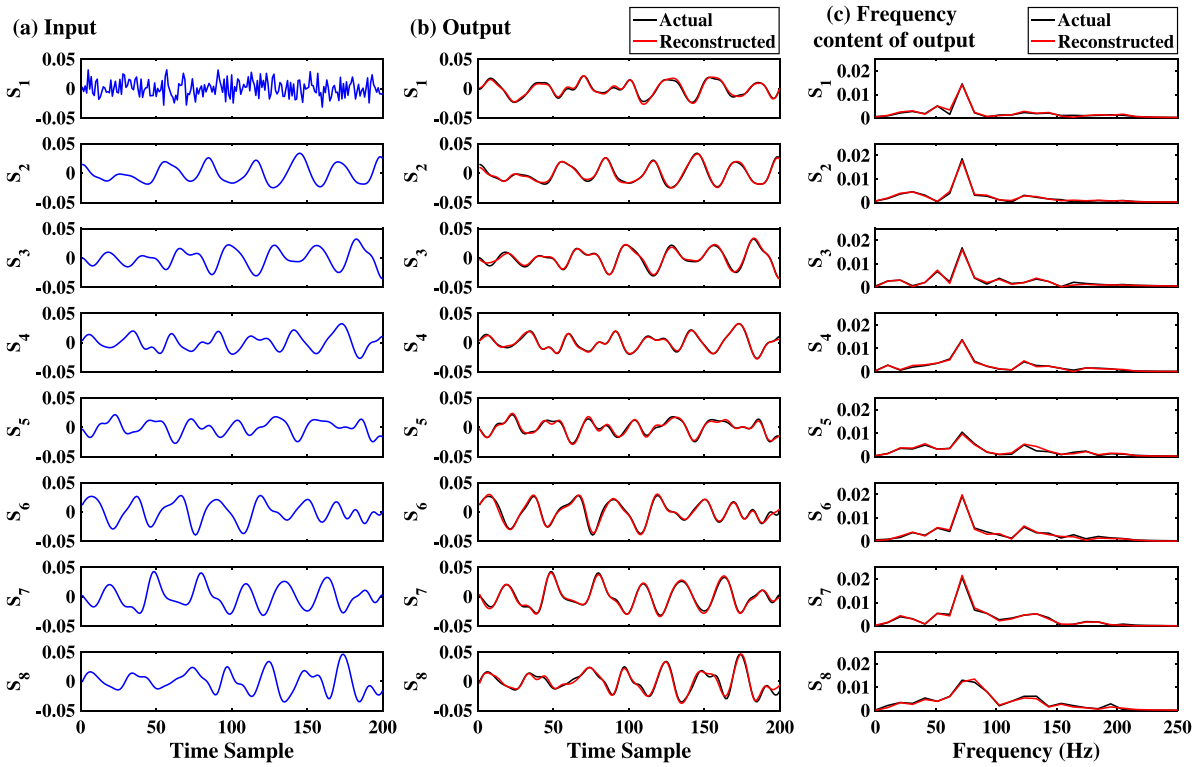

Fig. 23. Sample testing data (acceleration responses in m/s^2) for CAE-RC$_2$ – (a) Input (*Random* Fault in Sensor S_1); (b) Output (reconstructed time history for Sensor S_1) - as the reconstructed time history of the healthy sensors is unwarranted, it will be replaced with the original, unprocessed sensor data in the post-processing step; (c) Frequency content (m/s) of the actual and reconstructed data in Fig. 23(b).

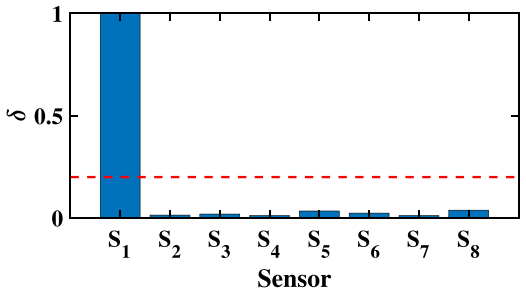

Fig. 24. Faulty sensor localization.

The accuracy of fault localization was 100% for both benchmarks, while the average reconstruction error (2-norm) over all types of single sensor faults was 0.2305% and 0.9386%, for the simulations benchmark and physical experiments benchmarks, respectively. The reconstruction error was the minimum for *Drift* fault as the goal of the CAE reconstructor is relatively simple, i.e., determining and eliminating the bias from the sensor data. On the other hand, for other faults like *Missing* and *Random*, the CAE reconstructs the corrected sensor data using information only from the other sensors and the spatiotemporal correlations. When these faults occur, the data corresponding to the faulty sensor does not contain any meaningful information about the original signal.

We provide a fast and accurate way to automatically detect faults in sensor data, identify the faulty sensor, and subsequently correct the fault. The performance of the presented framework is better than the other state-of-the-art methods in terms of accuracy and computational efficiency. Our method needs fewer models to train than other methods, which makes the proposed framework also less data-intensive. As demonstrated by the results from both numerical simulations and physical experiments, when adequately trained, the model architectures are robust against noise in sensor data, amenable to address faults in multiple sensors and for a range of faulty sensor combinations, and scalable with the size of the sensor network. The end-to-end framework is automatic, agnostic to types of sensors that capture vibration response, independent on the location and time history of the forcing function [70,71], and can significantly improve downstream applications that use the sensor data for system monitoring and maintenance decisions under uncertainty.

