Contents lists available at ScienceDirect

Applied Soft Computing Journal

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

Capsule Neural Networks for structural damage localization and quantification using transmissibility data

Joaquín Figueroa Barraza [a], Enrique Lopez Droguett [b,*], Viviana Meruane Naranjo [b], Marcelo Ramos Martins [a]

[a] *Naval Engineering Department, University of Sao Paulo, Brazil*
[b] *Mechanical Engineering Department, University of Chile, Santiago, Chile*

A R T I C L E I N F O

Article history:
Received 14 November 2019
Received in revised form 22 July 2020
Accepted 12 September 2020
Available online 1 October 2020

Keywords:
Vibration-based damage assessment
Structural health monitoring
Transmissibility functions
Capsule networks
Convolutional Neural Networks

A B S T R A C T

One of the current challenges in structural health monitoring (SHM) is to take the most advantage of large amounts of data to deliver accurate damage measurements and predictions. Deep Learning methods tackle these problems by finding complex relations hidden in the data available. Amongst these, Capsule Neural Networks (CapsNets) have recently been developed, achieving promising results in benchmark Deep Learning problems. In this paper, Capsule Networks are expanded to locate and to quantify structural damage. The proposed approach is evaluated in two case studies: a system with springs and masses that simulate a structure, and a beam with different damage scenarios. For both case studies, training and validation sets are created using Finite Element (FE) models and calibrated with experimental data, which is also used for testing. The main contributions of this study are: A novel CapsNets-based method for dual classification–regression task in SHM, analysis of both routing algorithms (dynamic routing and Expectation–Maximization routing) in the context of SHM, and analysis of generalization between FE models and real-life experiments. The results show that the proposed Capsule Networks with dynamic routing achieve better results than Convolutional Neural Networks (CNN), especially when it comes to false positive values.

1. Introduction

Early detection of damage in structures is vital for preventing accidents. Buildings, bridges, dams and other large-scale structures have many sources of external loads throughout their lifespan that cannot be controlled or measured. These loads result in structural damage which must be detected and tracked down as soon as possible so that preventive actions are taken. According to [1], structural health monitoring (SHM) is "the process of implementing a damage identification strategy for aerospace, civil and mechanical engineering infrastructure". It is the monitoring of one or more variables in a structure and the evaluation of its operational conditions. Among different methods of SHM [2], vibration-based methods stand out because of their ability to recognize global properties of the structure and the non-destructive evaluation methods. The authors in [3] review a large number of cases in which different vibration-based methods are used for structural damage identification.

A main challenge in vibration-based damage assessment is how to take the most out of data to estimate damage accurately. Using the raw measurements of a series of accelerometers is one straightforward way, but there is information about the structure that cannot be identified clearly through this method. Instead, transmissibility functions (TFs) have been widely used for damage assessment [4–9]. TFs relate the responses between two points of the structure, and, of all dynamic responses, they are the easiest to obtain in real-time because of the straightforward in-situ measurement. No modal extraction is needed; therefore, data contamination with modal extraction errors is avoided. In [4,7–9], the use of transmissibility for damage detection is introduced and validated. The first work uses TFs to detect damage in a simple simulated lumped-parameter mechanical system. The rest of the aforementioned works validate the approach through experimental procedures on a simplified model of a metallic aircraft wingbox (i.e., a plate incorporating stiffening elements) and a Gnat aircraft wing.

So far, the most common way to use vibration data to perform structural damage analysis is through inverse modeling approaches. This can be done by using a Finite Element (FE) model of a structure, which is updated using optimization algorithms to minimize the difference between the numerical and experimental data [10–12]. For example, the authors in [13] detect cracks in a beam using a genetic algorithm, first defining a new beam element with a number of embedded transverse edge cracks for

* Correspondence to: Beauchef 851, West Bldg, Office 411, Santiago, Chile.
E-mail address: elopezdroguett@ing.uchile.cl (E.L. Droguett).

https://doi.org/10.1016/j.asoc.2020.106732
1568-4946/© 2020 Elsevier B.V. All rights reserved.

Abbreviations

CapsNets	Capsule neural networks
CNN	Convolutional neural networks
Conv	Convolution
DL	Deep learning
DME	Damage missing error
DNN	Deep neural network
DOF	Degree of freedom
DR	Dynamic routing
DS	Dataset
EM	Expectation–Maximization
EMR	EM routing
FAE	False alarm error
FE	Finite element
GAN	Generative adversarial networks
HL	Hidden layer
LANL	Los Alamos National Laboratory
MLP	Multilayer perceptron
MSE	Mean sizing error
NN	Neural network
OS-ELM	Online sequential extreme learning machine
ReLU	Rectified linear unit
RUL	Remaining useful life
SHM	Structural health monitoring
SM	Spring–Mass
TF	Transmissibility function

computing natural frequencies, and then solving an optimization problem to pursue the solution. In [14], Meruane et al. propose a methodology for antiresonance frequencies from transmissibility functions. These frequencies are used in a model-updating algorithm for structures. The algorithm correlates antiresonance frequencies in the objective function.

As an alternative to model-based algorithms in vibration analysis, machine learning algorithms have been proposed. Indeed, researchers have used neural networks along with feature extraction to learn from data and to perform different tasks. In [15], neural networks are used for damage detection in truss and frame structures. Meruane et al. [16] use an online sequential extreme learning machine (OS-ELM) to improve neural networks learning speed and to reduce the number of parameters. The neural network is used to identify and to quantify damage in two experimental cases: an 8 DOF spring–mass system and a beam under different damage scenarios. In, [5] Meruane et al. use a linear approximation method along with antiresonant frequencies identified from transmissibility functions, which leads to solving multiple nonlinear equations.

In all of the aforementioned works, feature extraction is key. Generally, this process is performed by experts, but there is no consensus to a “correct” feature extraction process. To tackle this issue, and with a general motivation towards automation, deep learning techniques have proven useful [17–26]. The authors in [17] propose the use of autoencoders and deep neural networks (DNN) for structural damage detection. First, the input vector suffers a non-linear dimensionality reduction using an autoencoder, and then goes through a regression process. The framework uses frequencies and mode shapes as input vectors, and outputs stiffness reduction values in each element of the structure analyzed, which is a seven-storey steel frame structure. The authors in [18] use neural networks to build an enhanced non-linear principal component analysis technique for structural damage identification. This network has the shape of an autoencoder, with mirrored layers being pre-trained separately. After this, fine tuning is performed over the whole network. This framework is evaluated on two bridges with different damage scenarios.

Among deep learning techniques, Convolutional Neural Networks (CNN) naturally stand out in reliability problems for their automatic feature extraction process. CNNs have been also applied in the field of structural health monitoring [21–27]. In [22], Modarres *et al.* use CNNs to identify and to classify structural damage. The framework is validated with synthetic data of a composite sandwich panel and with real concrete bridge crack images. In [24], damage detection is proposed by employing a multi-scale module composed of several different convolutional layers stacked together. This is done for obtaining as many features from the input signal as possible. These features are then passed on to a neural network with a large number of layers, including residual learning, convolutional, pooling and dropout layers. This approach is validated on two datasets based on numerical simulations, and two datasets based on laboratory measurements. In [27], CNNs are used to locate and to quantify structural damage using transmissibility images, achieving promising results. They take advantage of the CNNs capability for analyzing images to feed transmissibility images with minimal data preprocessing.

Although CNNs have been used for years, achieving state-of-the-art results in various fields, they have some limitations. CNNs are not capable of recognizing hierarchies within an image (or in any data structure for that matter), as all neurons output in a low-level layer are used equally in a high-level layer.

There has been some research to solve the existing drawbacks within CNNs. In [28], Hinton et al. realize that, while the machine learning community is using scalar output neurons, the computer vision community uses complex vectors as outputs and tries to address this with an auspicious idea of neurons giving an output vector containing instantiation parameters. The most recent approach has shown promising results. Indeed, in [29], Sabour et al. present Capsule Networks (CapsNets), a capsule system in which each capsule is a set of neurons arranged into a vector whose length represents its activation value. Capsules are organized into layers, just like a neural network, and the activation of a next-level capsule layer depends on a routing algorithm. In each routing iteration, capsules predict the output of the next layer capsules. The agreement between predictions is measured and determines the activation values. This model has surpassed state-of-the-art results in MNIST dataset of handwritten number images [30]. In [31], a different view on capsules is presented. Instead of vectors, capsules are represented by a pose matrix containing information about position and orientation for a particular feature, and an activation unit. This is inspired in computer graphics whereby pose matrices are used to define viewpoints with respect to an observer (a camera, for example) and to establish hierarchies between different parts of a whole. This architecture has shown particularly promising results in smallNORB, which is a dataset containing images of 3D objects, taken from different angles and with different lightning configurations.

Although CapsNets have been presented recently, they have been applied in various fields. In [32], Afshar et al. use Magnetic Resonance Imaging (MRI) images to classify types of brain tumors with CapsNets, achieving better results than those from CNNs. In [33], Upadhyay et al. use CapsNets in Generative Adversarial Networks (GANs) as a replacement to CNN discriminators, resulting in an improvement of the generator performance. In [34], LaLonde et al. adapt CapsNets to object segmentation. This is applied to pathological lung segmentation using Computed Tomography (CT) scans. In [35], the EM routing algorithm is modified by measuring agreement by the amount of alignment in a linear subspace, instead of agreement in clusters. Successful experiments are performed on MIMIC-III public dataset [36]. In [37], Ruiz-Tagle Palazuelos et al. use CapsNets with dynamic routing for the remaining useful life (RUL) estimation on NASA Commercial Modular Aero Propulsion System Simulation (C-MAPSS) dataset, expanding their use to regression tasks. It can be noticed that none of the aforementioned models and applications of Capsule Networks is related to structural damage assessment. Most of them are based on benchmark studies, revealing that CapsNets are still in an early stage of development. Also, with the exception of [37], only classification tasks are shown, which means possible CapsNets applications in regression tasks should be explored.

Since the AlexNet breakthrough in 2012 [38], CNNs have been analyzed extensively and achieved state-of-the-art results in various fields, including vibration-based methods for SHM. Nonetheless, there are many challenges still to be addressed, such as generalization capabilities, volume of training data, number of hyper-parameters, amongst others. Particularly in the field of SHM, one specific challenge is the detection of cracks as soon as possible, at the earliest stage, and hopefully prevent their occurrence. This has not yet been achieved and is one of the reasons that motivate researchers to analyze other kinds of algorithms. In turn, Capsule Networks are a result of a continuation work by the same research group that presented CNNs for solving some of its drawbacks. Its promising results encourage the idea of researching Capsule Networks applications in SHM.

In this paper, a new Capsule Network is proposed to locate and to quantify structural damage, which involves a dual classification and regression task. Two different routing algorithms (dynamic routing and EM routing) are explored and assessed in terms of their applicability in the structural damage assessment context by means of two case studies: a spring–mass system and a beam under different damage scenarios. Each of them is studied in different scenarios according to the number of damaged elements. The main contributions of this research are:

- A novel CapsNets-based method for dual classification–regression task in SHM.
- Analysis of both routing algorithms (DR and EM routing).
- Analysis of generalization between FE models and real-life experiments.

The remainder of this paper is structured as follows: Section 2 provides the background on transmissibility functions. In Section 3, Capsule Networks are explained in detail. Section 4 shows the case studies to be analyzed and descriptions of the datasets. Section 5 presents and describes the proposed capsule neural networks for damage localization and quantification. Section 6 presents the results obtained for both case studies and comparison with other models. Section 7 provides some concluding remarks.

2. Background

This section presents a general background to transmissibility functions and structural damage.

2.1. Transmissibility functions

In vibrations analysis, *transmissibility* is a concept for measuring the response to a certain stimulus at a specific location in a structural element. It numerically describes how vibrations propagate through an element.

There are two ratios that apply the concept of transmissibility: *force transmissibility* and *displacement transmissibility*. Force transmissibility is the ratio between the response and the stimulus in terms of force; displacement transmissibility is equivalent to the first one, but measuring displacement amplitude. Typically, both are presented in the frequency domain.

Transmissibility can be measured in complex systems using accelerometers. The concept of transmissibility is not only useful when measuring the ratio between response and stimulus, but also between two different responses to the same stimulus, that is, responses in different positions. Accelerometers can be installed at various locations of a structure, thus capturing many transmissibility functions. Experimental transmissibility functions are calculated by:

$$T_{ir}^{k}(\omega) = \frac{X_{ik}(\omega)}{X_{rk}(\omega)} \tag{1}$$

where $T_{ir}^{k}(\omega)$ is the transmissibility function between points i and r subject to an excitation at k, and $X_{rk}(\omega)$ is the response of point r due to an excitation force at k. As observed, only the location of the exciting force is needed, not its magnitude, which is an advantage.

To build a dataset, each data point should correspond to one experiment submitted to a particular damage scenario. This process would take long to be completed experimentally. Therefore, numerical methods have been used as a replacement. This can be done because a well calibrated model can accurately describe the structural element or system and generate a large volume of data, each with a different scenario to work with. In the case of a linear structure, its motion is described by:

$$\boldsymbol{M}\ddot{x} + \boldsymbol{C}\dot{x} + \boldsymbol{K}x = \boldsymbol{f}(t), \tag{2}$$

where $\boldsymbol{M}$, $\boldsymbol{C}$, and $\boldsymbol{K}$ are mass, damping and stiffness matrices, $\boldsymbol{f}(t)$ is the external force vector, and x represents displacement, with its derivatives. Since transmissibility functions are presented in the frequency domain, Eq. (2) is transformed via Fourier Transform:

$$\left(-\omega^2\boldsymbol{M} + j\omega\boldsymbol{C} + \boldsymbol{K}\right)\boldsymbol{X}(\omega) = \boldsymbol{F}(\omega), \tag{3}$$

where ω is the frequency and $j = \sqrt{-1}$. From Eq. (3), the Frequency Response Function $\boldsymbol{H}(\omega)$ is defined:

$$\boldsymbol{X}(\omega) = \left(-\omega^2\boldsymbol{M} + j\omega\boldsymbol{C} + \boldsymbol{K}\right)^{-1}\boldsymbol{F}(\omega) = \boldsymbol{H}(\omega)\boldsymbol{F}(\omega), \tag{4}$$

$$\boldsymbol{H}(\omega) = \left(-\omega^2\boldsymbol{M} + j\omega\boldsymbol{C} + \boldsymbol{K}\right)^{-1}. \tag{5}$$

$\boldsymbol{H}(\omega)$ can be redefined in terms of $\boldsymbol{X}(\omega)$ and $\boldsymbol{F}(\omega)$:

$$\boldsymbol{H}(\omega) = \frac{\boldsymbol{X}(\omega)}{\boldsymbol{F}(\omega)}. \tag{6}$$

Thus, one can obtain the numerical transmissibilities through the definition:

$$T_{ir}^{k}(\omega) = \frac{\boldsymbol{X}_{ik}(\omega)}{\boldsymbol{X}_{rk}(\omega)} = \frac{\boldsymbol{H}_{ik}(\omega)}{\boldsymbol{H}_{rk}(\omega)}. \tag{7}$$

Transmissibility functions have been widely used in SHM because of their high sensitivity to damage [4–9,14,16,27,39,40]. Fig. 1 shows how transmissibilities are affected by damage. Although the form of the curve remains similar, there is a displacement of its peaks and valleys. This suggests the use of image-recognition algorithms to map these displacements to damage conditions.

2.2. Structural damage

As seen in the Introduction section, structural damage can be represented in more than one way. In this paper, damage is represented by a stiffness reduction, as expressed in Eq. (8):

$$\boldsymbol{K}_{\boldsymbol{i}}^{\boldsymbol{d}} = (1 - o_i)\boldsymbol{K}_i, \tag{8}$$

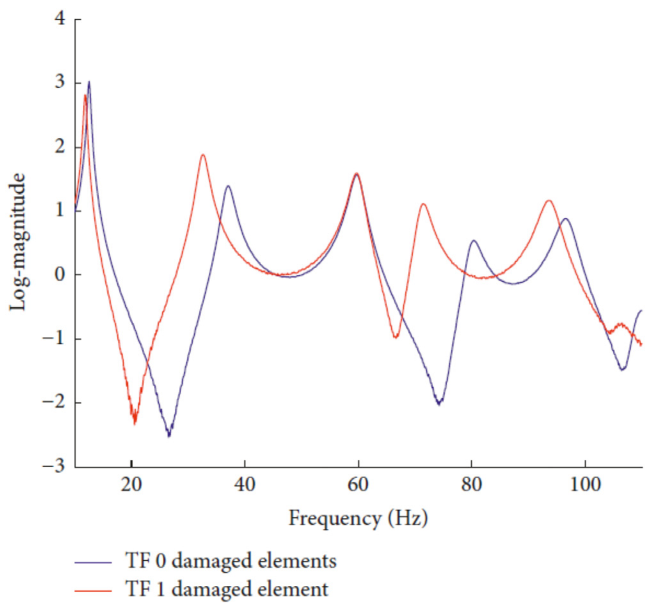

Fig. 1. Transmissibility functions from a spring–mass system. The red line corresponds to the undamaged case, whereas the blue line corresponds to the system with one random damaged element. An element corresponds to a spring–mass pair, and damage is represented by a stiffness reduction.
Source: [27]

where $\boldsymbol{K}_i^d$ is the damaged stiffness matrix of the ith element, $\boldsymbol{K}_i$ is the undamaged stiffness matrix, and o_i is the stiffness reduction of the ith element. This has shown good results in damage detection algorithms [5,14,16,27,41].

