UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MECÁNICA

A CAPSULE NEURAL NETWORK BASED MODEL FOR STRUCTURAL DAMAGE LOCALIZATION AND QUANTIFICATION USING TRANSMISSIBILITY DATA

TESIS PARA OPTAR AL GRADO DE
MAGISTER EN CIENCIAS DE LA INGENIERÍA MENCIÓN MECÁNICA
MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL MECÁNICO

JOAQUÍN EDUARDO FIGUEROA BARRAZA

PROFESOR GUÍA
ENRIQUE LÓPEZ DROGUETT

MIEMBROS DE LA COMISIÓN
VIVIANA MERUANE NARANJO
RUBÉN BOROSCHEK KRAUSKOPF

SANTIAGO DE CHILE
2019

RESUMEN DE LA TESIS PARA OPTAR
AL TÍTULO DE: Ingeniero Civil Mecánico y grado de Magister en Ingeniería Mecánica
POR: Joaquín Eduardo Figueroa Barraza
FECHA: Enero de 2019
PROF. GUÍA: Enrique López Droguett

A CAPSULE NEURAL NETWORK BASED MODEL FOR STRUCTURAL DAMAGE LOCALIZATION AND QUANTIFICATION USING TRANSMISSIBILITY DATA

Dentro de la ingeniería estructural, el monitoreo de condición usando diferentes tipos de sensores ha sido importante en la prevención de fallas y diagnóstico del estado de salud. El desafío actual es aprovechar al máximo las grandes cantidades de datos para entregar mediciones y predicciones precisas. Los algoritmos de aprendizaje profundo abordan estos problemas mediante el uso de datos para encontrar relaciones complejas entre ellos.

Entre estos algoritmos, las redes neuronales convolucionales (CNN) han logrado resultados de vanguardia, especialmente cuando se trabaja con imágenes. Sin embargo, existen dos problemas principales: la incapacidad de reconocer imágenes rotadas como tales, y la inexistencia de jerarquías dentro de las imágenes. Para resolver estos problemas, se desarrollaron las redes de cápsulas (Capsule Networks), logrando resultados prometedores en problemas de tipo *benchmark*.

En esta tesis, las Capsule Networks se modifican para localizar y cuantificar daños estructurales. Esto implica una tarea doble de clasificación y regresión, lo que no se ha realizado anteriormente. El objetivo es generar modelos para dos casos de estudio diferentes, utilizando dos algoritmos de *routing* diferentes. Se analizan y comparan los resultados entre ellos y con el estado del arte.

Los resultados muestran que las Capsule Networks con *Dynamic routing* logran mejores resultados que las CNN, especialmente cuando se trata de valores falsos positivos. No se observa sobreajuste en el conjunto de validación sino en el conjunto de prueba. Para resolver esto, se implementa la técnica de *dropout*, mejorando los resultados obtenidos en este último conjunto.

RESUMEN DE LA TESIS PARA OPTAR
AL TÍTULO DE: Ingeniero Civil Mecánico y grado de Magister en Ingeniería Mecánica
POR: Joaquín Eduardo Figueroa Barraza
FECHA: Enero de 2019
PROF. GUÍA: Enrique López Droguett

A CAPSULE NEURAL NETWORK BASED MODEL FOR STRUCTURAL DAMAGE LOCALIZATION AND QUANTIFICATION USING TRANSMISSIBILITY DATA

In the field of structural engineering, health monitoring using different kinds of sensors has taken an important place in failure prevention and health assessment. The current challenge is to take the most advantage of large amounts of data, to deliver accurate measurements and predictions. Deep Learning algorithms tackle these problems by using data to find complex relations between them.

Amongst these algorithms, Convolutional Neural Networks (CNN) have achieved state-of-the-art results, especially when working with images. However, there are two main issues with them: the incapability of recognizing rotated images as such (instead, the algorithm recognizes them as two different images), and the inexistence of hierarchies within images. To solve these problems, Capsule Networks were developed, achieving promising results in benchmark problems, like MNIST.