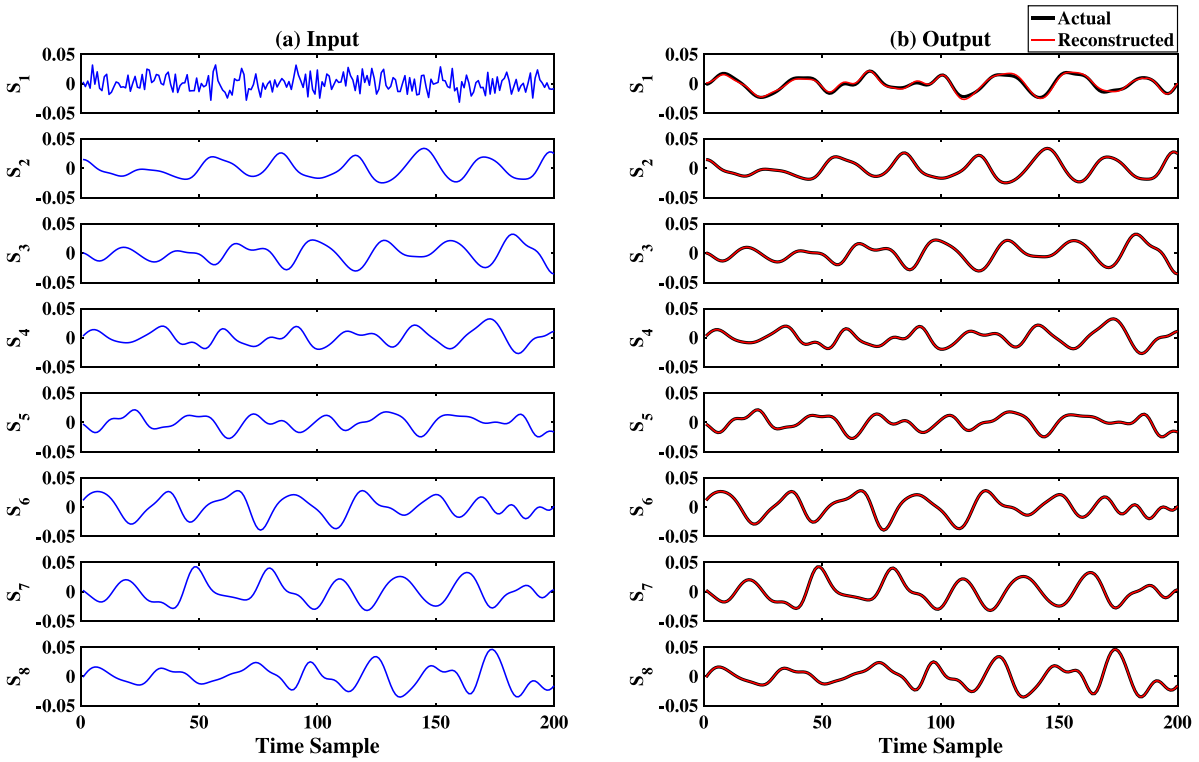

Fig. 25. (a) Sample input to the framework (m/s^2); (b) Final output of the framework (m/s^2) – it detects the presence of fault, the type of fault as *random*, the location of fault at S_1, and reconstructs the time history.

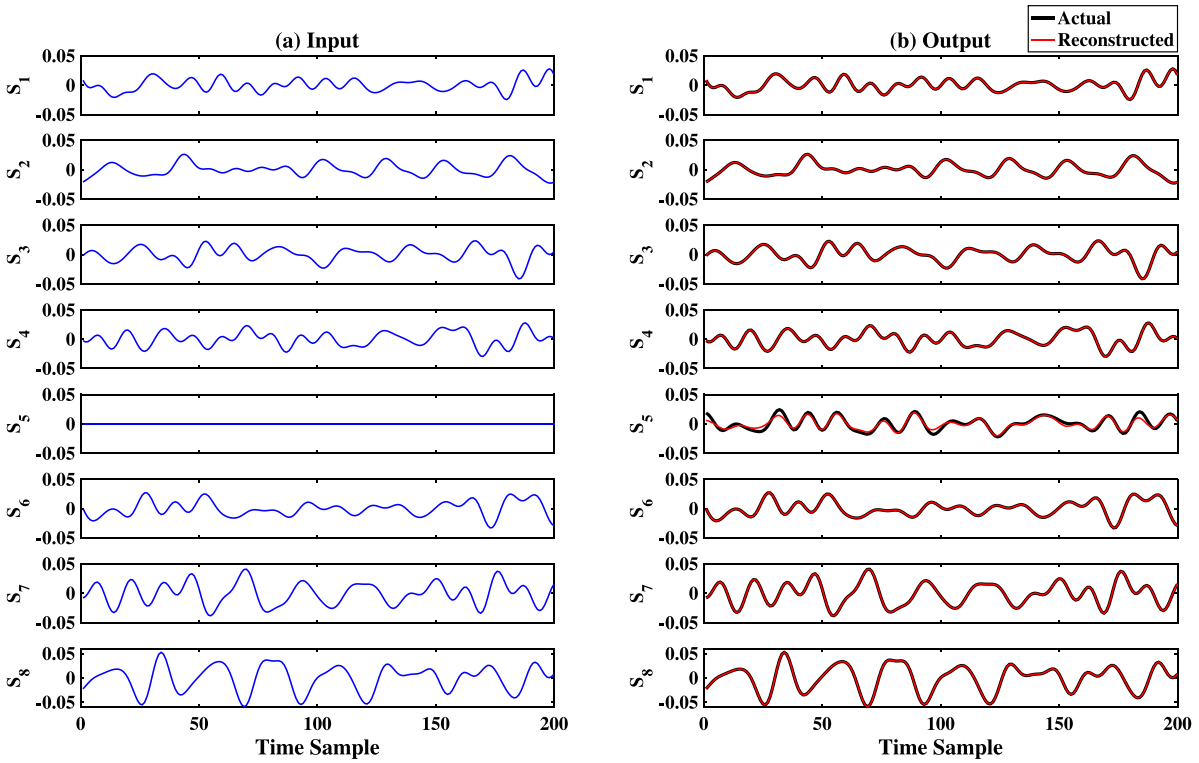

Fig. 26. (a) Sample input to the framework (m/s^2); (b) Final output of the framework (m/s^2) – it detects the presence of fault, the type of fault as *missing*, the location of fault at S_5, and reconstructs the time history.