3. Capsule networks

Most deep learning algorithms are born as an effort to imitate the way the human brain uses different pieces of information and makes the associative processes that lead to learning. Deep Neural Networks are the clearest example. Each node in the network is called "neuron" because its properties are vaguely similar to actual neurons in the human nervous system. While artificial neurons have a number of input channels and an output that can serve as an input to another neuron, biological neurons have dendrites that act as input channels, a body that processes information, and an axon that connects the output to another neuron.

In the case of CNNs, the original task is to imitate the way the human brain recognizes images. The convolution operation in CNNs performs operations in small windows of an image, just like the visual cortex analyzes a small region of the visual field. To cover the whole image, there is overlap between different regions, while in CNNs there is also an overlap measure that can be tuned.

Over the years, CNNs have proven to be useful and accurate, reaching state-of-the-art results in many fields. However, there is one aspect that makes researchers believe CNNs could work in a better way. The pooling operation that comes after a convolution, used to reduce the number of parameters in the network, and therefore reducing training time, produces positional and translational invariance. Pooling acts as a summary of features in a region of the image, meaning some information is lost. Since all of the features in that region are represented by one value (normally the maximum or the average), changes in that region are not perceived by the next layer. For the network to recognize a translation or rotation on a particular image, it is necessary to feed it with many images at different locations and rotations, and even then, it may not perform successfully, because pooling makes CNNs tolerant to small changes, but it does not really understand those changes.

To tackle the problem of invariance, capsules of neurons are introduced in [29]. This is, in a way, inspired by cortical microcolumns [42]. Microcolumns are sets of neurons organized in vertical columns. The authors in [43] state that a microcolumn plays an important role in object recognition through senses and explain the way it is done. Although it is explained with recognition only through touching, the same explanation can be applied to vision.

According to [43], a single microcolumn being paired to a sensor (a fingertip, for example) generates a location signal as the sensor approaches an object. This location signal activates the neurons in the microcolumn that can decode the signal to recognize a feature. This represents a prediction to what feature is to be sensed. When the object is sensed, some sets of neurons are activated. Neurons that were also activated due to the location signal are propagated to an output layer. These neurons represent all the objects containing the sensed feature at the sensed location. This action is repeated at multiple locations to discard possible elements and to accurately identify the correct object. Multiple microcolumns are interconnected to accelerate this process. In the case of touching an object with fingertips, touching it at different locations with just one fingertip may take some time, but touching with two or more fingertips certainly helps recognize the object much faster and accurately. With vision, the process is similar. To recognize an object, the eye inspects it at different locations. A trained brain would activate neurons according to the agreement between them on what the object should be. This is done very quickly because the eye has lots of sensors, meaning the process is done simultaneously with many microcolumns.

Inspired by the process explained above, CapsNets were proposed in [29]. A Capsule Network is a type of neural network in which, instead of applying a function to each neuron in a layer to define its activation state, a routing algorithm is applied to a whole set of neurons, now referred to as capsules. A capsule non-scalar output represents both the probability of existence of an entity and certain properties of it. As mentioned before, between two adjacent layers of capsules, a routing algorithm decides which capsules to activate, according to an inner iterative process.

These sets of neurons called capsules are inspired on microcolumns. Each capsule contains neurons with features that are extracted using convolutions (these features could also come from a lower-level capsule layer). These neurons evaluate the most possible output for the next layer capsules. The final output is decided based on a measure of agreement between all the predictions. The latter step is called routing by agreement and is an iterative process. The predictions are used to compute the actual output, and the agreement between these two determines the predictions made by capsules in the next iteration, which are used to update the output. This is done a few times (typically, 3–5).

Two different models of CapsNets for classification tasks have been proposed in the literature. In [29], a capsule is a set of neurons represented as a vector. The individual values are to capture features of an object, while the length of the vector shows the capsule activation probability. For the vector to represent a probability, its length value must be between 0 and 1, accomplished by the following squashing function:

$$v_j = \frac{\left\|s_j\right\|^2}{1+\left\|s_j\right\|^2}\frac{s_j}{\left\|s_j\right\|^2} \tag{9}$$

were v_j is the "squashed" value of the capsule output s_j.

The first layer of capsules comes from the output of a convolution. This output is rearranged into vectors with a previously specified dimension (and is shrunk using the squashing function), which are used to compute the output of a next layer set of capsules.

The algorithm with which the next layer capsules are computed using the current layer of capsules outputs is called *dynamic routing*. It takes predictions from the current level capsules about the output of the next layer capsules and computes the actual output according to an agreement metric between predictions.

The predictions about the next layer capsules are calculated by a multiplication with a matrix of weights:

$$\hat{\boldsymbol{u}}_{j|i} = \boldsymbol{W}_{ij}\boldsymbol{u}_i \tag{10}$$

where $\boldsymbol{u}_i$ is the output of capsule i in the current layer, $\boldsymbol{W}_{ij}$ is the weights matrix between capsule i in layer l and capsule j in layer $l+1$, and $\hat{\boldsymbol{u}}_{j|i}$ is the predicted output of capsule j given the output of capsule i. The output of capsules s_j in layer $l+1$ corresponds to a weighted sum over all $\hat{\boldsymbol{u}}_{j|i}$ (and shrunk using the squashing function):

$$s_j = \sum_i c_{ij}\hat{\boldsymbol{u}}_{j|i} \tag{11}$$

$$v_j = \frac{\left\|s_j\right\|^2}{1+\left\|s_j\right\|^2}\frac{s_j}{\left\|s_j\right\|^2} \tag{12}$$

where c_{ij} are called *coupling coefficients* and are calculated as:

$$c_{ij} = \frac{\exp(b_{ij})}{\sum_k exp(b_{ik})} \tag{13}$$

These coupling coefficients change iteratively because the b_{ij} logits are updated through the following:

$$b_{ij} \leftarrow b_{ij} + \hat{\boldsymbol{u}}_{j|i}\cdot v_j \tag{14}$$

The expression $\hat{\boldsymbol{u}}_{j|i}\cdot v_j$ measures the *agreement* between the actual output in layer $l+1$ and the prediction made by a capsule in layer l.

Eqs.(11), (13) and (14) are the core of the routing process. For each capsule in the next layer, the predicted outputs from all of the capsules in the current layer are combined linearly, as shown in Eq. (11). This combination obtains the output. The weights in the equation, $c_{i,j}$, represent the importance of each capsule prediction in the calculation of the capsules in the next layer. These weights are updated iteratively using Eqs. (13) and (14). The process of assigning the relevance of each capsule in the next layers capsules is referred to (by the authors in [29]) as *routing*. Fig. 2 shows a simplified representation of this process.

In [31], capsules are no longer seen as one entity, but two: a pose matrix and an activation unit. The pose matrix captures the pose of the image in space so that the algorithm recognizes rotated images, with one image rotated in different angles and not different objects, thus requiring less data to train and to achieve desirable results. The activation unit, similar to the length of the vector in a vector-like capsule, represents the probability of existence of a feature.

Similarly to the previous work on Capsule Networks [29], the first instance capsules appear after a convolutional layer. In this case though, not only is a rearrangement (*reshape* operation) necessary. To obtain the pose matrix, another convolution is done to the values obtained through previous convolutions, with a ReLU activation function. The result is rearranged to a matrix shape. To obtain the activation value, another parallel convolution is performed to get a single value output.

To get the votes (predictions made by the capsules in layer l regarding the capsules in layer $l+1$), the capsules are multiplied

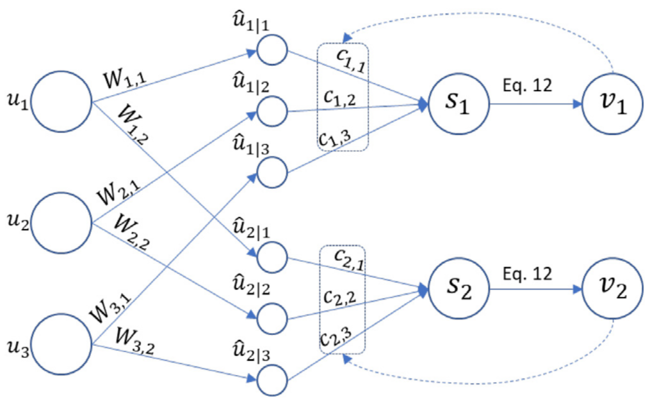

Fig. 2. Routing process example between a layer with three capsules and a layer with two capsules.

by a weights matrix modified during the training process. These votes are used to compute the output for the capsules in layer $l+1$, using an algorithm called *EM Routing*, where *EM* stands for *Expectation–Maximization*. This algorithm assigns capsules in layer l to clusters. Each cluster represents a capsule in layer $l+1$ and follows a Gaussian distribution. The mean of each Gaussian distribution μ is the output of each capsule in layer $l+1$.

The algorithm alternates between the E-step (Expectation) and the M-step (Maximization). During the E-step, assignment probabilities $\boldsymbol{R}_{ij}$ are calculated, which are the probabilities that capsule i in layer l is assigned to capsule j in layer $l+1$. This is done through the following equations:

$$\boldsymbol{p}_j = \frac{1}{\sqrt{\prod_h^H 2\pi\left(\sigma_j^h\right)^2}}\exp\left(-\sum_h^H\frac{\left(V_{ij}^h-\boldsymbol{\mu}_j^h\right)^2}{2\left(\sigma_j^h\right)^2}\right) \tag{15}$$

$$\boldsymbol{R}_{ij} = \frac{\boldsymbol{a}_j\boldsymbol{p}_j}{\sum_k \boldsymbol{a}_k\boldsymbol{p}_k} \tag{16}$$

In the expression for $\boldsymbol{p}_j$, V_{ij}^h is the hth dimension of the vote from capsule i to capsule j, $\boldsymbol{\mu}_j^h$ and σ_j^h are the hth dimension for the mean and variance of capsule j. For $\boldsymbol{R}_{ij}$, $\boldsymbol{a}_j$ is the activation value for capsule j (the sum appearing in the denominator uses all the capsules in layer $l+1$). This value is calculated in the M-step. In this step, the clusters characterizations are modified by updating $\boldsymbol{\mu}_j^h$ and σ_j^h. Also, $\boldsymbol{R}_{ij}$ is updated:

$$\boldsymbol{R}_{ij} = \boldsymbol{R}_{ij} * \boldsymbol{a}_i \tag{17}$$

$$\boldsymbol{\mu}_j^h = \frac{\sum_i \boldsymbol{R}_{ij} * V_{ij}^h}{\sum_i \boldsymbol{R}_{ij}} \tag{18}$$

$$\left(\sigma_j^h\right)^2 = \frac{\sum_i \boldsymbol{R}_{ij} * \left(V_{ij}^h - \boldsymbol{\mu}_j^h\right)^2}{\sum_i \boldsymbol{R}_{ij}} \tag{19}$$

The clusters are redefined to minimize a cost function that considers the probabilities that each cluster generates the values shown by the capsules in layer l. That cost function is used to update the activation value for each capsule in layer $l+1$:

$$cost^h = \left(\beta_u + \log\left(\sigma_h^h\right)\right)\sum_i \boldsymbol{R}_{ij} \tag{20}$$

$$\boldsymbol{a}_j = logistic\left(\lambda\left(\beta_a - \sum_h cost^h\right)\right) \tag{21}$$

where the values β_u, β_a, are learned to minimize the cost function, and λ decreases with each iteration according to a fixed rate.

EM routing is more complex than dynamic routing. However, the underlying idea is the same: for capsules to make predictions (in this case, $V_{i,j}$), and use them to calculate the real output of the next layer capsules. The approach is to see those capsules as multivariate Gaussian distributions and update their parameters through the *Expectation–Maximization* algorithm. Fig. 3 shows how this algorithm builds the next layer capsules. Firstly, capsules in the next layer are initialized. After the routing process, the mean and variance of every capsule are determined using the votes according to their assignment probabilities, $\boldsymbol{R}_{ij}$.

4. Case studies and datasets

In this section, the two case studies discussed herein are presented, followed by a description of the approach used for generating the datasets. The first case corresponds to a simple experimental structure with 8 degrees of freedom. The level and location of damage introduced in this structure is known. Therefore, this case study allows assessing the precision in the damage identification. Meanwhile, the second case corresponds to a beam with individual and multiple cracks. In this case, it is not straightforward to estimate the actual stiffness reduction, but it allows evaluating the algorithm in a more realistic structure. Both experimental case studies have been selected due to their availability on the internet and because they have been widely used as benchmark problems to evaluate damage identification algorithms [5,16,27,44–46].

4.1. Case study 1: Spring–mass system

The first case study corresponds to a spring–mass system designed by Los Alamos National Laboratory (LANL) [47]. It consists of eight aluminum disk masses connected by seven springs. Each mass is a disk of aluminum with a diameter of 76.2 mm and a thickness of 25.4 mm. The masses can slide freely along a center rod that provides support to the whole system and constrains the masses to translate along the rod. Boundary conditions are unrestrained. Fig. 4 illustrates the experimental test bed; springs and mass locations are designated sequentially with the first ones closest to the shaker attachment.

The excitation force comes from a shaker connected to the first mass, which provides a random excitation force. One accelerometer is connected to each mass. These accelerometers measure horizontal acceleration data and are used to calculate transmissibility functions. Data is acquired with a frequency resolution of 0.125 [Hz], in the range of 10–110 [Hz]. The resulting dataset is public and available at [48], which is where the authors of this paper obtained it from.

The numerical model is built in Matlab with linear springs and concentrated masses. The model parameters are as set as follows:

- Mass 1: 559.3 g (This mass is greater than the others because of the hardware required to attach the shaker).
- Masses 2 to 8: 419.4 g
- Spring constants: 56.7 kN m^{-1}

This system is adequate for testing damage identification algorithms because damage can easily (and accurately) be represented by a spring having less stiffness than the others. In the undamaged configuration, all springs are identical and have a linear spring constant equal. Here, damage is simulated by replacing the fifth spring with another spring with lower stiffness (55% reduction in stiffness), this means that the fifth element is damaged with $y_5 = 0.55$.

The models proposed in Section 5 (see Tables 4 and 5 for details) are trained using data generated from the numerical model for the mass–spring system and are then tested with the experimental damaged case.

Table 1
Damage scenarios for the beam case study — experimental cases.

Damage Scenario	Number of cuts	Distance from the left side [mm]	Damaged Element	Cut depth [mm]
1	1	637	13	9
2	2	361	8	8
		812	17	15
3	3	363	8	13
		574	12	8
		696	14–15	6

4.2. Case study 2: Beam

In [44], an experimental setup for damage assessment is proposed. The structure consists of a steel beam with a rectangular cross-section. The dimensions of the beam are length = 1 m and section = 25 × 10 mm^2. As in the first case, a shaker is connected to one end of the beam to generate a random excitation force, and the structure is suspended with two springs holding it. These springs have low stiffness to simulate a “free-free” scenario. Eleven accelerometers are connected to the beam to measure vibrations and to calculate transmissibilities. There are different techniques that have been proposed to determine the optimal vibration sensor placement for SHM applications. These techniques intend to maximize the information captured by a limited number of sensors, in particular information related to the different natural frequencies and mode shapes [49]. In the case of a beam with free-free boundary conditions, the best option is an even distribution of the sensors. A larger number of sensors will allow capturing information from higher frequencies modes, which are more sensitive to local damage. In this case, the eleven vibration sensors were placed evenly distributed on the beam. Data is acquired with a frequency resolution of 1 [Hz], in the range of 1–2000 [Hz]. The experimental setup is illustrated in Fig. 5.

To build the Finite Element (FE) model, unidimensional beam elements are used, with two nodes per element and two degrees of freedom per node. As illustrated in Fig. 6, The beam is divided into 20 elements of 5 [cm] each.

The structure is subjected to different damage scenarios containing single and multiple cracks. Cracks are introduced to the structure by saw cuts as illustrated in Fig. 7. As in the previous case study, here the models proposed in Section 5 (see Tables 4 and 5 for details) are tested with three experimental cases, described in Table 1. This table indicates the distance from the left-end to the cut, the corresponding element in the numerical model and the cut length. For this case study, these scenarios compose the test set, which are used after the training process to evaluate the different models in real-life situations.