In this thesis, Capsule Networks are used to locate and quantify structural damage, which means a dual classification (localization) and regression (quantification) task. Until now, this has not been done with vanilla Capsule Networks. The objective is to generate models for two different case studies, using two different routing algorithms. Results are analyzed and compared between them, and with the state-of-the-art.

Results show that Capsule Networks with Dynamic Routing achieve better results than CNN, especially when it comes to false positive values. Overfitting was observed not in the validation set but in the test set. To solve this, dropout was implemented, improving the results obtained in the test set.

Acknowledgment

Siento que soy una persona que tiene mucha suerte, porque toda la vida me he rodeado y he conocido personas maravillosas, las que me han enseñado mucho. A esas personas quiero agradecer.

A mis padres, por ser un permanente ejemplo de lo que es el amor incondicional, el esfuerzo, la bondad y la importancia de la familia. Todo lo que soy es gracias a ustedes. A mis hermanos, los cuales día a día me enseñan de persistencia y a nunca rendirse ante las dificultades de la vida, cada uno a su manera.

A mis padrinos, porque toda mi vida he sentido su cercanía, preocupación, y sentido de enseñanza.

A mis amigos de infancia, con quienes aún comparto después de muchos años de amistad. Gracias por las conversaciones, las risas (muchas) y el cariño. Aquí quisiera mencionar a Pedro, Paulina y Javiera.

A mis amigos de mecánica. Tuve el placer de no solo conocer personas tremendamente inteligentes de las que aprendí mucho, sino que también personas nobles y de buen corazón. En particular, quisiera mencionar a Matías, Gastón, Ignacio y Sergio.

Al profesor Enrique López, quien ha sido muy importante en estos últimos años para mi formación académica. Agradezco las oportunidades que me ha dado, la comprensión, la ayuda, y la paciencia. Parte importante de lo que estoy haciendo ahora es gracias a él.

Quisiera agradecer a la vida por poner en mi camino a todas esas personas.

Se agradece al programa "Geometallurgical Modeling and Mine Planning" del proyecto "CSIRO Chile International Centre of Excellence in Mining and Mineral Processing" Codigo 10CEII-9007.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

1. Introduction

1.1 General Background

In every engineered system safety is crucial, and as technological progresses are made constantly, more complex systems are built. This sometimes leads to more dangerous tasks, which is why nowadays safety is taking a more relevant role in engineering.

Safety also plays a relevant role in structural engineering. When designing any kind of structure, the first question to be asked is: will the structure be able to support the common loads this kind of structures suffer? The main goal is to prevent accidents, and it is achieved by constantly monitoring the structure and its damage state. Structures undeniably suffer damage through their operational life, and the mission is to study it and to be able to anticipate the point where this damage surpasses a certain threshold that leads to an accident.

Structural damage assessment is a subject in which studies are made to take preventive actions against some possible disastrous outcome regarding structures. Through the studies of their behavior at different operating conditions, there are two main objectives: obtain the diagnostic of the structure's health at a certain period of time (or another quantity used to measure the structure's life) and perform an accurate prediction for the structure's future health state. To do this, the concepts of *health* and *damage* must be defined.

Different kinds of analysis can be done to achieve any of these two goals. Vibration analysis is one of the most popular, because vibrations are strongly related to damage [1]–[3]. It has been studied that, through vibration analysis, properties of a structure can be determined. The use of this properties facilitates the damage analysis.

One of the main challenges in vibration-based damage assessment is how to measure data in a way of successfully sensing damage. Using the raw measure of a series of accelerometers could be the most straightforward way, but there may be information about the structure that cannot be identified clearly through this method. The idea of directly using the transmissibility functions (TF) has attracted many researchers [4]–[18].

TF relate the responses between two points of the structure. Among all dynamic responses, transmissibility functions are the easiest to obtain in real-time because the in-situ measurement is straightforward. The advantage is that no modal extraction is needed, thus contamination of the data with modal extraction errors is avoided and they are identified from response only data. Therefore, it does not involve the measurement of excitation forces. In [5], [12]–[14], the use of transmissibility for fault detection is introduced and validated in a series of papers from the same research group. The first work uses TFs to detect damage in a simple simulated lumped-parameter mechanical system. It is the first use of TFs as a tool to detect damage. The rest of the aforementioned works validate the approach through experimental procedures on simplified model of a metallic aircraft wingbox (i.e., a plate incorporating stiffening elements), and a Folland Gnat training aircraft wing.