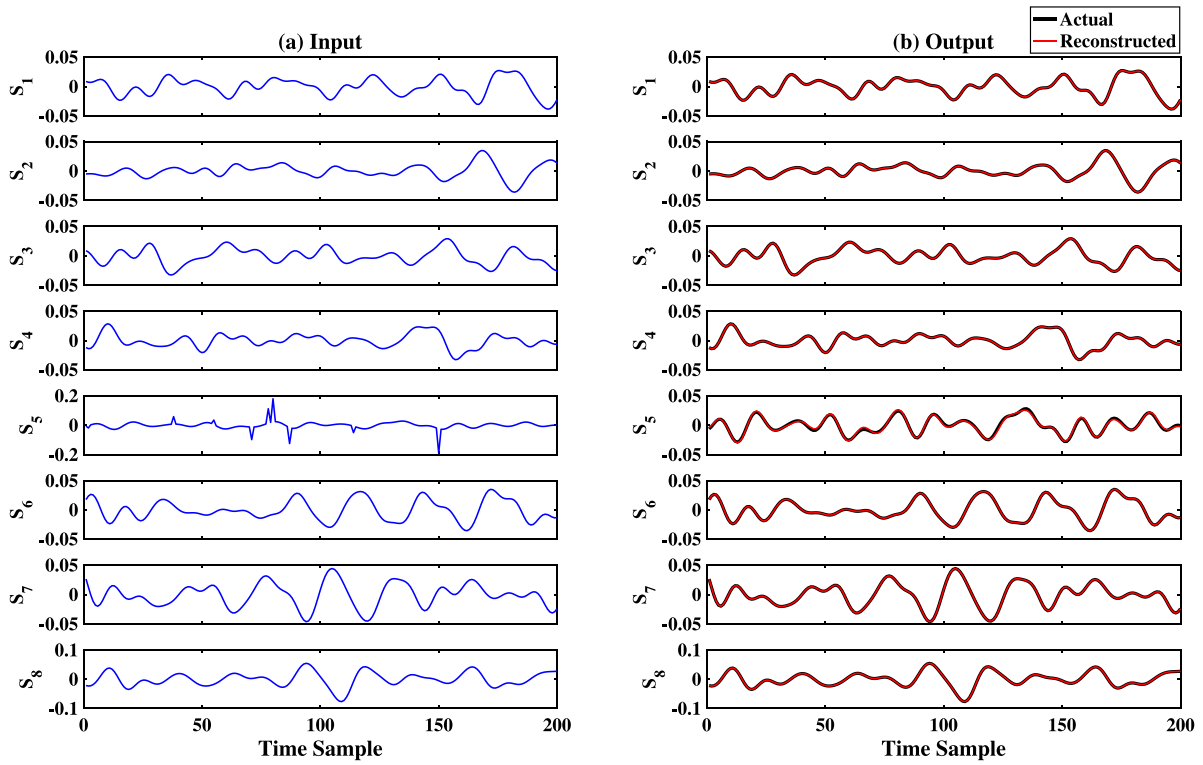

Fig. 27. (a) Sample input to the framework (m/s^2); (b) Final output of the framework (m/s^2) – it detects the presence of fault, the type of fault as $drift$, the location of fault at S_6, and reconstructs the time history.