Note that while in the first case study a stiffness reduction of a spring implies a damage by the same amount of the reduction, in this case, the saw cuts inflict stiffness reduction and, therefore, damage, but this damage value is unknown. For example, a cut depth of 15 [mm] (of a total depth of 25 [mm]) does not necessarily mean the element suffers a 60% stiffness reduction.

4.3. Datasets

To train the models presented herein, transmissibility functions are used. These are represented by grayscale images. To obtain them, the logarithmic magnitude of the different transmissibilities is scaled to the range of 0–255. Thus, each pixel intensity represents a different logarithmic magnitude at a specific frequency.

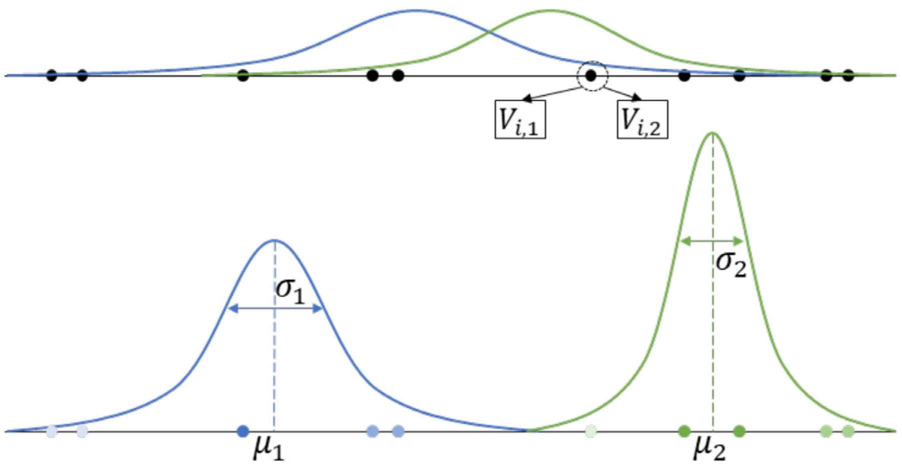

Fig. 3. EM routing example, with 10 capsules in the current layer and two capsules in the next layer. For simplicity, $h = 1$. In the upper part, capsules before routing are displayed. In the lower part, the product of the routing process is shown.

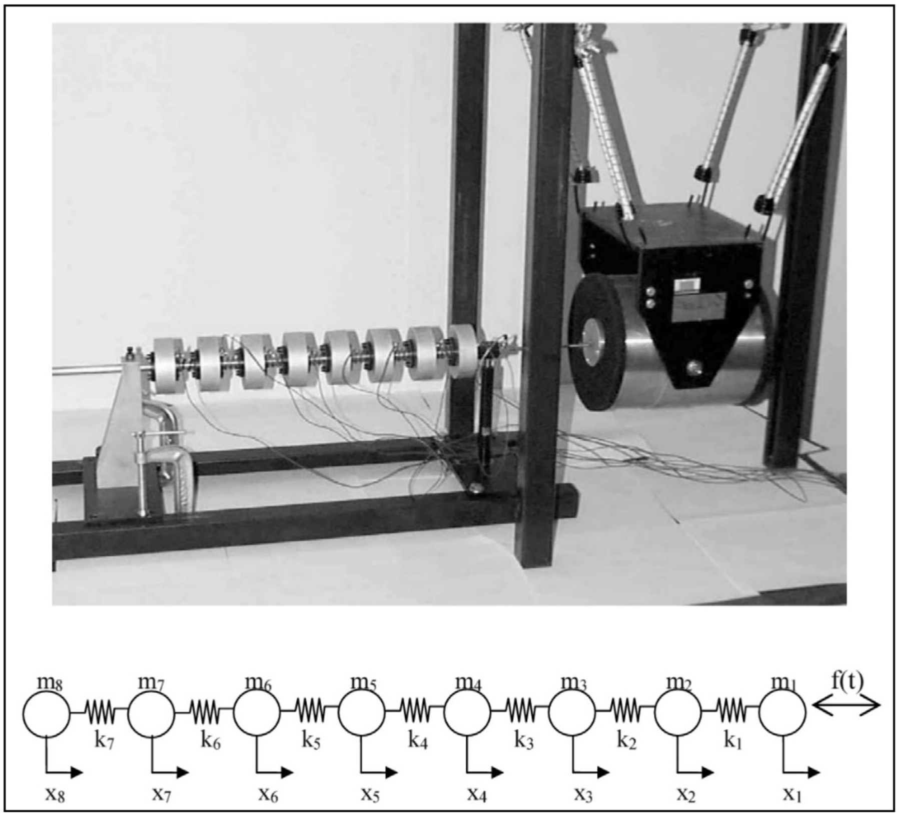

Fig. 4. 8 DOF Spring–Mass system.

Since more than one transmissibility function is measured per each datapoint, each row in the image represents one transmissibility. For both case studies, 10 transmissibilities are calculated in a frequency range of 0–2000 [Hz]. Each image is 10×96 in size. The frequency domain is converted using a bi-cubic interpolation. Fig. 8 shows an example of 10 transmissibility functions from a structural beam, while Fig. 9 shows those functions transformed into image format. As seen in Section 2.1, the relation between transmissibility functions and damage is a displacement of peaks and valleys, shown in Fig. 9 in white and black colors, respectively. This means that the damage information (number of damaged elements, location and amount of damage per element) is contained within each image. To date, the exact relationship between transmissibility functions and structural damage is unknown. This is why image recognition techniques are suitable for this problem.

Deep learning algorithms take advantage of large amounts of data. Since it is unfeasible to perform thousands of measurements for both case studies, training images are generated via a finite element model calibrated with experimental measurements, which are used for testing. For both case studies, four types of images are generated according to the number of damaged elements, as shown in Table 2. First, 10,000 images simulating structures with no damaged elements are generated using the FE model. Then, 30,000 images are generated, simulating structures with one, two and three damaged elements, respectively. Each damaged element has a stiffness reduction selected randomly.

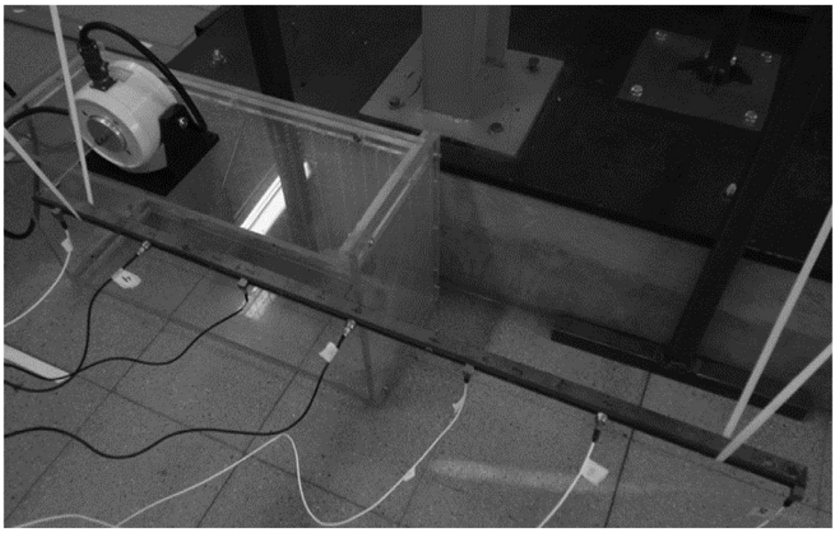

Fig. 5. Beam setup.

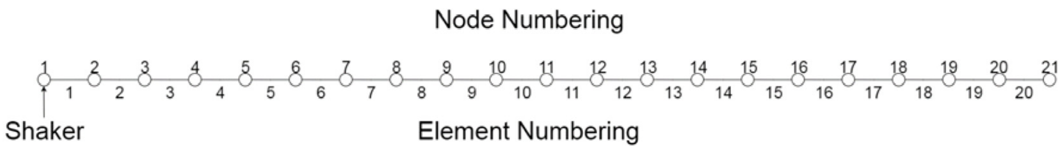

Fig. 6. Beam element numbering.

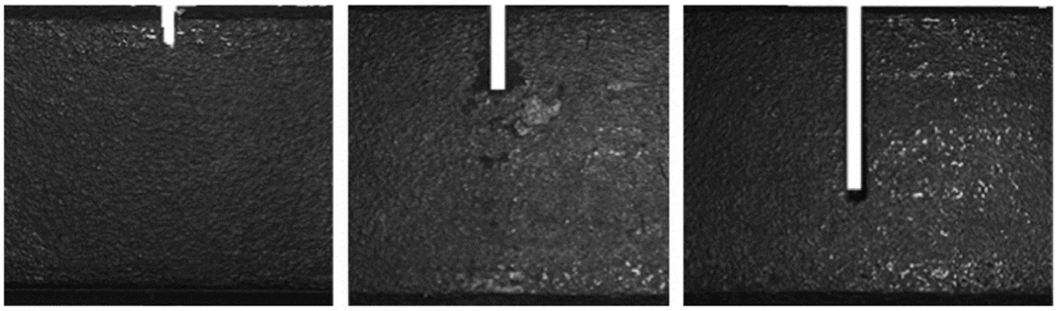

Fig. 7. Beam cut examples.

Table 2
Types of training images.

Damaged elements	Number of images
0	10,000
1	30,000
2	30,000
3	30,000

Table 3
Training Datasets.

Dataset	Types of images	Number of images
DS1	0 and 1 damaged elements	40,000
DS2	0, 1, and 2 damaged elements	70,000
DS3	0, 1, 2, and 3 damaged elements	100,000

After that, noise is injected, with values up to 6%, also selected randomly. This simulates sensor noise. According to [50], sensor noise does not exceed 6% when measuring vibrations.

With these four kinds of images, three datasets (DS) are proposed for each case study, as shown in Table 3. Even though datasets are unbalanced regarding the number of damaged elements each image represents (see Table 2), this is not an issue for the development of our work. First, images with damaged elements have different amounts of damage located at different places within the structure. This generates datasets with great variability. Second, our work's objective is to characterize damage correctly, and not to classify images according to the number of damaged elements its structure represents.

Each model is trained and validated with one of these datasets, using 85% for training, and 15% for validating. Then, the trained models are evaluated in the experimental scenarios detailed in Sections 4.1 and 4.2.

5. Proposed capsule neural networks for damage localization and quantification

The proposed capsule neural networks aim at both structural damage localization and quantification. Note that standard (also called vanilla) Capsule Neural Networks were presented initially for image classification purposes. We have found only one application with regression [37] for estimating the remaining useful life of turbofans. Thus, we extend the vanilla CapsNets to carry out

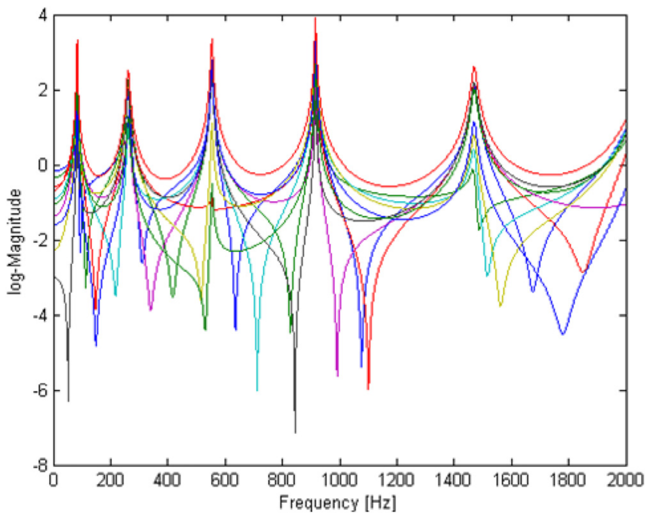

Fig. 8. Transmissibilities measured in a beam.

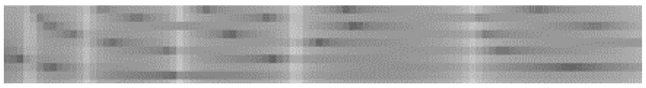

Fig. 9. Image representation used as input.

both damage localization (i.e., classification) and quantification (i.e., regression).

Indeed, vanilla CapsNets have two main parts: convolutions and routing. After a determined number of convolutions, capsules go through a routing process to calculate next level capsules, both their properties and activations. The capsules in the last level represent the network output, which are classes (e.g., health states). The capsule with the highest activation value represents the predicted class (multiple classes could also be accepted; in that case, the loss function should be modified). Note that this approach only works for structural damage localization, because each capsule would represent an element within the structure: if the capsule activation value is near to 1, the element presents damage. However, this method does not work for quantifying damage (i.e., for regression), because the capsules activations represent the probability of existence of a feature; in this case, structural damage. There is no output value that could represent and assess the amount of damage. To overcome this limitation, the proposed capsule networks include an MLP that uses the last level capsules output as input. The MLP output describes the amount of damage in every element in the structure. The network is used for supervised learning tasks; therefore, the loss function to be optimized during learning compares the output of the MLP after the capsules with the labels.

Fig. 10 shows the approach for damage localization and quantification. The input data structure (e.g., transmissibility image) is submitted to two convolution layers. Then, the primary capsules are obtained. The method for this depends on the type of capsules (matrix or vector). Then, through a routing process (which can be dynamic routing or EM routing), secondary capsules are obtained. The number of secondary capsules depend on what they are meant to represent. Finally, information from the secondary capsules is used as input for an MLP, which in turn calculates the amount of damage per element in the structure.

In this paper, both routing algorithms are explored and analyzed in the context of damage localization and quantification. The idea behind CapsNets is to recognize hierarchies within an image. Both capsule representations achieve this because the grouping of features in one unit helps recognize more complex features in the upper layers of the model. Note that in damage detection and measurement using transmissibility functions, not only the intensity of each peak matters, but also its location within the frequency range, as seen in Fig. 1.

The use of CapsNets for this problem is more appropriate than CNNs because they are better at recognizing spatial relationships within the image than CNNs. In CNNs, higher-level features are obtained out of linear combinations of low-level features that are passed through an activation function, thus not considering the interactions *between* the features within the image. CapsNets address this issue by storing pose information in each of the capsules. Since capsules are no longer scalar values but vectors or matrices, they store information about the relative position of the feature within the image and other features [29]. They send their information to higher-level capsules that “agree” with their own information. This ensures that capsules in the next layer consider not only the features of the lower-level capsules, but also their interactions with each other and their position within the image, establishing a spatial hierarchy of features. This is useful for our SHM problem because not only is the shape of the transmissibility curves important but also their displacement within the frequency range and the distance between peaks and valleys. While CNNs focus more on recognizing the shape of the transmissibility curves and have some trouble with dealing with displacements, CapsNets should address this issue through non-scalar entities and routing algorithms. The spatial hierarchy obtained from this process should deal with the relations between peaks and valleys within the frequency range (which, as explained in Section 2.1, are directly related to damage properties of the structure) in a better way than CNNs do, since it establishes hierarchical relations between features, which is what is needed for a better characterization of damage from transmissibility functions.

As mentioned before in this section, the original CapsNets algorithms introduced in [29] and [31] were presented as algorithms for image classification. In both algorithms, the output layer is comprised of capsules, and the one with the highest activation value determines the class the input image belongs to. It would be unfeasible to use this same configuration to locate and quantify structural damage in the way the data is presented. To solve this issue while keeping the advantages of CapsNets, the algorithm is extended with fully connected layers that take the last capsule layer as input, in order to locate damaged elements and quantify the amount of damage. Even though fully connected layers are also present in CNNs (and are applied in this way in [27]), they receive features as inputs, which means there is no classification tasks within the convolution operations, as is the case with routing in CapsNets. We postulate that this should help in obtaining more accurate results, as shall be corroborated by the results presented in Section 6.

For CapsNets with dynamic routing, the corresponding proposed general model is shown in Fig. 11.