Up to date, the most common way to use vibration data to perform structural damage analysis is through complex models built over theoretical concepts about vibrations, sometimes aided by Finite Element (FE) models. These models are updated by optimization algorithms [19]–[21]. For example, in [22] authors detect cracks in a beam using a genetic algorithm, first defining a new beam element with a number of embedded transverse edge cracks for computing natural frequencies, and then solving an optimization problem to search for the solution. In [6], Meruane uses a linear approximation method along with antiresonant frequencies that are identified from transmissibility functions, which leads to the solving of multiple nonlinear equations.

As an alternative to model-based algorithms in vibration analysis, machine learning algorithms have been proposed in a data-driven approach. The main reason is how slow these model-based algorithms are, making them not useful for real time applications. Researchers have used neural networks along with feature extraction to learn from data and perform different actions [23]–[26]. In [27], authors use neural networks for damage detection in truss and frame structures, being this one of the first applications of neural networks in the field. Meruane [28] uses an online sequential extreme learning machine (OS-ELM) to improve neural networks' learning speed and reduce the number of parameters. The neural network is used to identify and quantify damage in two experimental cases: an 8 DOF spring-mass system and a beam under different damage scenarios.

In all of the aforementioned works, feature extraction is key. Generally, this process is performed by experts in the matter, but there is no consensus to a “correct” feature extraction process. To solve this issue, and with a general motivation towards automation, deep learning techniques have proven useful. This also applies to fault diagnosis [29]–[34]. In [30], authors use deep neural networks (DNN) for fault diagnosis in rolling element bearings and planetary gearboxes, achieving better results than those achieved with artificial neural networks (ANN). Data is fed as frequency spectra and it is used to train a model capable of classifying among a number of health conditions, showing that the model successfully mines fault characteristics from the signals. In [31], Gan introduces hierarchical belief networks for fault diagnosis, by stacking two deep belief networks (DBN), the first one for identifying fault types, and the second one for recognizing fault severity rankings, taking as input the information from the first DBN. This is applied on a defective bearing dataset.

Among deep learning techniques, Convolutional Neural Networks (CNN) stand out in reliability problems for their automatic feature extraction process. Since CNNs were built to be worked on images, the most common application is image recognition [32], [35]–[40] . In [32], authors use CNNs for fault diagnosis from scalograms obtained from vibration signals.

Although CNNs have been used for years, achieving state-of-the-art results in various fields, there are some problems with their construction. CNNs are not capable of recognizing hierarchies within an image, as all neurons in a low-level layer are sent to a high-level layer. Instead, neurons should be distributed into higher level capsules according to their output, as neurons specify themselves during training.

There has been some research to solve the existing drawbacks within CNNs. In [41], Hinton realizes that, while machine learning community is using (at that time) 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. In [42], Sabour presents 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 novel *routing* algorithm. In each routing iteration, capsules predict the output of the next layer's capsules. 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 [43]. In [44] Hinton presents a different view on capsules. 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, were 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 [45], Afshar uses *Magnetic Resonance Imaging* (MRI) images to classify types of brain tumors with CapsNets, achieving better results than those obtained using CNNs. In [46], Upadhyay uses capsule networks in *Generative Adversarial Networks* (GANs) as a replacement to CNN discriminators, resulting in an improvement of the generator's performance. Xi [47] analyses capsule networks' performance on CIFAR10 dataset, which contains images of different classes of objects. The highlight for this research is that CIFAR10 is known to be more complex than MNIST, in terms of features. This means the classification task more challenging than with MNIST. In [48] LaLonde adapts capsule networks to object segmentation. This is applied to pathological lung segmentation using *Computed Tomography* (CT) scans. In [49], Andersen uses capsule networks in deep Reinforcement Learning (RL) in game environments achieving viability but lack of scalability. In [50], authors modify EM routing algorithm 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 [51]. In [52], authors train sparse latent capsules using only reconstruction loss, thus performing unsupervised training, achieving better generalization on benchmark datasets MNIST and affNIST. It can be noticed that, of all the aforementioned applications of capsule networks, none relate to structural damage assessment. Actually, most of them are based on benchmark studies, revealing that CapsNets are still in an early stage of development. Also, Capsule Networks have not been used for regression tasks, only classification or clustering tasks. There is still an important research area within Capsule Networks which hasn't been studied yet.