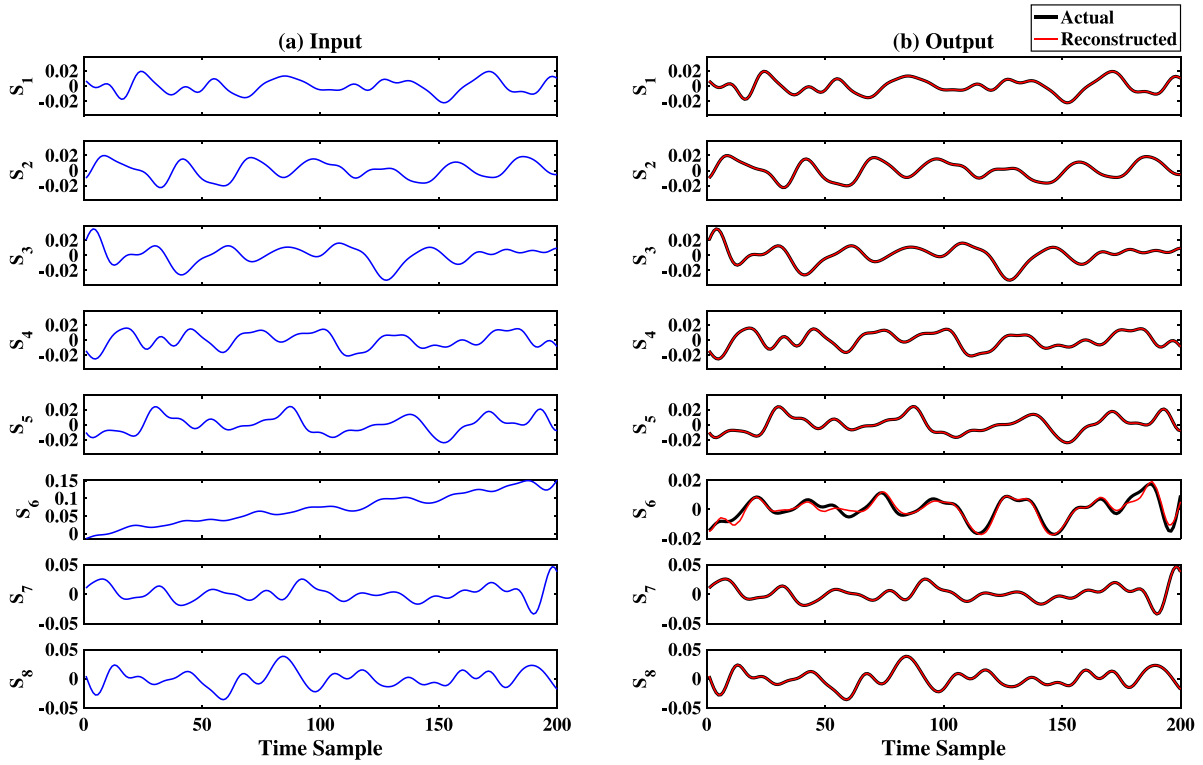

Fig. 28. (a) Sample input to the framework (m/s^2); (b) Final output of the framework (m/s^2) – it detects the presence of fault, the type of fault as $spiky$, the location of fault at S_5, and reconstructs the time history.

Table A.9
Hardware and software information for the deep learning framework.

CPU model and speed	Two CPU; Intel(R) Xeon(R) CPU @ 2.00 GHz
Available Hard disk	32 GB
Available RAM	13 GB
GPU type and spec	Tesla T4 — 16 GB
Programming	Python 3.6, Matlab 2021a
Deep learning framework	Tensorflow 1.14, Numpy, Keras 2.3.1, Talos

In practice, collecting faulty sensor data and manually annotating it to train deep neural network models is difficult. To address this lack of data, we propose to train on the numerical data and test on experimental data if the fault-features have a similar distribution. Transfer learning [72] can also be a good alternative, especially if the fault-features of the experimental data are significantly different from numerical data which was used in the training. Transfer learning can also be incorporated to train a bigger classifier and more reconstructors which address additional fault-types beyond those addressed in this study.

In this study, we assume that the system of interest is a linear time-invariant (LTI) system. As part of future work, this framework can be extended to nonlinear [73,74] and time-variant systems as well. As our framework works well for LTI systems, we expect it to perform well on linear time-variant systems too as suggested in [10]. Further, more error types can be included in the framework which will make the framework more practical. To further increase the practicality, another potential approach that can be explored is the use of transfer learning to use a network trained on one SHM problem to the other. Further, the efficacy of the proposed method can also be tested on structural health monitoring data obtained from a real system such as a bridge or a building. Uncertainty propagation and quantification of the proposed framework is also a scope of future study.

CRediT authorship contribution statement

Debasish Jana: Writing – original draft, Conceptualization, Methodology, Software, Formal analysis, Investigation, Data curation, Visualization. **Jayant Patil:** Writing – original draft, Conceptualization, Methodology, Investigation, Resources. **Sudheendra Herkal:** Writing – original draft, Conceptualization, Methodology, Software, Investigation, Data curation, Visualization. **Satish Nagarajaiah:** Conceptualization, Methodology, Funding acquisition, Writing – review & editing, Supervision. **Leonardo Duenas-Osorio:** Conceptualization, Funding acquisition, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We sincerely thank Ashesh Chattopadhyay, graduate student, Department of Mechanical Engineering, Rice University, and Rohan Mukherjee, graduate student, Department of Computer Science, Rice University for their valuable comments and suggestions. We also thank Daniel Neumann, Ryon Lab Technician at Rice University for assisting in the setup for physical experiments.

Funding

This research was made possible by financial support from Science and Engineering Research Board of India (SERB) and Rice University to Debasish Jana, Sudheendra Herkal, and Satish Nagarajaiah. The financial support by SERB-India is gratefully acknowledged. Jayant Patil and Leonardo Duenas-Osorio are grateful for the financial support by the U.S. Department of Defense (Grant No. W911NF-13-1-0340) and by the U.S. National Science Foundation (Grant No. CMMI-1541033).

Appendix A. Hardware specification and computation details

A.1. Hardware information

We have used Google Colab (free version) for training, validation, and testing of all our deep learning networks i.e., the CNN and the four reconstructors. In different runs google colab provides a little different hardware, but typically most of the time we were given the hardware with following specs — mentioned in Table A.9.

Table A.10
Computation cost of the framework.

Stage of the framework	Number of trainable parameters per epoch	Training time for all epoch (s)
Classifier (CNN)	1,001,745	815.92
Each Reconstructor (CAE)	1,216,421	2159.46
Total (CNN+4×CAE)	5,867,429	9453.76

Table A.11
Computation time/Latency of the framework.

Stage of the framework	Computation time per window
Pre-processing (windowing and normalization)	0.052 ms
Classifier (fault detection and classification)	43.27 ms
Reconstructor (data reconstruction)	44.27 ms
Post-processing (fault isolation)	0.021 ms
Total computation time per window	88 ms (approximately)

A.2. Computing cost

Conventionally, the computational cost of the deep learning based frameworks can be represented as the number of trainable parameters (parameters which get updated by the gradient descent optimizer) in each epoch [75]. In our proposed framework, the deep learning based stages are the classifier and the four reconstructors. The architecture of all the reconstructors are same. Total number of trainable parameter = Number of trainable parameter of the classifier + 4 × Number of trainable parameter of each reconstructor. The number of trainable parameters and the total training time is represented in Table A.10.