The input image is convoluted using 32 filters, thus generating 32 feature maps. To capture all of the transmissibility functions information in one filter, each of them has a dimension of 10×1 and a stride of 1. Then, a second convolution is performed, this time generating 64 feature maps. The filters size is 1×5 with a stride of 1. After both convolution layers, a ReLU activation function is applied to inject nonlinearity to the model. Before the primary capsules, a reshape operation is conducted to the 64 feature maps obtained by the last convolution, which turns each of the $1 \times 91 \times 64$ neurons with a scalar output into $1 \times 91 \times 8$ 8-dimensional vectors. A squashing function is applied to all capsules to ensure their lengths represent activation probabilities. Routing, as described in [29], is then performed to compute the secondary capsules values. Besides varying the

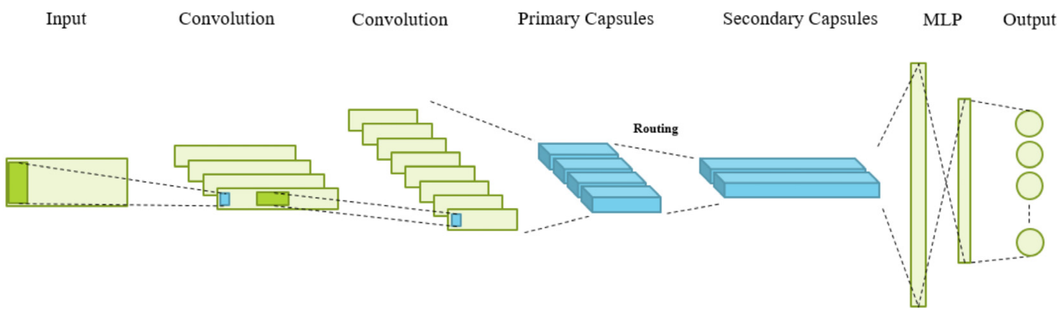

Fig. 10. Proposed Capsule Networks general approach.

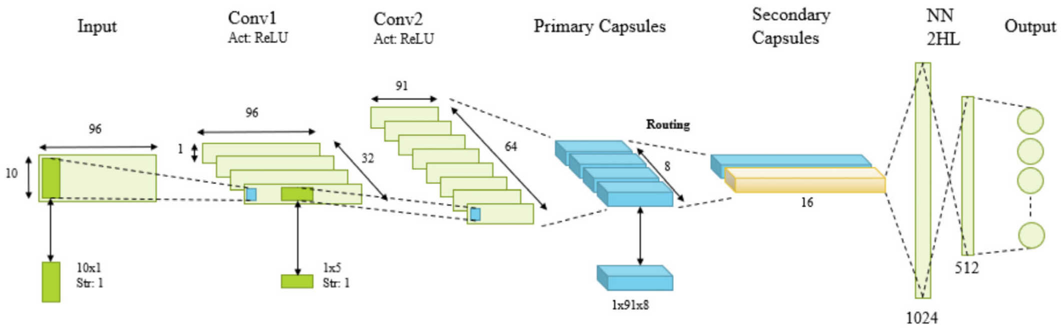

Fig. 11. Capsule Networks with dynamic routing, proposed general model.

Table 4
Models description.

Model name	Case study	Routing	Dataset	Dropout
SM_DR_DS1	Spring–Mass (SM)	Dynamic (DR)	1 (DS1)	No
SM_DR_DS2			2 (DS2)	
SM_DR_DS3			3 (DS3)	
SM_EMR_DS1		EM (EMR)	1	
SM_EMR_DS2			2	
SM_EMR_DS3			3	
B_DR_DS1	Beam (B)	Dynamic	1	
B_DR_DS2			2	
B_DR_DS3			3	
B_EMR_DS1		EM	1	
B_EMR_DS2			2	
B_EMR_DS3			3	
B_DR_DS1_D		Dynamic	1	Yes (D)
B_DR_DS2_D			2	
B_DR_DS3_D			3	

number of routing iterations for achieving the best results, the original routing algorithm is applied as presented in [29].

Note that the number of secondary capsules depends on the dataset (see Table 3). For CapsNets with dynamic routing, there are three types of specific models per case study, one for each dataset. For dataset DS1, there are only two possible health states: undamaged and one damaged element. Therefore, there are two secondary capsules. Following the same logic, for datasets DS2 and DS3, there are 3 and 4 secondary capsules, respectively. The objective is for secondary capsules to detect what kind of scenario (in terms of number of damaged elements) the image represents. Thus, the capsule with the greatest length shows what kind of scenario the model predicts to correspond to an input image, and its properties are chosen to be fed to a neural network, the rest of them being discarded. This is an implicit classification task. This capsule is then fed to a neural network with two hidden layers, the first with 1024 hidden units and the second one with 512, both with ReLU activation functions. The final output is a N-dimensional vector, with N being the number of elements in the structure. Each neuron in the layer represents the amount of damage in the corresponding element.

In the case of EM routing, the proposed general model is shown in Figure Fig. 12. The first convolution is equal to the one used for dynamic routing. The second convolution layer is comprised of 32 1 $\times$ 6 filters with a stride of 3 to reduce the network size and to capture abstract features using overlapping. As shown in Section 2, each capsule contains an activation value and a pose matrix. The pose matrix represents the capsule features, which correspond to complex features in the transmissibility functions images, whereas the activation value represents the probability of existence of those features in the pose matrix. To obtain the primary capsules pose matrices, the output from the second convolution layer (Conv2) is submitted to a convolution with no activation function. This generates 4 $\times$ 4 matrices. To obtain the activation values, the Conv2 layer output is simultaneously submitted to a convolution with a sigmoid activation function. This convolution generates numbers between 0 and 1, which are meant to represent probabilities of existence of the features in the corresponding capsule.

After obtaining the primary capsules, EM routing is performed to obtain both the activation values and pose matrices for the secondary capsules layer. As with dynamic routing, the EM routing algorithm is applied exactly as described in [31], with the exception of the variation of the number of routing iterations in the fine tuning process to achieve the best results possible. The number of secondary capsules in the layer corresponds to the number of nodes in the structural element. For the first case study (spring–mass system), there are seven secondary capsules, whereas for the second case study (beam), there are 18 secondary capsules. All the information contained in those capsules is used as input to an MLP with 2 hidden layers with ReLU activation functions. The final output corresponds to the amount of damage in every element of the structure.

In this paper, various models based on the general models shown in Figs. 11 and 12 are developed, as shown in Tables 4 and 5. Their configurations are determined via grid search, in which

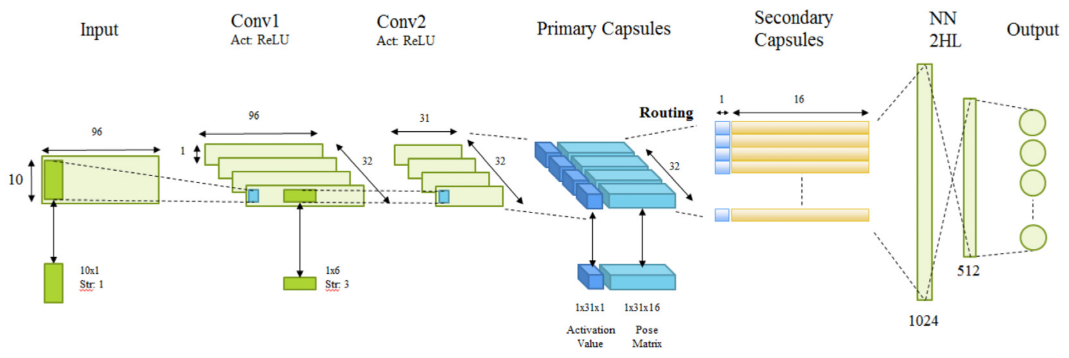

Fig. 12. Capsule Networks with EM routing, proposed general model.

several configurations are tested to see which one achieves the best results. Number of layers, types of layers, neurons per layer, activation functions, capsule sizes, capsules per layer, number of routing iterations and weights initializers are the grid search parameters.

During training, for each model, the mean squared error is used as cost function:

$$C = \frac{1}{NO}\sum(y_i - o_i)^2 \tag{22}$$

where y_i is the estimated output, o_i is the observed output and NO is the number of nodes. This cost function is appropriate for the task because it compares, node by node, the network output with the real damage for each training image, and it stimulates the algorithm to allocate damage to the correct nodes.

To evaluate the proposed models' performance, proper metrics must be defined. Since the tasks are to locate and to quantify structural damage, performance is measured in terms of how the different models are able to locate and to quantify damage. Thus, to assess the models' capacity to quantify damage, *Mean Sizing Error* (MSE) is used:

$$MSE = \frac{1}{NO}\sum |y_i - o_i| \tag{23}$$

where NO is the number of output nodes, y_i is the estimated output for node i, and o_i is the real output for node i. For localization performance measurement, *Damage Missing Error* (DME) and *False Alarm Error* (FAE) assess the fraction of damaged elements unnoticed and undamaged elements tagged as damaged, respectively. DME is given by:

$$DME = \frac{1}{NT}\sum \epsilon_i^l \tag{24}$$

where NT is the number of true damaged elements and ϵ_i^l is defined by:

$$\epsilon_i^l = \begin{cases} 1, & y_i \leq \alpha_c, o_i \geq 0 \\ 0, & \sim \end{cases} \tag{25}$$

To consider a damaged element as detected, y_i must be greater than a limit value α_c, which is considered as $\alpha_c = MSE$. FAE is given by:

$$FAE = \frac{1}{NF}\sum \epsilon_i^u \tag{26}$$

where NF is the number of predicted damage locations and ϵ_i^u is given by:

$$\epsilon_i^u = \begin{cases} 1, & y_i \geq \alpha_c, o_i = 0 \\ 0, & \sim \end{cases} \tag{27}$$

When training a model, the three performance metrics should all be as small as possible. Since there is no standard to determine whether a model has acceptable performance metrics or not, one must make comparisons between models to determine which one is more suitable. However, having a clear understanding on what each metric represents leads to a better analysis of the results. Indeed, MSE gives a measure of how accurate the damage quantification for each damaged element is, while DME and FAE indicate the number of false negatives (a damaged element being unrecognized by the model) and false positives (undamaged elements assessed as damaged), respectively. These last two metrics generally have competing behaviors, meaning that a model presenting a low DME will most likely have a high FAE, and vice versa. In terms of structural integrity and safety in SHM, a high DME value is more detrimental than a high FAE, since a false negative means a potentially dangerous situation is being unnoticed, which could lead to an accident. In this sense, appropriate models must have lower DME values than FAE.

Furthermore, we also explore the impact of regularizing the proposed models via dropout. Models B_DR_DS1_D, B_DR_DS2_D and B_DR_DS3_D are trained using dropout in the first hidden layer of the neural network for regression, with $p = 0.5$.

The summarized methodology shown in this section is illustrated in Fig. 13. It shows how, at the beginning, data is generated through a FE model calibrated with experimental data. Then, data is organized into the different dataset shown in Table 3. For each of those datasets, training (85%) and validation (15%) sets are randomly defined. While the training set is used to adjust the network trainable weights, the validation set is used in every epoch to see how the model behaves in a dataset which does not intervene in the training process, to look for overfitting. After training, performance metrics are obtained and each model is tested using the experimental cases. Testing in real experimental cases shows if the models, despite being trained using images generated by a FE model, can be applied to recognizing and quantifying damage in real-life situations.

6. Results and discussion

This section reports and discusses the results from training each of the 15 models developed in the previous section. Each model is analyzed in terms of training with the datasets generated through a finite element method and testing with experimental data.

6.1. Case study 1: Spring–mass system

Table 6 shows the training results for the proposed models for both routing alternatives, using numerical data. *DME* and *FAE* metrics are strongly related to the capability of the model to locate damaged elements. *DME* is particularly important for structural reliability as it measures the false negatives given by the model. A high DME means the model has greater probability of not detecting a damaged element, leads to an important safety

Table 5
Model layers specifications.

Model name	Input	Conv1	Conv2	Primary capsules	Secondary capsules	HL1	HL2	Output
SM_DR_DS1	10 × 96	1 × 96 × 32	1 × 91 × 64	1 × 91 × 8 × 8	16 × 2	1024	512	7
SM_DR_DS2					16 × 3			
SM_DR_DS3					16 × 4			
SM_EMR_DS1			1 × 31 × 32	1 × 31 × (16+1) × 32	16 × 7			
SM_EMR_DS2								
SM_EMR_DS3								
B_DR_DS1			1 × 91 × 64	1 × 91 × 8 × 8	16 × 2			18
B_DR_DS2					16 × 3			
B_DR_DS3					16 × 4			
B_EMR_DS1			1 × 31 × 32	1 × 31 × (16+1) × 32	16 × 18			
B_EMR_DS2								
B_EMR_DS3								
B_DR_DS1_D			1 × 91 × 64	1 × 91 × 8 × 8	16 × 2	1024(D*)		
B_DR_DS2_D					16 × 3			
B_DR_DS3_D					16 × 4			

Table 6
Training performance metrics, first case study.

Model	*MSE* [%]	*DME* [%]	*FAE* [%]	Time per epoch [s]
SM_DR_DS1	0.045	0.37	22.8	22.6
SM_DR_DS2	0.091	0.94	16.2	49.0
SM_DR_DS3	0.209	1.58	9.77	101.4
SM_EMR_DS1	0.067	0.33	48.92	46.1
SM_EMR_DS2	0.172	1.36	18.83	81.7
SM_EMR_DS3	0.253	1.46	16.45	116.8

issue. In this regard, each of the six models show *DME* values under 1.6%. This shows that the proposed model considers localization of damage during the optimization stage, even though the cost function nature is primarily related to quantification. The minimum value the cost function can achieve is when not only the quantity of damage in the structure is well identified, but also when it is located in the correct spots, because it evaluates the difference between the real and predicted outputs in every element.

In turn, the *FAE* metric measures the number of false positives, and in the six models, the maximum value reaches 49%. As models get more complex (in terms of the kind of images used for training), the *FAE* values decrease. This is explained by the fact that in those models there are images with more damaged elements, in which case it is less likely for them to give a false negative. A data point with 3 damaged elements can only have 4 false positives, whereas one with 1 damaged element can have 6 false positives.

Comparing CapsNets with dynamic routing and with EM routing, Table 6 shows that results are very similar in terms of performance metrics, except for *FAE*. CapsNets with dynamic routing show, for all models, less false positives than those with EM routing, with a mean value of 36% less. Training time also establishes a difference as models with dynamic routing take much less time in training that those with EM routing.

With the training process completed, damage predictive capabilities are tested based on the experimental results discussed in Section 4.1, i.e., for the case of detecting the fifth damaged element in the spring–mass system. Indeed, as shown in Fig. 14, the Capsule Networks models with both routing algorithms successfully identify and quantify the 5th damaged element. However, when the models are trained to detect scenarios with two and three damaged elements in the structure, an increase in false positives is observed, i.e., the *FAE* value increases, although training results show the opposite. This can be explained by the fact that false positive values increase in their magnitude, not in their quantity. It is not that there are more false positive values, but that those fewer values are more perceptible.

The predictive capabilities of the CapsNets based models are also compared against a Convolutional Neural Network based model, presented in [27]. This CNN model is trained and tuned based on the same case studies and datasets. This makes this CNN model suitable for comparison with the models and results presented in this work. The CNN model's architecture is as follows: a first convolution with 32 filters of 10 × 1, followed by a second convolution with 64 filters of 1 × 5; this is followed by a fully connected neural network layer with 1024 hidden units. This configuration is quite similar to the Capsule Networks models without the routing part, in order to show how routing impacts the final model predictive capabilities. From the experimental results reported in Fig. 10 in the case with 55% damage reduction in the fifth element, note that all three models achieve similar results as they are able to correctly quantify (within a range of ±7%) the damaged element with no false negatives. False positive values appear mainly adjacent to the damaged element. Even though the spring–mass system is not a continuous one, each spring is connected to two masses; thus, a stiffness reduction in one spring will affect the vibrations of two masses. Then, a model may interpret this as a stiffness reduction in the adjacent element, leading to a false positive. However, other false positive values that are not near that region are due to a model error, i.e., although every kind of false positive is an error, the latter kind is associated to a deficiency of the model itself and thus not explained by physical reasons.

Also note that the CNN-based model presents more false positives than the CapsNets-based ones. When using the models trained to detect two and three damaged elements (i.e., trained with datasets 2 and 3, respectively), Fig. 14 shows that the total percentages of false positives in the CNN model add up to 23% and 18%, respectively. For CapsNets with dynamic routing, these values reach 10% and 8%. When using EM routing, the values are 9% and 7%, respectively. Moreover, false positives coming from the CNN-based model are more distributed along the spring–mass system than those from the CapsNets model. For example, in Fig. 14(c), a 5% false positive is shown in the first element of the system, whereas the two CapsNets models do not present any amount of damage in that element.

In terms of generalization capabilities, neither CNN nor CapsNets models suffer from overfitting: all the models are capable of identifying the correct damaged element and to measure damage accurately. All the models present false negatives, but they are within the range of 0%–10% damage.

Based on the results above, one can argue that the CapsNets-based models are competent in performing localization and quantification of structural damage in the spring–mass system, as it is also the case for the CNN, but to a lesser degree.