In this thesis, Capsule Networks will be used to locate and quantify structural damage. Two different routing algorithms (Dynamic Routing and EM Routing) will be used to simultaneously perform classification (damage localization) and regression (damage quantification). Two different datasets will be available: one corresponding to a spring-mass system and another corresponding to a beam under different damage scenarios. Each of them will be studied in different cases according to the number of damaged elements.

1.2 Objectives and Statement

1.2.1 General Objective

Develop a new Capsule Network for structural damage localization and quantification based on transmissibility functions data.

1.2.2 Specific Objectives

- Developing a Deep Learning architecture capable of extracting relevant features from transmissibility functions.
- Create a model for damage localization and quantification using Capsule Networks and dynamic routing between capsules (DR).
- Create a model for damage localization and quantification using Capsule Networks and EM routing between capsules (EMR).
- Apply and validate the DR model's performance on two experimental cases: an eight degree-of-freedom (DOF) mass-spring system and a beam under multiple damage scenarios.
- Apply and validate the EMR model's performance on two experimental cases: an eight degree-of-freedom (DOF) mass-spring system and a beam under multiple damage scenarios.
- Compare results with the state-of-the-art

1.2.3 Statement and Thesis Scope

In this thesis, a novel Deep Learning algorithm called Capsule Networks will be used to analyze two case studies: A mass spring system and a beam under different damage conditions. According to its properties, Capsule Networks should be an upgrade to CNN, which is why the direct comparison will be with that model.

There are two types of Capsule Network models, according to the routing algorithm: Dynamic Routing, and EM Routing. Both algorithms will be used, following the steps the respective papers used.

2. Theoretical Background

This chapter describes the most relevant theoretical aspects necessary for this thesis' development.

2.1 Structural Damage

As seen in chapter 1, structural damage can be represented in more than one way. In this thesis, damage is represented by a reduction in stiffness, as expressed in equation 2.1:

$$\boldsymbol{K}_i^d = (1 - y_i)\boldsymbol{K}_i \tag{2.1}$$

, were $\boldsymbol{K}_i^d$ is the damaged stiffness of the i-th element, $\boldsymbol{K}_i$ is the undamaged stiffness, and y_i is the stiffness reduction of the i-th element. This has shown good results in damage detection algorithms [53].

2.2 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 describes numerically how vibrations propagate through an element. This is very useful for isolation purposes. It is a common objective to reduce the level of vibrations on a certain system. To do this, isolators are installed to damp the propagation of vibrations. Clearly transmissibility measures are relevant to achieve this purpose.

There are two ratios that apply the concept of transmissibility. They are *force transmissibility* and *displacement transmissibility*. Force transmissibility is the ratio between the response and the stimulus in terms of force, and 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, this is, responses in different positions. This being said, accelerometers can be installed at various locations of an element, thus generating many transmissibility functions. Experimental transmissibility functions are calculated by:

$$T_{ir}^{k}(\omega) = \frac{X_{ik}(\omega)}{X_{rk}(\omega)} \tag{2.2}$$

, were $T_{ir}^{k}(\omega)$ 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 it can be seen, only the location of the exciting force is needed, not its magnitude, which is an advantage. Sometimes it may be useful to use an alternative method:

$$T_{ir}^{k}(\omega) = \frac{X_{ik}(\omega)X^{*}{}_{rk}(\omega)}{X_{rk}(\omega)X^{*}{}_{rk}(\omega)} \tag{2.3}$$

, were $X^{*}{}_{rk}(\omega)$ is the complex conjugated of $X_{rk}(\omega)$. This is done to reduce uncorrelated noise.