A.3. Computing time or inference speed (latency) of the framework

For each forward pass of the framework takes an approximate total of 88 ms. The breakdown for each stages of the framework is mentioned in Table A.11.

Appendix B. Selection of sampling frequency of data acquisition

In this section, we provide some guidelines for selecting the sampling frequency of data acquisition so that adequately trained CNN and CAE models will perform reasonably well on new problems. The primary recommendation for data collection while using this end-to-end framework is to select a sampling frequency that is significantly higher than the observable natural frequency of the structure. In the CAE reconstructors, convolution and max-pooling operations effectively downsample the data. If the sampling frequency is not higher, there is a chance of aliasing in the structural frequency content. In such cases, the signals with high-frequency content in the response will not be reconstructed effectively. The selected sampling frequency only affects the CAE reconstructor models and not the CNN classifier. We recommend that the sampling frequency f_s should be at-least $8f_{high}$, where f_{high} is the highest observable structural frequency due to the external forcing function, as two convolutional layers are present in the autoencoder bottleneck. In both the benchmark problems described in this paper, the sampling frequency was ten times the f_{high} i.e., $f_s = 10f_{high}$.

Appendix C. Selection of time-window size of input–output

The size of the time-window is an important parameter in this framework. This window size is chosen such that all the signal frequency is contained in a single window. As an example, a very low-frequency signal may not be preserved in a small window. However, a larger window will increase the total number of model parameters of all the models in the entire framework. This trade-off between the sampling frequency of data acquisition, structural frequency content, and the size of the time window needs to be balanced to achieve adequate accuracy with a reasonable computational cost. In this study, the time-window size was selected to contain 200-time steps. For the numerical simulation benchmark, this represented 1 s of sensor data, as the fundamental frequency of the shear-type building was 2.37 Hz. Similarly, for the experimental benchmark, the time-window size of 200-time steps represented 0.1 s of sensor data as the fundamental frequency of the physical model was relatively large at 71.72 Hz.

In this paper, we considered the window length which has time sample of 200. The data acquisition sampling frequency of numerical simulation is 200 Hz and for the experimental validation it was 2000 Hz — this means that the proposed approach can process on 1 s of data for the example shown in numerical simulation and 0.1 s of data for the example considered in experimental validation. If the user seeks to detect temporary faults with a much shorter time window, then the data acquisition frequency has to be increased.

Table D.12
Tuned hyper-parameters for numerical simulation.

Model	Batch size	Dropout	Learning rate
CNN classifier	500	0.4	0.0001
Missing CAE	50	0.4	0.0001
Random CAE	100	0.5	0.001
Drift CAE	50	0.5	0.001
Spiky CAE	100	0.5	0.0001

Table D.13
Kernel size optimization for CNN.

Kernel size		Train	Validation
1st layer	2nd layer	Loss	Loss
11 × 1	5 × 1	0.0113	0.0722
11 × 1	7 × 1	0.0120	0.0748
15 × 1	3 × 1	0.0184	0.0766
15 × 1	5 × 1	0.0157	0.0775
15 × 1	7 × 1	0.0144	0.0798
7 × 1	3 × 1	0.0145	0.0835
7 × 1	5 × 1	0.0120	0.0783

Table D.14
Kernel size optimization for CAEs.

Kernel size		Train loss	Validation loss
1st layer	2nd layer		
31 × 5	31 × 3	0.0751	0.0516
11 × 5	11 × 3	0.0765	0.0633
21 × 5	21 × 3	0.075	0.0632
41 × 5	41 × 3	0.0781	0.0597
31 × 3	31 × 3	0.0758	0.0609

Appendix D. Hyper-parameter tuning in deep learning framework

D.1. Hyper-parameter optimization for the numerical simulation

The hyperparameters for the CNN and the CAE models were optimized using grid search implemented using the Talos Python package. The Talos package works with the Keras, and TensorFlow framework [59]. The tuned hyperparameters of the CNN classifier and the 4 CAE reconstructors are shown in Table D.12.

D.2. Classifier kernel size ablation study for the numerical simulation

We optimized the size of the CNN kernel using trial and error. Kernel size for the 1st and 2nd layer is chosen as 11 × 1 and 5 × 1 respectively corresponding to the lowest validation loss. From Table D.13, we also observe that change in both training and validation losses are very small for different kernel sizes.

D.3. Reconstructor kernel size optimization for the numerical simulation

Kernel sizes for the CAEs are optimized using trial and error. Kernel size for 1st and 2nd layer is chosen as 31 × 5 and 31 × 3 respectively. From Table D.14, we can also infer that the change in training or validation loss is very small for other kernel sizes.

D.4. Hyper-parameter optimization for experimental validation

The hyperparameters of the CNN classifier and the four CAE reconstructor models for this experimental dataset were optimized using the Talos package [59]. The optimized hyperparameters for training the models are tabulated in Table D.15.

Table D.15
Tuned hyper-parameters for experimental data.

Model	Batch size	Dropout	Learning rate
CNN classifier	500	0.7	0.0001
Missing CAE	500	0.4	0.001
Random CAE	100	0.4	0.001
Drift CAE	50	0.4	0.0001
Spiky CAE	50	0.4	0.0001

Table E.16
Confusion matrix of the CNN classifier (in %).