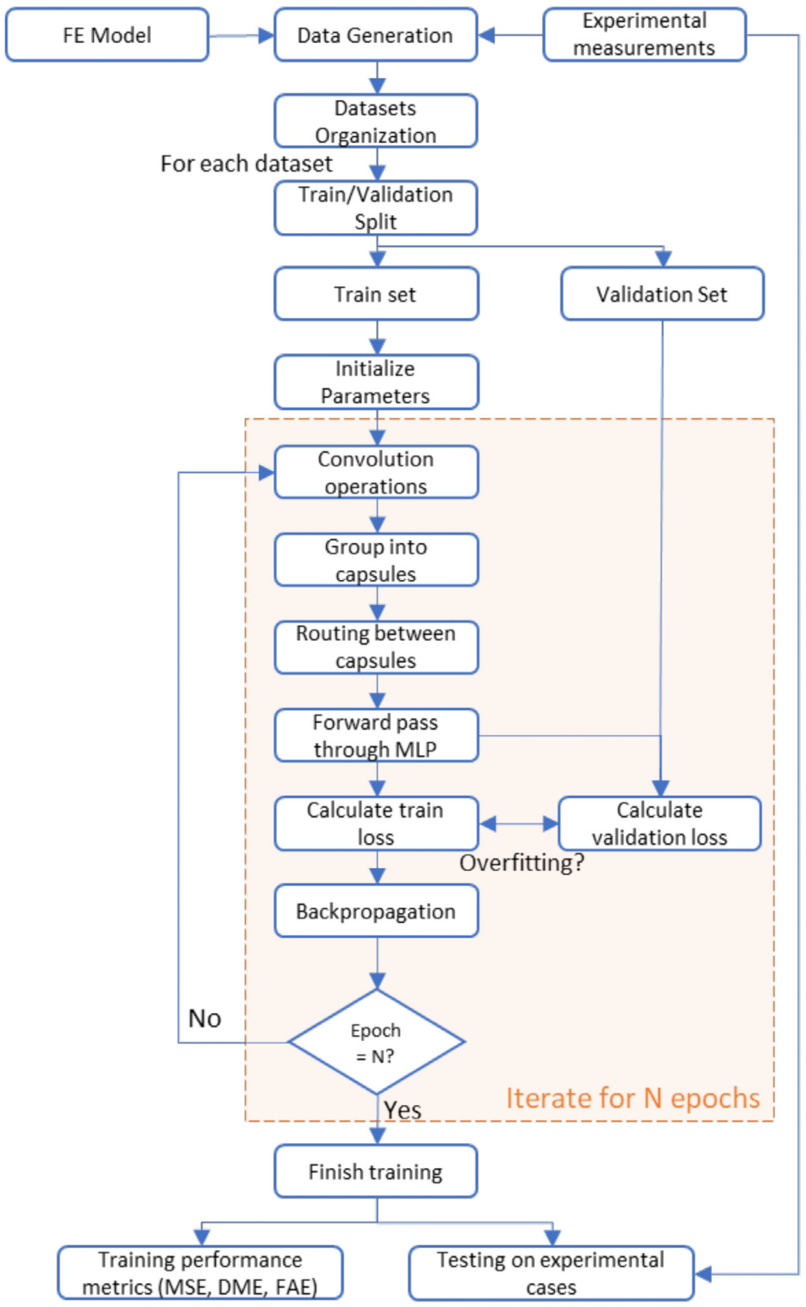

Fig. 13. Methodology for damage localization and quantification using CapsNets.

6.2. Case study 2: Structural beam

The performance metrics for this case study are reported in Table 7. *MSE* values are observed to increase as more complex datasets are used (more damaged elements are to be identified), regardless of the routing algorithm. For the dynamic routing, the minimum *MSE* reaches a value of 0.038% and the maximum is 0.783%. In the case of EM routing, the minimum and maximum are 0.132% and 0.948%, respectively. This behavior can be explained by noticing that models trained with more diverse datasets (DS2 and DS3 are more diverse than DS1, as they include more types of images) are fed with images that will probably have a greater amount of total (or accumulated) damage in the structure, because there are images with more damaged elements. Thus, the task of recognizing the proper amount of damage and assigning it to the correct beam elements is harder when there are up to two (e.g., *B_EMR_DS2* and *B_DR_DS2* models) or even three elements (e.g., *B_EMR_DS3* and *B_DR_DS3* models) that might be damaged, as opposed to models with up to one damaged element (as is the case of *B_EMR_DS1* and *B_DR_DS1*). This also explains the increase in *DME*, since the more the total damage increases, the greater is the chance of misallocating it through the beam. As with the first case study, *FAE* values decrease because as the detection complexity increases by having images with more damaged elements, there are less possibilities for false alarms.

In Fig. 15 to Fig. 17, *FAE* and *DME* are presented for the six models according to the damage percentage. This is important considering that it is not the same to have, for example, a false negative in an element with a damage of 70% than a 3% one, for the first case would mean the model failed to identify a 70% damaged element, which is far more serious in terms of

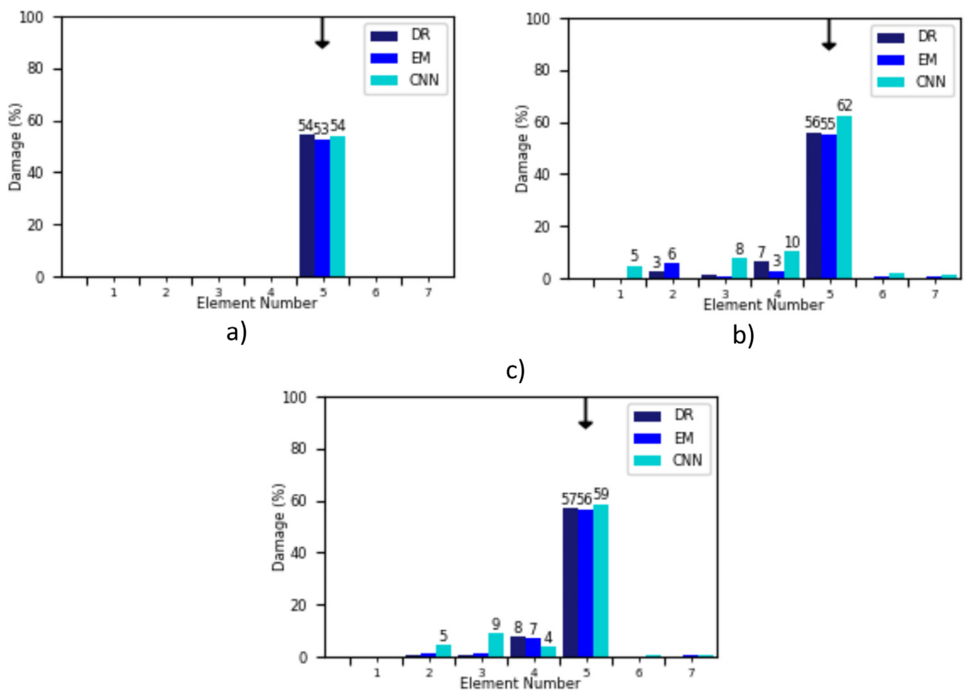

Fig. 14. Comparison between models performance in: (a) dataset 1 (i.e., models *SM_DR_DS1* and *SM_EMR_DS1* compared with CNN), (b) dataset 2 (i.e., models *SM_DR_DS2* and *SM_EMR_DS2* compared with CNN), and (c) dataset 3 (i.e., models *SM_DR_DS3* and *SM_EMR_DS3* compared with CNN).

Table 7
Training performance, second case study.

Model	*MSE* [%]	*DME* [%]	*FAE* [%]	Time per epoch [s]
B_DR_DS1	0.038	1.17	65.30	22.3
B_DR_DS2	0.415	5.33	55.28	30.1
B_DR_DS3	0.783	6.17	44.70	132.2
B_EMR_DS1	0.132	1.59	77.43	153.8
B_EMR_DS2	0.464	4.36	52.08	260.1
B_EMR_DS3	0.948	6.14	45.24	375.9

safety than the second case. The results show that, for models trained in dataset 1 (i.e., for detecting one damaged element), false alarms and false negatives only accumulate in the range of 0%–10% damage. For models trained in datasets 2 and 3 (i.e., for detecting two and three damaged elements, respectively), those values accumulate in the ranges of 0%–10%, 10%–20%, and 20%–30%. These results suggest that any element with more than a 30% damage is properly identified (i.e., it is not a false positive), and that any element with more than a 30% damage will certainly be identified as such (the proper amount of damage depends on the *MSE* metric). Also note that false negatives are far more scarce than false positives, without the need of penalizing a false negative more than a false positive in the loss function. This indicates that the proposed CapsNets models are conservative, which is preferable from a structural safety and reliability perspective.

Comparing both routing algorithms, and based on the results in Table 7 and Fig. 15 to Fig. 17, it can be observed that dynamic routing-based models outperform the models using the EM routing when trained to detect one damaged element (trained on dataset 1). However, that is not so clear for the models developed to detect two and three damaged elements (trained in datasets 2 and 3, respectively). When identifying two damaged elements, dynamic routing is superior at quantifying damage (i.e., better *MSE*), but less accurate when localizing damage (i.e., lower *DME* and *FAE*) than EM routing. When detecting and quantifying three damaged elements, both routing algorithms present virtually the same performance. This can be explained by the types of models used for each routing algorithm. The models used along with the dynamic routing algorithm have 2, 3 or 4 secondary capsules, depending on whether the model is required to identify up to three damaged elements (and, thus, four health states). Model *B_DR_DS1*, in turn, has two secondary capsules as there are only two health states: undamaged beam or one damaged element. That is, the health state diagnostics (i.e., classification task) is easier than in the other models because there are only two options: from a routing point of view, results show that the two clusters formed by the dynamic routing algorithm are very separated, unlike those in the other models. The algorithm can isolate undamaged beams with relative ease. However, segregation between multiple damaged elements results harder for the other models. Moreover, in the case of CapsNets-based models using EM routing, there are 18 secondary capsules, one for each element in the beam. The activation of each capsule indicates the probability of existence of a damaged element in the beam. This means that the secondary capsules layer comprises the task of locating damage and not only identifying the number of damaged elements, as is the case of the capsules in the dynamic routing. As seen in Table 5, all EM routing models are the same. This also explains why the results do not vary as much as in the case of models using dynamic routing.

All the proposed CapsNets models in Table 7, with both dynamic routing and EM routing, are tested with three unseen damage scenarios and compared to CNN-based models in detecting damage under the three experimental scenarios shown in Table 1 (see Fig. 18 to Fig. 26). Since damage is represented as a stiffness reduction, there is no precise way of knowing the amount of damage a cut inflicts on the beam. However, we know a deep cut inflicts more damage than a shallow one. This gives a rough estimate of what the real damages should be.

First, let us consider the models developed and trained to detect and to quantify up to one damaged element in the beam (i.e., models trained in dataset 1). Based on Fig. 18 to Fig. 20, the best results are achieved in the experimental damage scenario 1 (i.e., one damaged element in the beam). Moreover, results for experimental cases 2 and 3 (Fig. 19 and Fig. 20, respectively) show

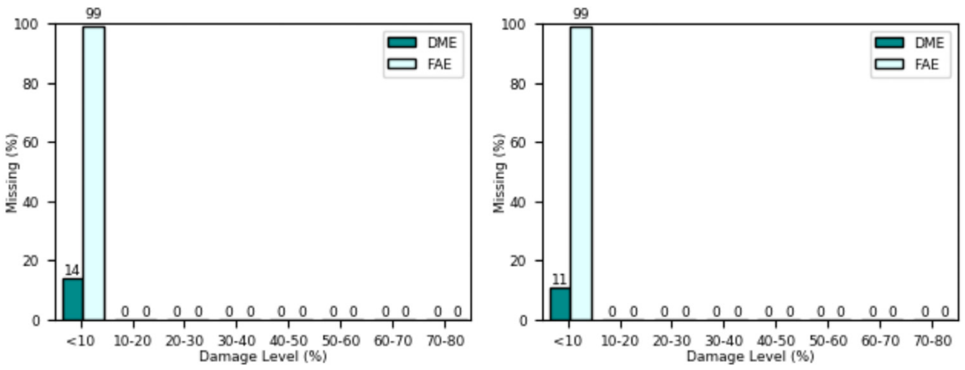

Fig. 15. DME and *FAE* versus damage level for models *B_DR_DS1* (left) and *B_EM_DS1* (right).

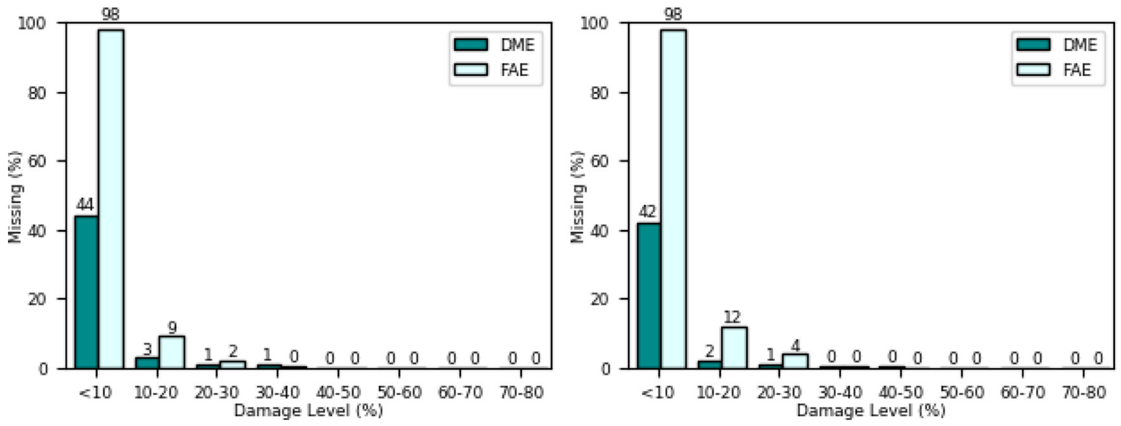

Fig. 16. DME and *FAE* versus damage level for models *B_DR_DS2* (left) and *B_EM_DS2* (right).

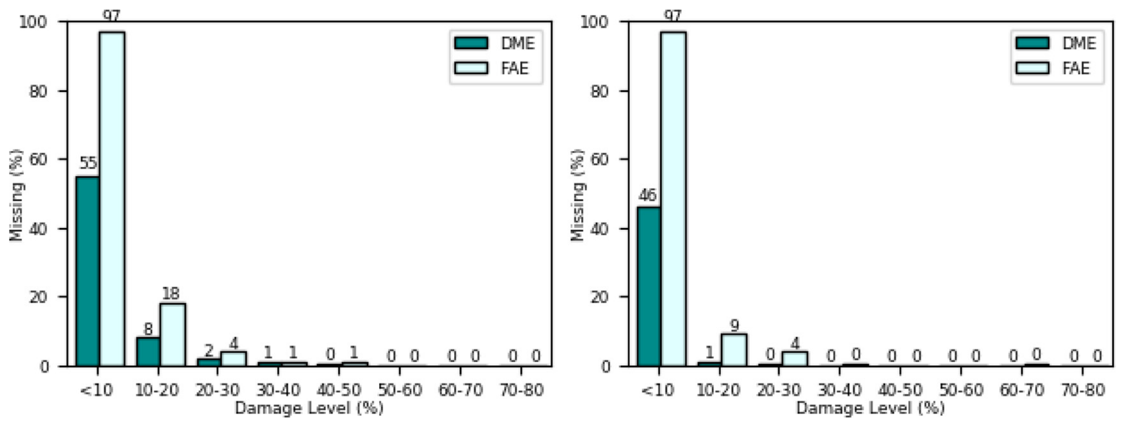

Fig. 17. DME and *FAE* versus damage level for models *B_DR_DS3* (left) and *B_EM_DS3* (right).

that the models do not present a good generalization capability, mainly because they were not trained with these types of scenarios. In the case of dynamic routing, *B_DR_DS1* model is intrinsically built to recognize only 0 or 1 damaged element due to its secondary capsules. This affects its generalization capabilities. This behavior is replicated in models trained in datasets 2 and 3. Amongst Figs. 21–23, Fig. 22 presents the most precise outcome of all the scenarios, being the one with the smallest number of false positives, and concurrently less important ones, in terms of damage quantity. This is also shown in Figs. 24–26, where the latter shows the most precise results.

In terms of false negatives, the results from the testing in the damage scenarios in Table 1 are somewhat inferior (in terms of damage size) to the ones obtained in the training process. This discrepancy is far more substantial in those results from EM routing than those from dynamic routing. Also, Fig. 21 shows false positives with damage size over 30%, which are higher than the false positives reported in Fig. 16. This is because training and validation sets are built using FE model, whereas the test set is built on experimental cases on a real beam. The results show that the algorithms do well at learning the equation behind the FE model. However, this leads to a lack of generalization when dealing with experimental cases, whereby the physical phenomenon behind them would have been represented by a much more complex equation. This is dealt with below.