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

$$\boldsymbol{M}\ddot{x} + \boldsymbol{C}\dot{x} + \boldsymbol{K}x = \boldsymbol{f}(t) \tag{2.4}$$

, 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, equation 2.4 is transformed via Fourier Transform:

$$(-\omega^2\boldsymbol{M} + j\omega\boldsymbol{C} + \boldsymbol{K})\boldsymbol{X}(\omega) = \boldsymbol{F}(\omega) \tag{2.5}$$

, were ω is the frequency and $j = \sqrt{-1}$. From equation 2.5, the Frequency Response Function $\boldsymbol{H}(\omega)$ is defined:

$$\boldsymbol{X}(\omega) = (-\omega^2\boldsymbol{M} + j\omega\boldsymbol{C} + \boldsymbol{K})^{-1}\boldsymbol{F}(\omega) = \boldsymbol{H}(\omega)\boldsymbol{F}(\omega) \tag{2.6}$$

$$\boldsymbol{H}(\omega) = (-\omega^2\boldsymbol{M} + j\omega\boldsymbol{C} + \boldsymbol{K})^{-1} \tag{2.7}$$

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

$$\boldsymbol{H}(\omega) = \frac{\boldsymbol{X}(\omega)}{\boldsymbol{F}(\omega)} \tag{2.8}$$

Thus, the experimental transmissibilities can be obtained through the definition:

$$T_{ir}^{k}(\omega) = \frac{\boldsymbol{X}_{ik}(\omega)}{\boldsymbol{X}_{rk}(\omega)} = \frac{\boldsymbol{H}_{ik}(\omega)}{\boldsymbol{H}_{rk}(\omega)} \tag{2.9}$$

2.3 Machine Learning and Deep Learning

Machine learning is a categorization for many algorithms in which there is a "learning" process. The algorithm is fed with data, and this data is used to adjust the model so that a specific task is performed.

There are two main tasks in machine learning. These are: *supervised* and *unsupervised* learning, and they differ in the existence of feedback. In supervised learning, when the model is adjusting itself (from now on, this process will be called *training*), it receives the "correct answer". For example, recognizing hand-written number images is a supervised learning task, if during training the model "sees" the correct label for each image, and training is based on the difference between the answer delivered by the model, and the correct one. By contrast, unsupervised learning tasks do not count with a "correct answer", and separate data using only the properties of the data itself, and not a label.

Most of the supervised learning tasks applicable to reliability engineering are *regression* and *classification*. In regression, the task is to determine a mapping function from an input variable to a continuous output variable, in order to use this function to predict new outputs when new input is fed. Classification is just like regression, with the difference that the output is a discrete *class label*. The model learns the mapping function between the input data and its corresponding labels.

In unsupervised learning, the most common task is *clustering*, in which the model tries to divide data by creating different groups (or *clusters*) and assigning data to one of this groups. There is no label, meaning the algorithm determines the criteria for both division and assignment.

Deep learning corresponds to a series of novel and advanced algorithms in machine learning. There is no official definition for what deep learning is, but there is some consensus that it involves the extraction of complex features and the use of complex nonlinearities. Many of these algorithms use stacking of layers, in which every layer learns representation about data at different levels of abstraction. This presents the main difference with normal machine learning algorithms. Because deep learning models are built with more layers, there are more sources on nonlinearities. This makes the features more complex.

2.4 Feed Forward Neural Networks (FFN)

In machine learning, a Feedforward Neural Network (FNN) is a model in which a function is approximated through a series of operations and function compositions to perform regression or classification. It is called like that because of its resemblance with neural activity on humans and other animals. The motivation behind is to imitate the way some animals (including humans) perform cognitive activities. In the nervous system of most mammals there are neurons. These are the basic unit for the system, and they are responsible for receiving, processing and transmitting information through electrical and chemical signals. Neurons have an input region, an output region, and an activation state. It will be seen later that all of these properties are relevant parts of a FFN.

The basic unit of a FNN is a *neuron*. A neuron takes the output of predecessor neurons and performs a weighted sum, plus a bias. This result is taken as an input for another neuron.

$$y = \sum_i f((W_i * x_i) + b) \tag{2.10}$$

Neurons are organized into layers, as it is shown in figure 2.1. Inputs and outputs define an *input layer* and an *output layer*, respectively. Layers connecting input and output layers are called *hidden layers*.