		Predicted class				
		Normal	Missing	Random	Drift	Spiky
Actual class	Normal	100	0	0	0	0
	Missing	1.3	98.6	0	0.05	0.05
	Random	0	0	99.95	0	0.05
	Drift	4.45	0	0	95.55	0
	Spiky	3.15	0.15	0.05	0	96.55

Table E.17
$||RE||_2$ for each type of sensor data fault.

Fault type	Training error (%)	Testing error (%)
Missing	0.0136	0.0277
Random	0.0158	0.0227
Drift	0.0132	0.0408
Spiky	0.0164	0.0271

Appendix E. Performance of the framework on displacement data

Accelerometers are the most common type of sensors used in structural health monitoring application [76]. Displacement measurement devices like Linear Variable Differential Transformer (LVDT) and lasers need a reference point to measure the displacement responses. Our proposed method is applicable to any type of vibration response i.e. displacement, velocity, acceleration, and strain response. In this study we prove that our framework can be used successfully to reconstruct the time history and the frequency content of accelerometer data.

As the proposed framework works successfully with acceleration response, it would very likely work for the displacement and velocity responses. This is because the deep learning models are agnostic of the dimension and scale of the input signals due to input normalization. The displacement and velocity responses are the time integrals of acceleration responses while strain is the spatial derivative of the displacement response data. Based on the reviewers suggestion, we have added the performance of our framework for displacement response data in the revised manuscript. The confusion matrix for the classifier model is reported in Table E.16 while the reconstruction error for the end-to-end framework is reported in Table E.17.

We intend to further examine the models in our framework numerically and experimentally on strain data using strain gauges, 'strain paint', and 'strain sensing smart skin' [77–81] as part of future work.

References

[1] S. Li, M. Pozzi, What makes long-term monitoring convenient? A parametric analysis of value of information in infrastructure maintenance, Struct. Control Health Monit. 26 (5) (2019) e2329.
[2] K. Ni, N. Ramanathan, M.N.H. Chehade, L. Balzano, S. Nair, S. Zahedi, E. Kohler, G. Pottie, M. Hansen, M. Srivastava, Sensor network data fault types, ACM Trans. Sensor Netw. 5 (3) (2009) 1–29.
[3] B. Spencer Jr., S. Nagarajaiah, State of the art of structural control, J. Struct. Eng. 129 (7) (2003) 845–856.
[4] S. Nagarajaiah, Y. Yang, Modeling and harnessing sparse and low-rank data structure: a new paradigm for structural dynamics, identification, damage detection, and health monitoring, Struct. Control Health Monit. 24 (1) (2017) e1851.
[5] S. Jeong, M. Ferguson, K.H. Law, Sensor data reconstruction and anomaly detection using bidirectional recurrent neural network, in: Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems 2019, Vol. 10970, International Society for Optics and Photonics, 2019, 109700N.
[6] J. Quevedo, R. Pérez, T. Escobet, Real-Time Monitoring and Operational Control of Drinking-Water Systems, Springer, 2017.
[7] Y. Yang, S. Nagarajaiah, Blind denoising of structural vibration responses with outliers via principal component pursuit, Struct. Control Health Monit. 21 (6) (2014) 962–978.
[8] Z. Li, B. Koh, S. Nagarajaiah, Detecting sensor failure via decoupled error function and inverse input–output model, J. Eng. Mech. 133 (11) (2007) 1222–1228.
[9] B. Koh, Z. Li, P. Dharap, S. Nagarajaiah, M. Phan, Actuator failure detection through interaction matrix formulation, J. Guid. Control Dyn. 28 (5) (2005) 895–901.
[10] P. Dharap, B.-H. Koh, S. Nagarajaiah, Structural health monitoring using armarkov observers, J. Intell. Mater. Syst. Struct. 17 (6) (2006) 469–481, http://dx.doi.org/10.1177/1045389X06058793.