Now, comparing the proposed CapsNets models with CNN, Fig. 18 to Fig. 26 show that the CapsNets models with dynamic routing present better results for all the experimental damage

Table 8
Comparison between models with and without dropout, training phase.

Model	*MSE* [%]	*DME* [%]	*FAE* [%]	Time per epoch [s]
B_DR_DS1	0.038	1.17	65.30	22.3
B_DR_DS1_D	0.138	3.79	51.67	22.5
B_DR_DS2	0.415	5.33	55.20	30.1
B_DR_DS2_D	0.610	9.96	47.96	48.11
B_DR_DS3	0.783	6.17	44.70	132.2
B_DR_DS3_D	0.949	10.53	46.99	137.4

scenarios in Table 1 than both the CNN and the CapsNets models with EM routing. In fact, the most accurate cases are shown in Figs. 18, 22 and 26, whereby CapsNets models with dynamic routing correctly quantify the damaged elements and exhibit the smallest number of false positives. Also note that most of these values are located next to the actual damaged elements. This is related to the fact that, unlike the spring–mass system, a beam is a continuous structure where elements are not clearly delimited, and divisions are set arbitrarily.

Moreover, the CapsNets models with EM routing deliver a higher rate of false positive, as shown, for example, in Fig. 21. For diagnostics purposes, these false positives can generate confusion in the analysis as they might act as distractors and thus make the model unreliable from a practical perspective. This occurs because the CapsNets models based on EM routing are more complex in terms of parameters to be learned than the ones using dynamic routing, thus affecting their generalization capability. Indeed, CapsNets models with EM use 18 secondary capsules, one for each element in the beam. Even though the same type of models for the EM-based models are used for the spring–mass system, overfit is not observed and a good generalization is therefore obtained. This behavior can be attributed to each element of the spring–mass system being independent, and the system itself being a discrete one, unlike the beam which is continuous, and also because CapsNets models with EM routing for the spring–mass system only have 7 capsules instead of 18 (for the beam example), thus resulting in a simpler model, fewer parameters, which reduces the risk of overfitting given the same training dataset.

To detect overfitting, a data subset is used only to estimate the loss function value. This subset is not used for training purposes. This validation loss is compared to the training set loss to detect overfitting. In this case, no signs of overfitting were detected. There is no overfitting between training and validation sets, but there are clear signs of overfitting between the training set and the experimental cases. This occurs because, unlike the experimental cases, training and validation sets are built from a FE model and are presented as TF images. Thus, overfitting signs uncover the capacity of the algorithms to recognize the mathematical model behind the generation of images. They focus on describing the underlying equations. These equations achieve only a simplified representation of the real phenomenon, which is why experimental cases show poorer results than expected.

6.3. Case study 2: Structural beam — application of dropout

For overcoming overfitting and thus achieve better generalization between the training sets (which are generated through a FE model) and the experimental cases, dropout [51] with $p = 0.5$ (this value is defined via sensitivity analysis) is added to the first hidden layer for dynamic routing models only. This means that, with each iteration, each neuron in the layer will have 50% chance of being turned off and not considered in the learning process. This lowers its training results, but achieves better generalization by avoiding overfitting. Training results are shown in Table 8:

Each model is tested using the experimental cases and compared with the best result previously achieved, which is dynamic routing without dropout. The results are shown in the following figures:

As shown in Table 8, by using dropout, the training performance metrics (*MSE*, *DME* and *FAE*) decrease. However, greater generalization capability is achieved. It is clearly noted that training results lower their quality, particularly with *MSE* and *DME*. However, these results are very similar to what it is shown in the experimental results in Fig. 27 to Fig. 29. Model *B_DR_DS1* correctly locates damage for images with one damaged element and shows no false alarms, which is very important because it shows the model can isolate damaged elements in a very reliable way. For cases 2 and 3, the model cannot recognize the element with the least amount of damage. However, Fig. 27 shows that model *B_DR_DS1_D* can assign a greater amount of damage to the image than without dropout, and even though it presents a larger false positive in the 7th element, it corresponds to the fact that the 8th element is greatly damaged (cut depth: 13 [cm]).

For dataset 2, experimental results also improve with the use of dropout. The models still perform well in cases 1 and 2, showing only a small percentage of false positives. In case 2, model *B_DR_DS2_D* is capable of recognizing the element with the small amount of damage as well as the one with a great amount, showing only a 6% of false positive value located in the 18th element, next to the truly damaged element. In case 3, despite *B_DR_DS2_D* being trained with up to 2 damaged elements images, the algorithm correctly detects the three damaged elements, with the drawback of showing an important number of false positives. This shows that the algorithms achieve a better capacity for generalization when using dropout, and from two points of view: the first one refers to the capacity of the model to learn generalizable features from the FEM images to the experimental cases. The second one (which is more complex and difficult to achieve) refers to the capacity of the models to locate and to quantify more damaged elements per beam than those from which the algorithm learned the task (Fig. 28).

For dataset 3, the algorithm correctly assesses damage better than without using dropout. Particularly, Fig. 29 shows that the use of dropout directly affects the quantity of false positive values. Fig. 29 (upper right) shows that the use of dropout takes away the false positive from element 11 but misallocates the less damaged element. Instead of allocating it to the 8th element, it is allocated to the 7th. For damage assessment purposes, however, this is not a highly relevant mistake because the element is displaced by only one element, but its quantity is correctly assigned. Finally, Fig. 29 shows that the model recognizes that there are 3 damaged elements, it locates them correctly, and assigns the correct amount of damage to each damaged element. Also, the most relevant false positive value has a damage of 2% and is located next to the element with the highest damage percentage. Comparing *B_DR_DS3_D* with *B_DR_DS3*, there is a significant difference in terms of false positives. A 25% false positive, located at element 7, and a 16% one located at element 11 act as great distractors in terms of assessment, which are not present when using dropout.

6.4. Case study 2: Structural beam — comparison with other methods

The results for the second case study using DS3 (up to 3 damaged elements) shown above are compared to other methods in the literature, namely OS-ELM, MLP and CNN. Also, we test our approach using Random Forest to compare our results with a machine learning algorithm outside neural networks. For the methods used in [16] and [27], the same two case studies from

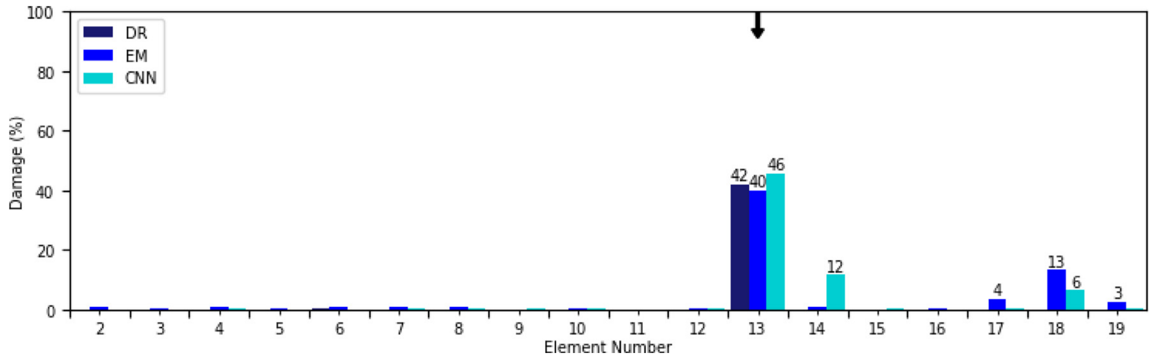

Fig. 18. Comparison between the models performance in damage scenario 1, dataset 1 (i.e. models *SM_DR_DS1* and *SM_EMR_DS1* compared with CNN).

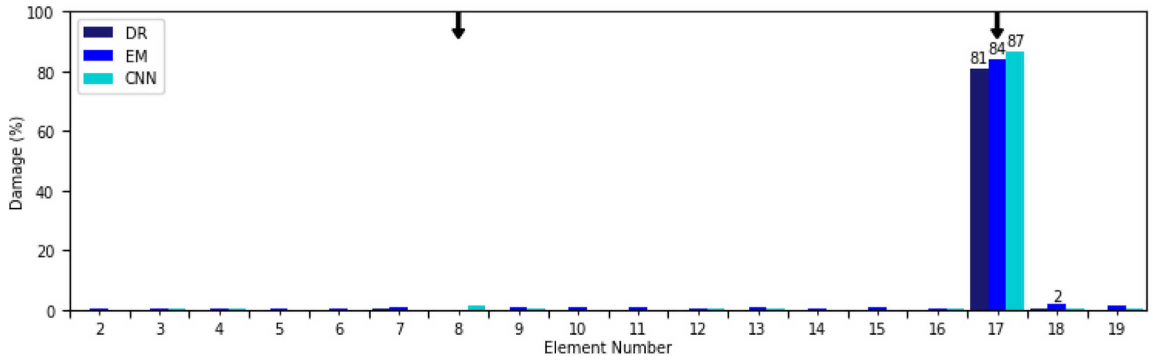

Fig. 19. Comparison between the models performance in damage scenario 2, dataset 1.

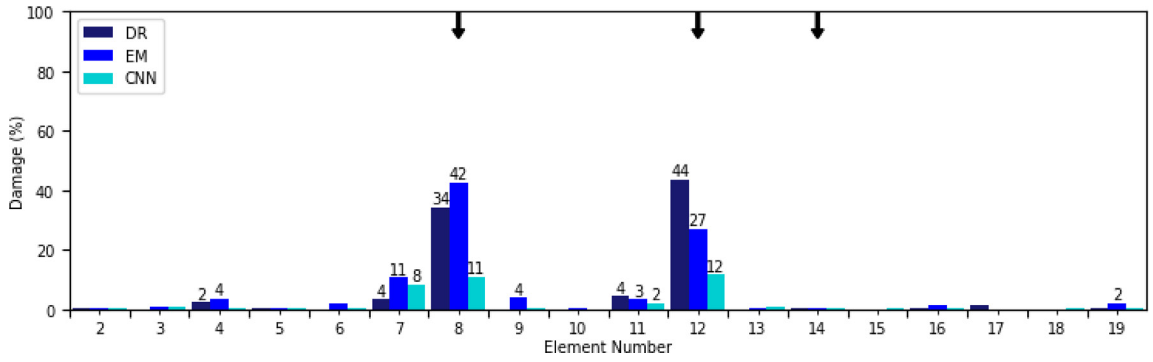

Fig. 20. Comparison between the models performance in damage scenario 3, dataset 1.

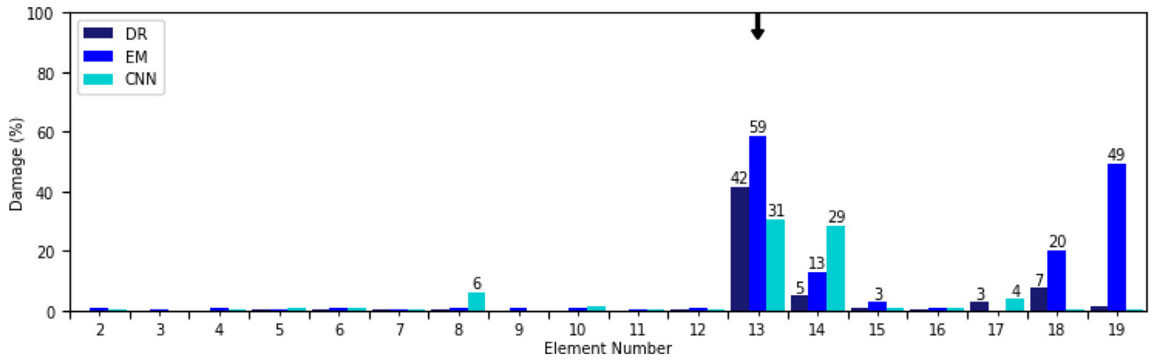

Fig. 21. Comparison between the models performance in damage scenario 1, dataset 2 (i.e. models *SM_DR_DS2* and *SM_EMR_DS2* compared with CNN).

this paper are analyzed. However, the experimental scenarios within the case studies are not the same, thus, we cannot perform a fair comparison of the test results. Therefore, we compare the training performance in terms of DME and FAE measures. The structures of the different models (all of them properly tuned to achieve the best results) used for comparison are:

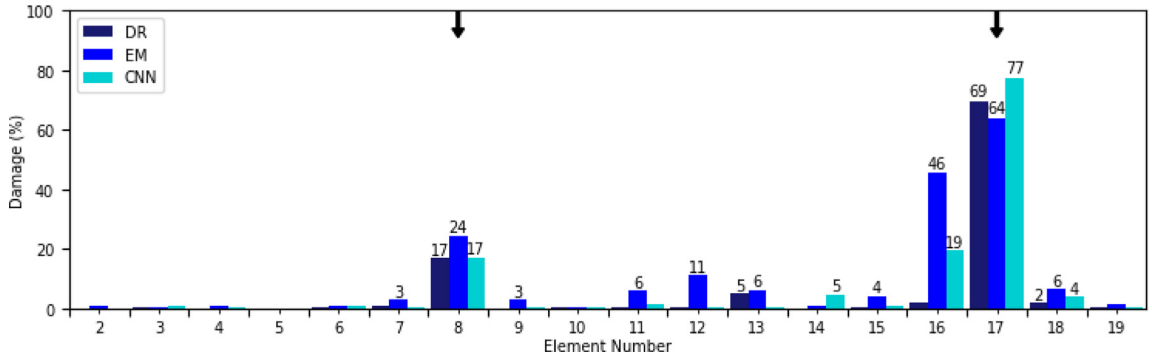

Fig. 22. Comparison between the models performance in damage scenario 2, dataset 2.

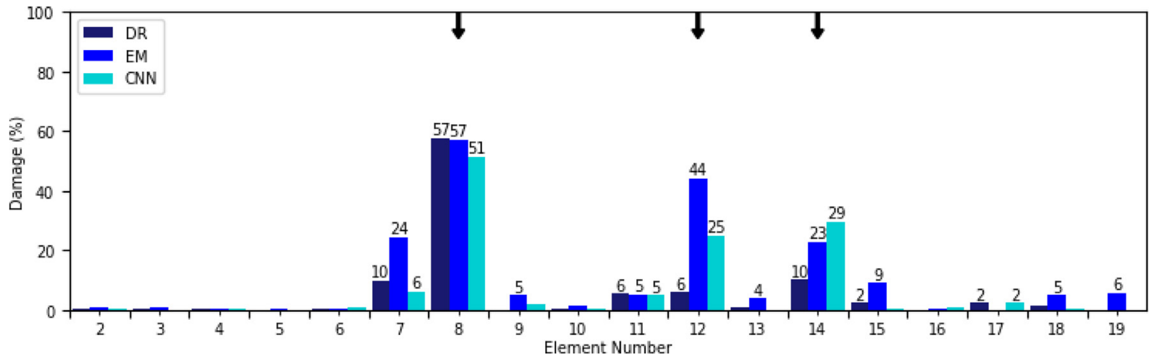

Fig. 23. Comparison between the models performance in damage scenario 3, dataset 2.

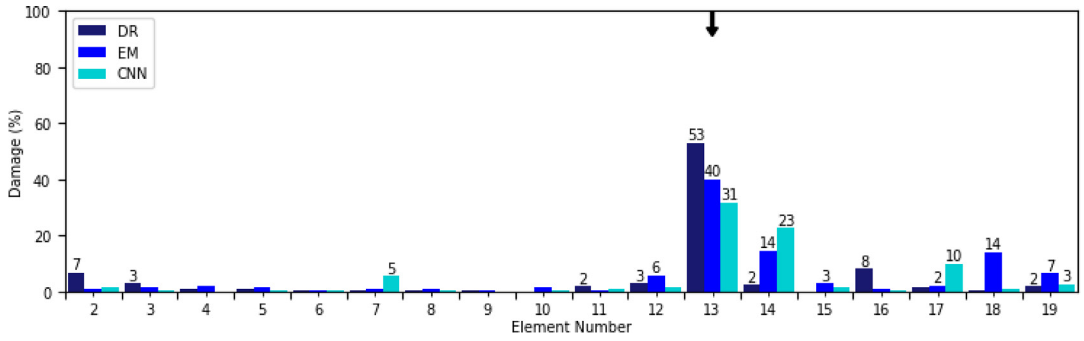

Fig. 24. Comparison between the models performance in damage scenario 1, dataset 3 (i.e. models *SM_DR_DS3* and *SM_EMR_DS3* compared with CNN).