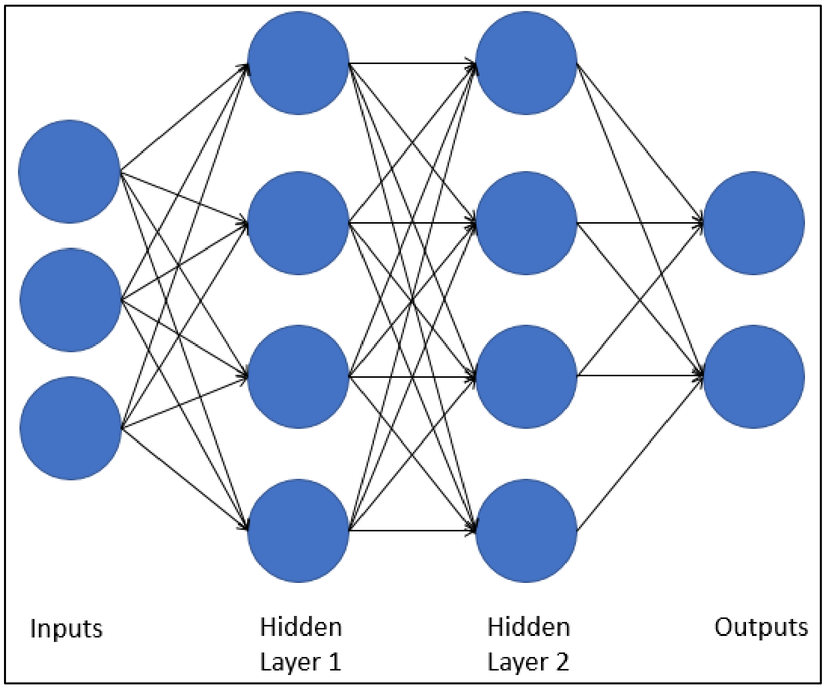


Figure 2.1 Neural Network diagram

In a FNN, a certain function is applied to neurons in one layer before taking that output to the next layer. This function is called a*ctivation function*, and acts as a source for nonlinearity in the model. The activation functions between layers allow the network to learn more complex characteristics. There are several activation functions applied in neural networks, the most used being: *Tanh, RELU* and *Sigmoid*.

- Linear
- Tanh
- RELU
- Sigmoid

In this thesis, the *RELU* function is used. This function turns to 0 every negative number and keeps the positive ones.

$$f(x) = \max(0, x) \tag{2.11}$$

Basically, a FNN establishes a relationship between input and output information, by performing weighted sums and applying activation functions. The neural network adjusts the different values of weights in order to *learn* the most suitable relationship between input and output. The process of learning consists on optimizing *cost function*. Part of building a neural network is choosing an appropriate cost function. This depends mainly on the task (regression or classification), but also on the type of data. For regression, some common cost functions are:

- Mean Squared Error
- Root Mean Squared Error
- Cross-Entropy
- Exponential Cost
- Kullback-Leibler divergence

In this thesis, the following cost function is used:

$$Cost = \frac{1}{NO}\sum(y_i - o_i)^2 \tag{2.12}$$

To evaluate how good the model solves the specific task, a performance metric is defined. There are various kinds of performance metrics mainly depending on the task to be performed (regression or classification). For classification, some typical performance metrics are:

- Accuracy
- Confusion Matrix
- Precision

- Sensitivity
- Specificity
- F1 Score

On the other hand, for regression, some performance metrics are:

- Accuracy
- R^2 Score
- Mean Squared Error

Other metrics can be constructed, depending entirely on the specific problem to be solved

2.5 Backpropagation

The process in which the different weights and biases in a NN are modified for the model to "learn" some relation is called *training*. Training is the process in which a set of parameters θ is to be found to minimize a cost function $J(\theta)$. Intrinsically this means obtaining θ such that $J'(\theta) = 0$. For simple models (like a linear regression, for example), this optimization problem presents no major difficulties, for the computation of $J'(\theta)$ is straightforward. In deep learning models, this calculation cannot be done by conventional methods. *Backpropagation* is a way of calculating the gradient for the cost function, $J'(\theta)$, to then optimize the network, most probably using *stochastic gradient descent*. The goal is to calculate the partial derivate with respect to any weight and bias in the network. To do this, backpropagation calculates the error at the end of the network and propagates it to the its beginning.