[11] G. Olsson, M. Nielsen, Z. Yuan, A. Lynggaard-Jensen, J.-P. Steyer, Instrumentation, Control and Automation in Wastewater Systems, IWA publishing, 2005.
[12] M. Mourad, J.-L. Bertrand-Krajewski, A method for automatic validation of long time series of data in urban hydrology, Water Sci. Technol. 45 (4–5) (2002) 263–270.
[13] K. Ustoorikar, M. Deo, Filling up gaps in wave data with genetic programming, Mar. Struct. 21 (2–3) (2008) 177–195.
[14] C. Yoo, K. Villez, I. Lee, S. Van Hulle, P.A. Vanrolleghem, Sensor validation and reconciliation for a partial nitrification process, Water Sci. Technol. 53 (4–5) (2006) 513–521.
[15] K. Smarsly, K.H. Law, Decentralized fault detection and isolation in wireless structural health monitoring systems using analytical redundancy, Adv. Eng. Softw. 73 (2014) 1–10.
[16] K. Rokneddin, J. Ghosh, L. Dueñas Osorio, J.E. Padgett, Seismic reliability assessment of aging highway bridge networks with field instrumentation data and correlated failures, II: Application, Earthq. Spectra 30 (2) (2014) 819–843.
[17] G. Kerschen, P. De Boe, J.-C. Golinval, K. Worden, Sensor validation using principal component analysis, Smart Mater. Struct. 14 (1) (2004) 36.
[18] J. Kullaa, Sensor validation using minimum mean square error estimation, Mech. Syst. Signal Process. 24 (5) (2010) 1444–1457.
[19] K.H. Law, S. Jeong, M. Ferguson, A data-driven approach for sensor data reconstruction for bridge monitoring, in: 2017 World Congress on Advances in Structural Engineering and Mechanics, 2017.
[20] Y. Cheng, Q. Liu, J. Wang, S. Wan, T. Umer, Distributed fault detection for wireless sensor networks based on support vector regression, Wirel. Commun. Mob. Comput. 2018 (2018) 4349795:1–4349795:8.
[21] D.L. Donoho, et al., Compressed sensing, IEEE Trans. Inform. Theory 52 (4) (2006) 1289–1306.
[22] R.G. Baraniuk, Compressive sensing, IEEE Signal Process. Mag. 24 (4) (2007).
[23] Y. Bao, H. Li, X. Sun, Y. Yu, J. Ou, Compressive sampling–based data loss recovery for wireless sensor networks used in civil structural health monitoring, Struct. Health Monit. 12 (1) (2013) 78–95.
[24] D. Jana, S. Nagarajaiah, Y. Yang, S. Li, Real-time cable tension estimation from acceleration measurements using wireless sensors with packet data losses: analytics with compressive sensing and sparse component analysis, J. Civ. Struct. Health Monit. (2021) 1–19.
[25] K. Dragos, K. Smarsly, Distributed adaptive diagnosis of sensor faults using structural response data, Smart Mater. Struct. 25 (10) (2016) 105019.
[26] S. Jeong, M. Ferguson, R. Hou, J.P. Lynch, H. Sohn, K.H. Law, Sensor data reconstruction using bidirectional recurrent neural network with application to bridge monitoring, Adv. Eng. Inform. 42 (2019).
[27] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Nature 323 (6088) (1986) 533–536.
[28] F.P. Xizhao Wang, Recent advances in deep learning, Int. J. Mach. Learn. Cybern. 11 (2020) 747–750.
[29] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[30] M. Schuster, K.K. Paliwal, Bidirectional recurrent neural networks, IEEE Trans. Signal Process. 45 (11) (1997) 2673–2681.
[31] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.
[32] D.C. Ciresan, U. Meier, J. Masci, L.M. Gambardella, J. Schmidhuber, Flexible, high performance convolutional neural networks for image classification, in: Twenty-Second International Joint Conference on Artificial Intelligence, 2011.
[33] B. Zhao, H. Lu, S. Chen, J. Liu, D. Wu, Convolutional neural networks for time series classification, J. Syst. Eng. Electron. 28 (1) (2017) 162–169.
[34] T. Young, D. Hazarika, S. Poria, E. Cambria, Recent trends in deep learning based natural language processing, Ieee Comput. Intell. Mag. 13 (3) (2018) 55–75.
[35] K. Wang, K. Li, L. Zhou, Y. Hu, Z. Cheng, J. Liu, C. Chen, Multiple convolutional neural networks for multivariate time series prediction, Neurocomputing 360 (2019) 107–119.
[36] J. Cao, J. Wang, Stock price forecasting model based on modified convolution neural network and financial time series analysis, Int. J. Commun. Syst. 32 (12) (2019) e3987.
[37] Y. LeCun, G. Hinton, Deep learning, Nature 521 (2015) 436–444.
[38] I. Goodfellow, Y. Bengio, A. Courville, Deep learning, 2016.
[39] K. Jarrett, K. Kavukcuoglu, M. Ranzato, Y. LeCun, What is the best multi-stage architecture for object recognition? in: 2009 IEEE 12th International Conference on Computer Vision, IEEE, 2009, pp. 2146–2153.
[40] M. Ranzato, F.J. Huang, Y. Boureau, Y. LeCun, Unsupervised learning of invariant feature hierarchies with applications to object recognition, in: 2007 IEEE Conference on Computer Vision and Pattern Recognition, 2007, pp. 1–8.
[41] V. Turchenko, A. Luczak, Creation of a deep convolutional auto-encoder in caffe, in: 2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Vol. 2, 2017, pp. 651–659.
[42] Z. Tang, Z. Chen, Y. Bao, H. Li, Convolutional neural network-based data anomaly detection method using multiple information for structural health monitoring, Struct. Control Health Monit. 26 (1) (2019) e2296.
[43] S. Nagarajaiah, K. Erazo, Structural monitoring and identification of civil infrastructure in the United States, Struct. Monit. Maint. 3 (1) (2016) 51.
[44] D. Li, Y. Wang, J. Wang, C. Wang, Y. Duan, Recent advances in sensor fault diagnosis: A review, Sensors Actuators A 309 (2020) 111990.
[45] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015, arXiv preprint arXiv:1502.03167.
[46] Y.A. LeCun, L. Bottou, G.B. Orr, K.-R. Müller, Efficient backprop, in: Neural Networks: Tricks of the Trade, Springer, 2012, pp. 9–48.
[47] R. Gomez, Understanding categorical cross-entropy loss, binary cross-entropy loss, softmax loss, logistic loss, focal loss and all those confusing names, 2018, URL: https://gombru.github.io/2018/05/23/crossentropyloss/ (Visited on 29/03/2019).
[48] X.-J. Mao, C. Shen, Y.-B. Yang, Image restoration using convolutional auto-encoders with symmetric skip connections, 2016, arXiv preprint arXiv:1606.08921.
[49] J. Masci, U. Meier, D. Cireşan, J. Schmidhuber, Stacked convolutional auto-encoders for hierarchical feature extraction, in: International Conference on Artificial Neural Networks, Springer, 2011, pp. 52–59.
[50] J. Xie, L. Xu, E. Chen, Image denoising and inpainting with deep neural networks, in: Advances in Neural Information Processing Systems, 2012, pp. 341–349.
[51] C. Sammut, G.I. Webb (Eds.), Mean squared error, in: Encyclopedia of Machine Learning, Springer US, Boston, MA, 2010, p. 653, http://dx.doi.org/10.1007/978-0-387-30164-8_528.
[52] J.M. Johnson, T.M. Khoshgoftaar, Survey on deep learning with class imbalance, J. Big Data 6 (1) (2019) 1–54.
[53] C. Shorten, T.M. Khoshgoftaar, A survey on image data augmentation for deep learning, J. Big Data 6 (1) (2019) 1–48.
[54] A. Debnath, G. Waghmare, H. Wadhwa, S. Asthana, A. Arora, Exploring generative data augmentation in multivariate time series forecasting: Opportunities and challenges, Solar-Energy 137 (2021) 52–560.
[55] S. Haykin, N. Network, A comprehensive foundation, Neural Netw. 2 (2004) (2004) 41.
[56] M.J. Kearns, A bound on the error of cross validation using the approximation and estimation rates, with consequences for the training-test split, in: Advances in Neural Information Processing Systems, 1996, pp. 183–189.
[57] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 2010, pp. 249–256.
[58] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.
[59] Autonomio talos [computer software], 2019, https://github.com/autonomio/talos (Accessed: 2020-08-01).