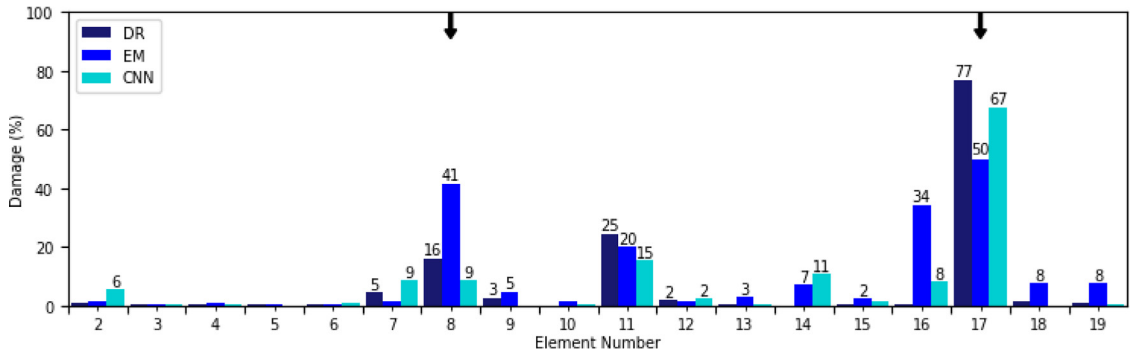

Fig. 25. Comparison between the models performance in damage scenario 2, dataset 3.

- OS-ELM: This model uses antiresonant frequencies as inputs. As authors state in [16], the best performance of the model is obtained using a network with a single layer of 165 nodes with a sigmoid activation function.

- MLP: One hidden layer of 1024 units and ReLU activation function. The output layer has the same number of units as the number of elements in the structure. The model is

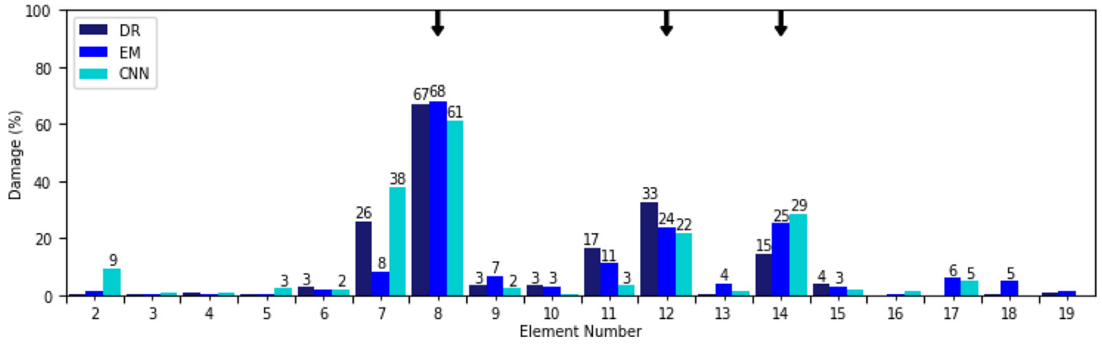

Fig. 26. Comparison between the models performance in damage scenario 3, dataset 3.

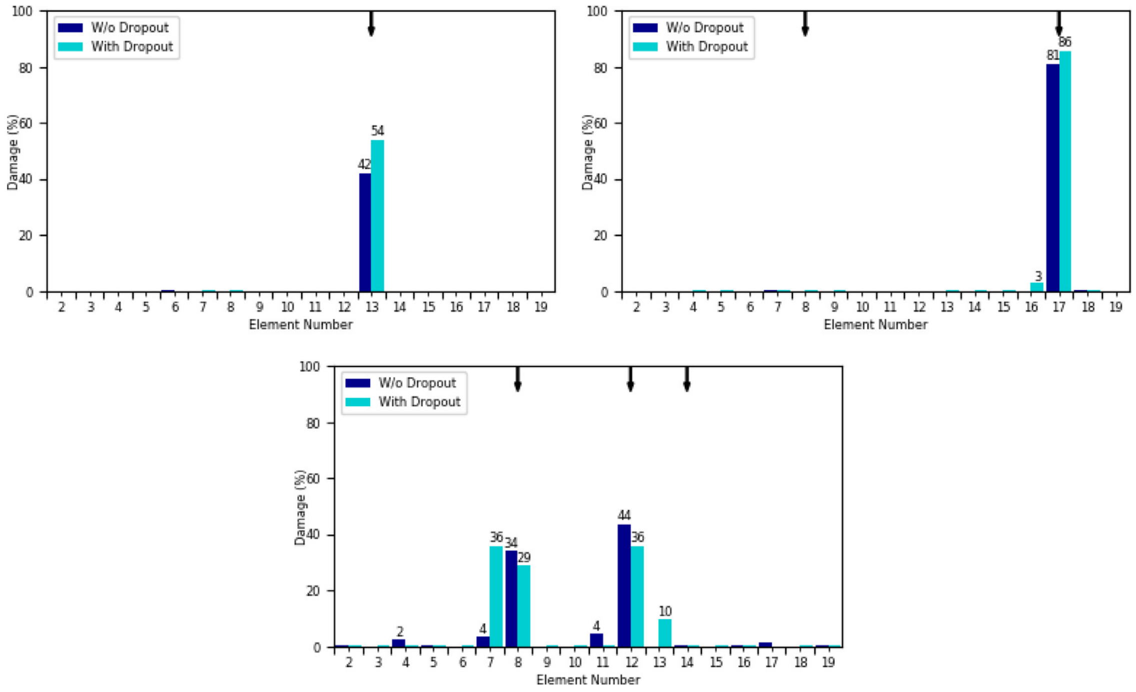

Fig. 27. Comparison between *B_DR_DS1* and *B_DR_DS1_D* performance in all three experimental cases.

optimized via Adam optimization algorithm and dropout is used with 0.5 keep probability.

- CNN: Two convolution layers that generate feature maps of $1 \times 96 \times 32$ and $1 \times 91 \times 94$, respectively, followed by a fully connected layer of 1024 units. All layers have a ReLU activation function, and dropout is used with 0.5 keep probability. Adam optimizer is used.
- Random Forest: 300 estimators and sampling with replacement are used when training.

As in Figs. 15–17, the results are divided according to the real damage values to see how the algorithms perform at different damage levels. The results are displayed in Tables 9 and 10. It can be seen that the MLP model is the one that has the lowest performance, showing the highest percentages at all levels. The OS-ELM algorithm shows high levels of DME up to 20% damages. This means that damaged elements with 20% damage or less have a high probability of being undetected. Observing the false alarms (FAE), the same issue is present. Up to 30% damage levels, there are many false alarms compared to the rest of the models. The Random Forest model shows the best results in both tables, showing only an 8% of false negatives (DME) at the <10% damage level. False alarm values are also the lowest ones. This shows how the model can identify and measure damage accurately when trained with FE model images. Nonetheless, performance drops when it is tested in experimental scenarios. Fig. 30 shows how the model assesses the damage scenarios with one, two, and three damaged elements, respectively. Fig. 30(a) shows correct damage identification, but one false alarm with 14% damage. In Fig. 30(b), the element with a shallow cut (see Table 1) is not recognized by the Random Forest model. Even though the one with the deepest cut is recognized, the model predicts less damage than in the CapsNets models, and has a 30% damage false positive next to it. For the case with three damaged elements, the Random Forest model recognizes two of them, also having considerable false alarms distributed through the beam. These results show how, despite the Random Forest presenting better performance metrics in the training stage, the evaluation in experimental scenarios demonstrates it has poor generalization in real cases.

Regarding computation times, one must evaluate the time a trained model takes to assess a new datapoint. This is due to the fact that a deployed model is constantly assessing new scenarios, whereas training occurs only once. The time the model takes for doing this determines whether it is capable of doing online monitoring or not. In relation to this,

Table 11 presents a comparison of all the models mentioned in this section when assessing a new datapoint, with exception

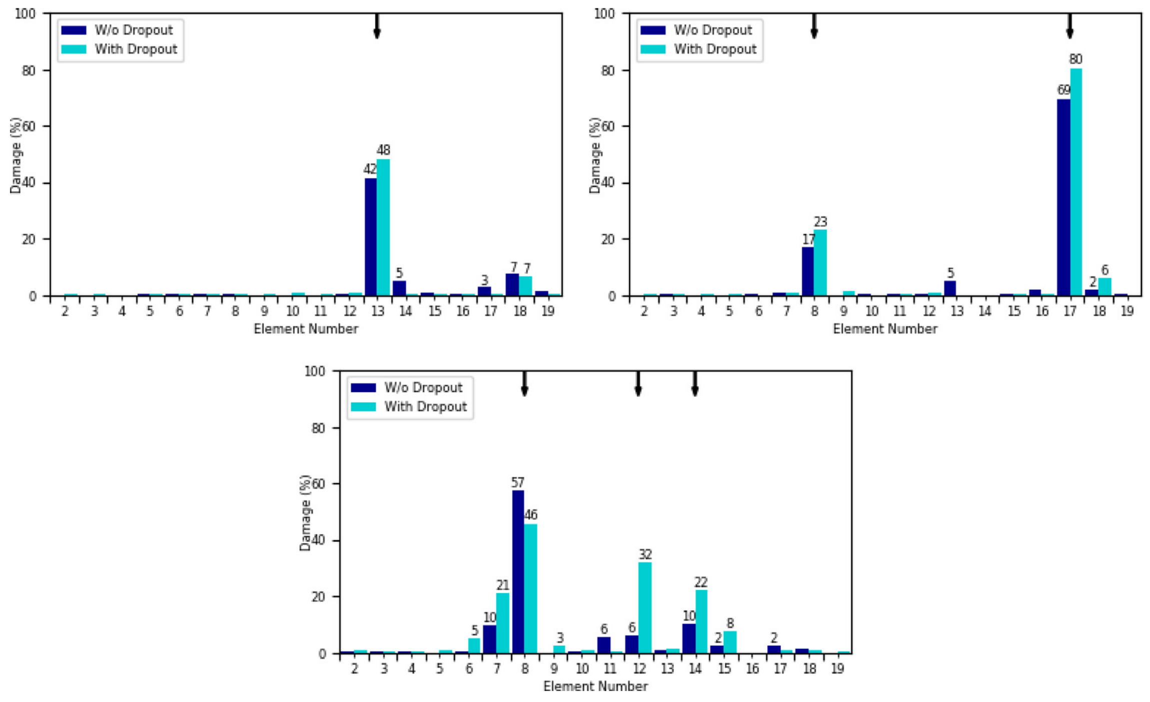

Fig. 28. Comparison between *B_DR_DS2* and *B_DR_DS2_D* performance in all three experimental cases.

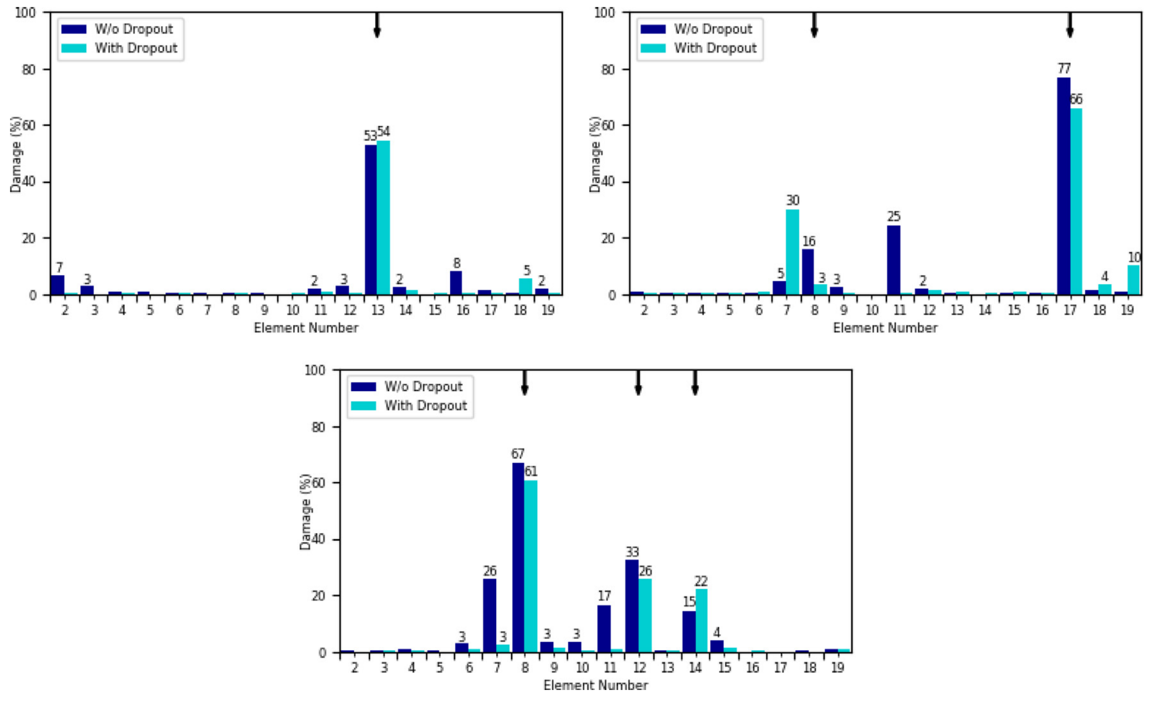

Fig. 29. Comparison between *B_DR_DS3* and *B_DR_DS3_D* performance in all three experimental cases.

of MLP. However, knowing that MLPs are less complex algorithms than CNNs, we can assume its computation time might be lower than 0.31 [s]. For the rest of the algorithms, there is a clear difference between the Deep Learning models (CNN and CapsNets-based models) and the rest (OS-ELM and RF), being CapsNets with EM routing the model that takes the most time, and Random Forest the one that takes the less time, being the latter 3,900 times faster than the former, and 1,650 times faster than CapsNets with dynamic routing and dropout. However, this does not discard them as models used for online monitoring, because they all take less than a second to assess a new image [5].

7. Conclusions

In the present work, capsule neural networks were developed for locating and quantifying damage in structural elements. Two types of routing algorithms within capsule networks were studied: dynamic routing and EM routing. Both general models were analyzed with two case studies: a spring–mass system and an experimental beam. In both cases, various models were developed and trained using images containing 10 transmissibility functions, each image representing various damage scenarios. Images were created using a FE model tuned via experimental data. The trained

Table 9
Comparison of DME performance between CapsNets models and the existing literature.

DME [%]									
Reference	Algorithm	Damage Level [%]							
		<10	10-20	20-30	30-40	40-50	50-60	60-70	70-80
[27]	**MLP**	82	72	46	20	9	4	2	1
[16]	**OS-ELM**	78	42	8	2	0	0	0	0
[27]	**CNN**	56	8	1	0	0	0	0	0
–	**Random Forest**	8	0	0	0	0	0	0	0
–	**CapsNets EM**	46	1	0	0	0	0	0	0
–	**CapsNets DR**	55	8	2	1	0	0	0	0
–	**CapsNets DR with Dropout**	75	23	8	2	1	0	0	0

Table 10
Comparison of FAE performance between CapsNets models and the existing literature.

FAE [%]									
Reference	Algorithm	Damage Level [%]							
		<10	10-20	20-30	30-40	40-50	50-60	60-70	70-80
[27]	**MLP**	96	74	44	18	5	1	0	0
[16]	**OS-ELM**	99	86	28	5	2	0	0	0
[27]	**CNN**	98	37	8	2	0	0	0	0
–	**Random Forest**	97	8	0	0	0	0	0	0
–	**CapsNets EM**	97	9	4	0	0	0	0	0
–	**CapsNets DR**	97	18	4	1	1	0	0	0
–	**CapsNets DR with Dropout**	97	18	5	0	0	0	0	0

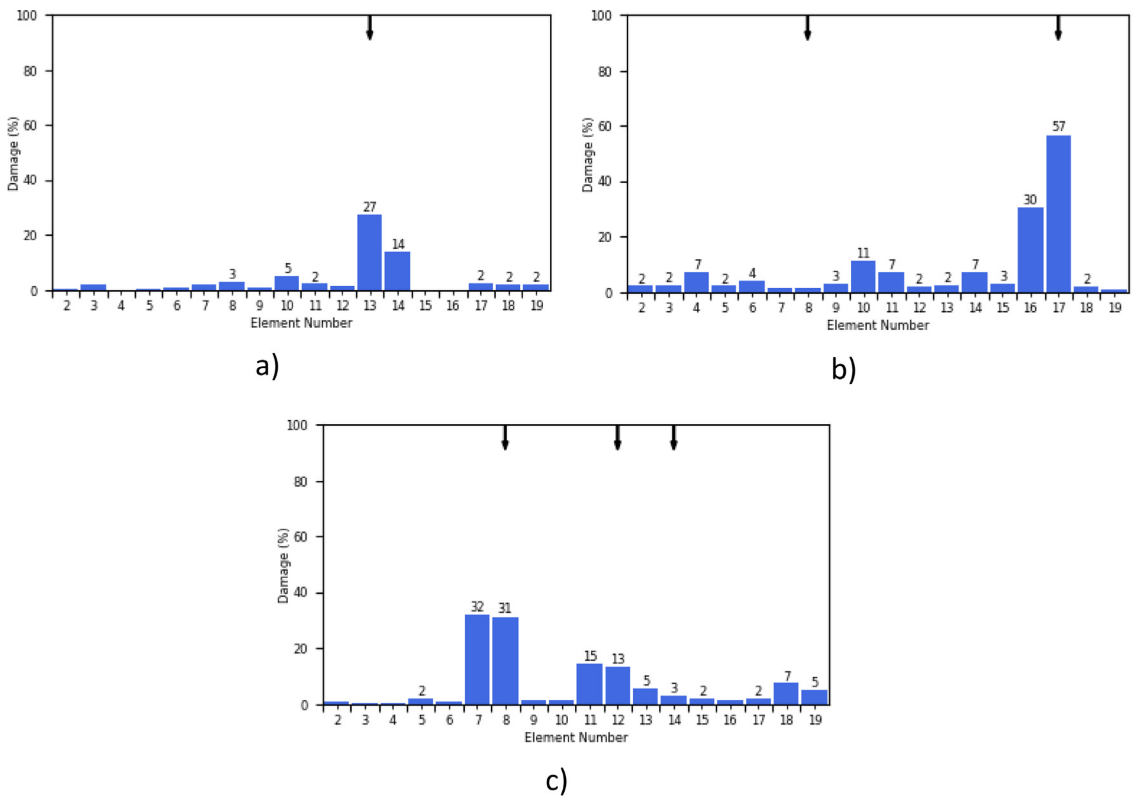

Fig. 30. Random Forest model evaluation in experimental scenarios.

models were validated using experimental cases and compared with CNN, Random Forest, and models taken from the literature.

Even though the results show capsule networks outperform CNN, there were important differences between the two case studies. In the spring–mass system case study, the results from the FE model-based data matched the experimental results, there are no false negative values and few false positives. In the second case study involving the structural beam, the results based on the FE model-based data differ from the experimental results, leading to the conclusion that some overfitting is occurring. For this reason, dropout was applied to the model with dynamic routing, slightly decreasing the validation performance metrics, but obtaining better generalization capacity. Moreover, these results from models incorporating dropout layers outperform CNNs, notably reducing false positive values, while maintaining a correct damage estimation at the correct locations.

From a practical perspective, model *B_DR_DS3_D* can be considered the most suitable model among the proposed ones. It can adequately assess cases with one, two and three damaged

Table 11
Assessment time for new datapoints.

Algorithm	Computation time for new datapoint [s]
MLP	No information available
OS-ELM	0.005
CNN	0.31
Random Forest	0.0002
CapsNets EM	0.78
CapsNets DR	0.33
CapsNets DR with Dropout	0.33

elements. Despite the training results for this model not being the best ones, tests on experimental data show that model *B_DR_DS3_D* has a stable performance across various numbers of damaged elements. When compared to other models (as shown in Tables 9 and 10) *B_DR_DS3_D* stands out for showing acceptable training results and the best performance in real scenarios. This result closes the gap between research and real-life applications, because it shows how an algorithm trained on simulated data can achieve good levels of generalization and yield acceptable results in real life cases. These results show that CapsNets are a promising approach for damage identification and localization in structures and further research should be carried out. For example, the analysis of more complex structures should produce interesting insights on how CapsNets locate and quantify damage in more complex scenarios.

Although the results are promising, the proposed capsule neural networks have some limitations. Due to the use of routing algorithms, CapsNets are computationally expensive to train, particularly when compared to convolutional neural networks. To circumvent this, the models considered herein were constructed with only one routing-between-capsules layer. This might be a limitation for other applications. Thus, deeper models should be analyzed as well as ways to accelerate the training process. Moreover, as is the case with neural networks in general, CapsNets provide a single point estimate as results, meaning there is no uncertainty quantification and propagation. Therefore, one could further extend the proposed models by developing Bayesian capsule neural networks able to explicitly quantify uncertainty (both model and parameter uncertainty) [52,53].

CRediT authorship contribution statement

Joaquín Figueroa Barraza: Conceptualization, Methodology, Validation, Formal analysis, Investigation, Software. **Enrique Lopez Droguett:** Conceptualization, Methodology, Validation, Formal analysis, Investigation. **Viviana Meruane Naranjo:** Conceptualization, Methodology, Validation, Investigation. **Marcelo Ramos Martins:** Conceptualization, Methodology.

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.

Acknowledgment

The authors acknowledge the partial financial support by the Chilean National Fund for Scientific and Technological Development (Fondecyt) under Grant No. 1190720.

References

[1] C.R. Farrar, K. Worden, An introduction to structural health monitoring, Philos. Trans. R. Soc. A Math. Phys. Eng. Sci. 365 (1851) (2007) 303–315, http://dx.doi.org/10.1098/rsta.2006.1928.
[2] A.C. Tokognon, B. Gao, G.Y. Tian, Y. Yan, Structural health monitoring framework based on internet of things: A survey, IEEE Internet Things J. 4 (3) (2017) 629–635, http://dx.doi.org/10.1109/JIOT.2017.2664072.
[3] X. Kong, C.S. Cai, J. Hu, The state-of-the-art on framework of vibration-based structural damage identification for decision making, Appl. Sci. 7 (5) (2017) http://dx.doi.org/10.3390/app7050497.
[4] K. Worden, Structural fault detection using a novelty meassure, J. Sound Vib. 201 (1) (1997) 85–101.
[5] V. Meruane, A. Ortiz-Bernardin, Structural damage assessment using linear approximation with maximum entropy and transmissibility data, Mech. Syst. Signal Process. 54 (2015) 210–223, http://dx.doi.org/10.1016/j.ymssp.2014.08.018.
[6] N.M.M. Maia, R.A.B. Almeida, A.P.V. Urgueira, R.P.C. Sampaio, Damage detection and quantification using transmissibility, Mech. Syst. Signal Process. 25 (7) (2011) 2475–2483, http://dx.doi.org/10.1016/j.ymssp.2011.04.002.
[7] G. Manson, K. Worden, D. Allman, Experimental validation of a structural health monitoring methodology. Part II. Novelty detection on a Gnat aircraft, J. Sound Vib. 259 (2) (2003) 345–363, http://dx.doi.org/10.1006/jsvi.2002.5167.
[8] K. Worden, G. Manson, D. Allman, Experimental validation of a structural health monitoring methodology: Part I. Novelty detection on a laboratory structure, J. Sound Vib. 259 (2) (2003) 323–343, http://dx.doi.org/10.1006/jsvi.2002.5168.
[9] G. Manson, K. Worden, D. Allman, Experimental validation of a structural health monitoring methodology: Part III. Damage location on an aircraft wing, J. Sound Vib. 259 (2) (2003) 365–385, http://dx.doi.org/10.1006/jsvi.2002.5169.
[10] R. Perera, A. Ruiz, A multistage FE updating procedure for damage identification in large-scale structures based on multiobjective evolutionary optimization, Mech. Syst. Signal Process. 22 (4) (2008) 970–991, http://dx.doi.org/10.1016/J.YMSSP.2007.10.004.
[11] Y.-J. Cha, O. Buyukozturk, Structural damage detection using modal strain energy and hybrid multiobjective optimization, Comput. Civ. Infrastruct. Eng. 30 (5) (2015) 347–358, http://dx.doi.org/10.1111/mice.12122.
[12] S.M. Seyedpoor, A two stage method for structural damage detection using a modal strain energy based index and particle swarm optimization, Int. J. Non. Linear. Mech. 47 (1) (2012) 1–8, http://dx.doi.org/10.1016/J.IJNONLINMEC.2011.07.011.
[13] N. Khaji, M. Mehrjoo, Crack detection in a beam with an arbitrary number of transverse cracks using genetic algorithms, J. Mech. Sci. Technol. 28 (3) (2014) 823–836, http://dx.doi.org/10.1007/s12206-013-1147-y.
[14] V. Meruane, Model updating using antiresonant frequencies identified from transmissibility functions, J. Sound Vib. 332 (4) (2013) 807–820, http://dx.doi.org/10.1016/j.jsv.2012.10.021.
[15] Szewczyk, Z. Peter, P. Hajela, Damage detection in structures based on feature-sensitive neural networks, J. Comput. Civ. Eng. 8 (2) (1994) 163–178, http://dx.doi.org/10.1061/(ASCE)0887-3801(1994)8:2(163).
[16] V. Meruane, Online sequential extreme learning machine for vibration-based damage assessment using transmissibility data, J. Comput. Civ. Eng. 30 (3) (2016) 04015042, http://dx.doi.org/10.1061/(ASCE)CP.1943-5487.0000517.
[17] C.S.N. Pathirage, J. Li, L. Li, H. Hao, W. Liu, P. Ni, Structural damage identification based on autoencoder neural networks and deep learning, Eng. Struct. 172 (May) (2018) 13–28, http://dx.doi.org/10.1016/j.engstruct.2018.05.109.
[18] M. Silva, A. Santos, R. Santos, E. Figueiredo, C. Sales, J.C.W.A. Costa, Deep principal component analysis: An enhanced approach for structural damage identification, Struct. Heal. Monit. (2018) http://dx.doi.org/10.1177/1475921718799070.
[19] Y. Bao, Z. Tang, H. Li, Y. Zhang, Computer vision and deep learning–based data anomaly detection method for structural health monitoring, Struct. Heal. Monit. 18 (2) (2019) 401–421, http://dx.doi.org/10.1177/1475921718757405.
[20] C.S.N. Pathirage, J. Li, L. Li, H. Hao, W. Liu, R. Wang, Development and application of a deep learning–based sparse autoencoder framework for structural damage identification, Struct. Heal. Monit. 18 (1) (2019) 103–122, http://dx.doi.org/10.1177/1475921718800363.
[21] Y. Yu, C. Wang, X. Gu, J. Li, A novel deep learning-based method for damage identification of smart building structures, Struct. Heal. Monit. 18 (1) (2019) 143–163, http://dx.doi.org/10.1177/1475921718804132.
[22] C. Modarres, N. Astorga, E.L. Droguett, V. Meruane, Convolutional neural networks for automated damage recognition and damage type identification, Struct. Control Heal. Monit. 25 (10) (2018) 1–17, http://dx.doi.org/10.1002/stc.2230.

[23] Y.-J. Cha, W. Choi, O. Büyüköztürk, Deep learning-based crack damage detection using convolutional neural networks, Comput. Civ. Infrastruct. Eng. 32 (5) (2017) 361–378, http://dx.doi.org/10.1111/mice.12263.

[24] T. Guo, L. Wu, C. Wang, Z. Xu, Damage detection in a novel deep-learning framework: A robust method for feature extraction, Struct. Heal. Monit. 28 (2019) http://dx.doi.org/10.1177/1475921719846051.

[25] Y.Z. Lin, Z.H. Nie, H.W. Ma, Structural damage detection with automatic feature-extraction through deep learning, Comput. Civ. Infrastruct. Eng. 32 (12) (2017) 1025–1046, http://dx.doi.org/10.1111/mice.12313.

[26] Y. Gao, K.M. Mosalam, Deep transfer learning for image-based structural damage recognition, Comput. Civ. Infrastruct. Eng. 33 (9) (2018) 748–768, http://dx.doi.org/10.1111/mice.12363.

[27] S. Cofre-Martel, P. Kobrich, E. Lopez Droguett, V. Meruane, Deep convolutional neural network-based structural damage localization and quantification using transmissibility data, Shock. Vib. 2019 (2019) 1–27, http://dx.doi.org/10.1155/2019/9859281.

[28] G.E. Hinton, A. Krizhevsky, S.D. Wang, Transforming auto-encoders, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 6791 LNCS, (PART 1) Springer, Berlin, Heidelberg, 2011, pp. 44–51.

[29] S. Sabour, N. Frosst, G.E. Hinton, Dynamic routing between capsules, no. nips, 2017.

[30] Y. LeCun, C. Cortes, C.J.C. Burges, The MNIST dataset of handwritten digits, 1998, http://yann.lecun.com/exdb/mnist/.

[31] G. Hinton, S. Sabour, N. Frosst, M Atrix Capsules With Em Routing, 2018, pp. 1–15.

[32] P. Afshar, A. Mohammadi, K.N. Plataniotis, Brain tumor type classification via capsule networks, 2018.

[33] Y. Upadhyay, P. Schrater, Generative adversarial network architectures for image synthesis using capsule networks, 2018, pp. 1–9.

[34] R. LaLonde, U. Bagci, Capsules for object segmentation, 2018.

[35] M.T. Bahadori, Spectral capsule networks, ICLR Work. (2018) 1–5.

[36] A.E.W. Johnson, et al., MIMIC-III, a freely accessible critical care database, Sci. Data 3 (2016) 160035, http://dx.doi.org/10.1038/sdata.2016.35.

[37] A. Ruiz-Tagle Palazuelos, E.L. Droguett, R. Pascual, A novel deep capsule neural network for remaining useful life estimation, Proc. Inst. Mech. Eng. Part O J. Risk Reliab. (2019) http://dx.doi.org/10.1177/1748006x19866546, 1748006X1986654.

[38] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Adv. Neural Inf. Process. Syst. (2012) 1–9, http://dx.doi.org/10.1016/j.protcy.2014.09.007.

[39] S. Chesné, A. Deraemaeker, Damage localization using transmissibility functions: A critical review, Mech. Syst. Signal Process. 38 (2) (2013) 569–584, http://dx.doi.org/10.1016/j.ymssp.2013.01.020.

[40] Y.L. Zhou, N.M.M. Maia, M. Abdel Wahab, Damage detection using transmissibility compressed by principal component analysis enhanced with distance measure, JVC/Journal Vib. Control 24 (10) (2018) 2001–2019, http://dx.doi.org/10.1177/1077546316674544.

[41] M.I. Friswell, J.E.T. Penny, Crack modeling for structural health monitoring, Struct. Health Monit. 1 (2) (2016) 139–148.

[42] R.J. Douglas, K.A. Martin, A functional microcircuit for cat visual cortex, J. Physiol. 440 (1) (1991) 735–769, http://dx.doi.org/10.1113/jphysiol.1991.sp018733.

[43] J. Hawkins, S. Ahmad, Y. Cui, A theory of how columns in the neocortex enable learning the structure of the world, Front. Neural Circuits 11 (2017) 0–17, http://dx.doi.org/10.3389/fncir.2017.00081.

[44] V. Meruane, J. Mahu, Real-time structural damage assessment using artificial neural networks and antiresonant frequencies, Shock. Vib. 2014 (2014) 1–14, http://dx.doi.org/10.1155/2014/653279.

[45] S.E.-O. Bahlous, M. Abdelghani, H. Smaoui, S. El-Borgi, A modal filtering and statistical approach for damage detection and diagnosis in structures using ambient vibrations measurements, J. Vib. Control 13 (3) (2007) 281–308.

[46] Z.-B. Yang, Y.-N. Wang, H. Zuo, X.-W. Zhang, Y. Xie, X.-F. Chen, The Fourier spectral Poincare map method for damage detection via single type of measurement, Measurement 113 (2018) 22–37.

[47] T.A. Duffey, S.W. Doebling, C.R. Farrar, W.E. Baker, W.H. Rhee, Vibration-based damage identification in structures exhibiting axial and torsional response, J. Vib. Acoust. 123 (1) (2001) 84, http://dx.doi.org/10.1115/1.1320445.

[48] Los Alamos National Laboratory, SHM Data Sets and Software. [Online]. Available: https://www.lanl.gov/projects/national-security-education-center/engineering/software/shm-data-sets-and-software.php.

[49] W. Ostachowicz, R. Soman, P. Malinowski, Optimization of sensor placement for structural health monitoring: A review, Struct. Heal. Monit. 18 (3) (2019) 963–988, http://dx.doi.org/10.1177/1475921719825601.

[50] C. Zang, M. Imregun, Structural damage detection using artificial neural networks and measured FRF data reduced via principal component projection, J. Sound Vib. 242 (5) (2001) 813–827, http://dx.doi.org/10.1006/jsvi.2000.3390.

[51] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (2014) 1929–1958.

[52] E.L. Droguett, F. Groen, A. Mosleh, The combined use of data and expert estimates in population variability analysis, Reliab. Eng. Syst. Saf. 83 (3) (2004) 311–321, http://dx.doi.org/10.1016/j.ress.2003.10.007.

[53] E. Rabiei, E.L. Droguett, M. Modarres, A prognostics approach based on the evolution of damage precursors using dynamic Bayesian networks, Adv. Mech. Eng. 8 (9) (2016) 1–19, http://dx.doi.org/10.1177/1687814016666747.