[60] R.M. Barrett, Troubleshooting Accelerometer Installations, Wilcoxon Research, Maryland, 2004.
[61] B.-H. Mevik, R. Wehrens, The pls package: Principal component and partial least squares regression in r, J. Statist. Softw. Artic. 18 (2) (2007) 1–23.
[62] S. De Jong, Simpls: an alternative approach to partial least squares regression, Chemometr. Intell. Lab. Syst. 18 (3) (1993) 251–263.
[63] H. Drucker, C.J. Burges, L. Kaufman, A.J. Smola, V. Vapnik, Support vector regression machines, in: Advances in Neural Information Processing Systems, 1997, pp. 155–161.
[64] A.J. Smola, B. Schölkopf, A tutorial on support vector regression, Stat. Comput. 14 (3) (2004) 199–222.
[65] C. Huang, S. Nagarajaiah, Experimental study on bridge structural health monitoring using blind source separation method: arch bridge, Struct. Monit. Maint. 1 (1) (2014) 69.
[66] Pasco bridge model, 2020, https://www.pasco.com/products/lab-apparatus/mechanics/structures-systems/me-6992 (Accessed: 2020-08-01).
[67] Mb dynamics (modal 50a) mass shaker, 2020, https://www.mbdynamics.com/products/modal-exciters-amplifiers/modal-50a/ (Accessed: 2020-08-01).
[68] Pcb accelerometer, 2020, https://www.pcb.com/products?model=333B32 (note = Accessed: 2020-08-01).
[69] National instrument data acquisition system, 2020, https://www.ni.com/en-us/support/downloads/software-products/download.signalexpress.html#322415 (Accessed: 2020-08-01).
[70] D. Jana, S. Mukhopadhyay, S. Ray-Chaudhuri, Fisher information-based optimal input locations for modal identification, J. Sound Vib. 459 (2019) 114833.
[71] D. Jana, D. Ghosh, S. Mukhopadhyay, S. Ray-Chaudhuri, Optimal input locations for stiffness parameter identification, in: Model Validation and Uncertainty Quantification, Vol. 3, Springer, 2020, pp. 399–403.
[72] L. Torrey, J. Shavlik, Transfer learning, in: Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques, IGI global, 2010, pp. 242–264.
[73] W. Chen, D. Jana, A. Singh, M. Jin, M. Cenedese, G. Kosova, M.R. Brake, C.W. Schwingshackl, S. Nagarajaiah, K.J. Moore, et al., Measurement and identification of the nonlinear dynamics of a jointed structure using full-field data, part I: Measurement of nonlinear dynamics, Mech. Syst. Signal Process. 166 (2022) 108401.
[74] M. Jin, G. Kosova, M. Cenedese, W. Chen, A. Singh, D. Jana, M.R. Brake, C.W. Schwingshackl, S. Nagarajaiah, K.J. Moore, et al., Measurement and identification of the nonlinear dynamics of a jointed structure using full-field data; part II-nonlinear system identification, Mech. Syst. Signal Process. 166 (2022) 108402.
[75] M. Kubat, An Introduction to Machine Learning, Springer, 2017.
[76] M. Abdulkarem, K. Samsudin, F.Z. Rokhani, M.F. A Rasid, Wireless sensor network for structural health monitoring: A contemporary review of technologies, challenges, and future direction, Struct. Health Monit. 19 (3) (2020) 693–735.
[77] P. Dharap, Z. Li, S. Nagarajaiah, E. Barrera, Nanotube film based on single-wall carbon nanotubes for strain sensing, Nanotechnology 15 (3) (2004) 379.
[78] K.J. Loh, T.-C. Hou, J.P. Lynch, N.A. Kotov, Carbon nanotube sensing skins for spatial strain and impact damage identification, J. Nondestruct. Eval. 28 (1) (2009) 9–25.
[79] P.A. Withey, V.S.M. Vemuru, S.M. Bachilo, S. Nagarajaiah, R.B. Weisman, Strain paint: Noncontact strain measurement using single-walled carbon nanotube composite coatings, Nano Lett. 12 (7) (2012) 3497–3500.
[80] P. Sun, S.M. Bachilo, R.B. Weisman, S. Nagarajaiah, Carbon nanotubes as non-contact optical strain sensors in smart skins, J. Strain Anal. Eng. Des. 50 (7) (2015) 505–512.
[81] P. Sun, S.M. Bachilo, S. Nagarajaiah, R.B. Weisman, Toward practical non-contact optical strain sensing using single-walled carbon nanotubes, ECS J. Solid State Sci. Technol. 5 (8) (2016) M3012.