First, the output is computed through forward propagation. Then, the error according to the last hidden layer is calculated:

$$\delta^L = a^L - y \tag{2.13}$$

, were a^L is the output generated by the last layer L, and y corresponds to the network's correct output. Using this value, the error for the previous layers is calculated:

$$\delta^{l-1} = \mathbf{\Theta}^{l-1}\delta^l g'(z^{l-1}) \tag{2.14}$$

, were $\mathbf{\Theta}^{l-1}$ is a set of hyperparameters, and $g'(z^{l-1})$ is the derivative for the activation function. After each error calculation, a Δ coefficient (originally initialized as $\Delta = 0$) is updated:

$$\Delta^l = \Delta^l + a^l\delta^{l+1} \tag{2.15}$$

This process is repeated with each neuron of each layer in the network, and with every element in the dataset. Finally, D_{ij}^l is calculated as:

$$D_{ij}^l = \begin{cases} \frac{1}{m}\Delta_{ij}^l + \lambda\Theta_{ij}^l & if\ j \neq 0 \\ \frac{1}{m}\Delta_{ij}^l & if\ j = 0 \end{cases} \tag{2.16}$$

, where λ corresponds to a regularization term, and $j = 0$ means the hyperparameter is a bias. The i index indicates the elements in the dataset. D_{ij}^l is calculated because it is proven that $D_{ij}^l = \frac{\partial}{\partial\Theta_{ij}^l}J(\Theta)$, which is the objective of backpropagation.

2.6 Minibatch Gradient Descent

After computing the gradient using backpropagation, the optimization problem can be solved. In machine learning, and especially deep learning, lots of data are used to train models. This makes the gradient computation very expensive.

To solve this issue, the training set is divided into *minibatches*. The gradient is computed using one minibatch, and weights and biases are updated through:

$$\Theta \rightarrow \Theta - \epsilon \mathbf{g} \tag{2.17}$$

, where Θ represents the weights and biases, **g** is the gradient, and ϵ is a hyperparameter called *learning rate*. After Θ is updated, the process is repeated with the next minibatch, and so on until the whole training set is covered. Each update process is called *step*. The whole training set is covered a certain number of times. This number is called *epoch*. It is said that each time the whole training set is used, an epoch has passed.

2.7 Overfitting and Dropout

The process of training depends of a dataset. The model learns properties or features from this dataset with the objective of applying these properties to predict an output when a new unknown input is fed. To perform a correct prediction, the model must learn properly the features from the training set and must be capable of generalizing. The lack of generalization is called *overfitting* and occurs when the model fails to predict additional data because it is too adjusted to the training set. Most of the times this happens because the model interprets the data's noise as a variation due to the data's structure.

To prevent overfitting, there are some techniques that have proven useful, such as cross-validation, regularization, or early stopping. In the context of neural networks, *dropout* has shown great results. In [54], authors present and explain the technique, in which some neurons are "turned off" during training, with some propability 1-*p*, which is independent for every neuron. This is done with every neuron of a layer in the forward and backwards pass and can be done in every layer. With this, the network reduces its size, not using all of its "tools" to perform learning, though in testing, all the neurons are used.

2.8 Convolutional Neural Networks (CNN)

A Convolutional Neural Network (CNN) is a kind of NN specially orientated to image recognition tasks. It is just like a NN, with neurons, layers, weights and activation functions, but the input images are submitted to mathematical operations for feature extraction. These are *convolutions* and *pooling*.

Mathematically, a convolution is represented by equation 2.18, where $S(i,j)$ is the output of the convolution, and $I(i-m,j-n)$ is the input. It is an "inspection" though the image with a *filter* or *kernel* $K(m,n)$. This kernel goes through an image in an ordered way making operations over the images values with weights. These weights are updated during training to extract features. A visual representation of a convolution is shown in figure 2.2.

$$S(i,j) = \sum_{m}\sum_{n} I(i-m,j-n)K(m,n) \tag{2.18}$$

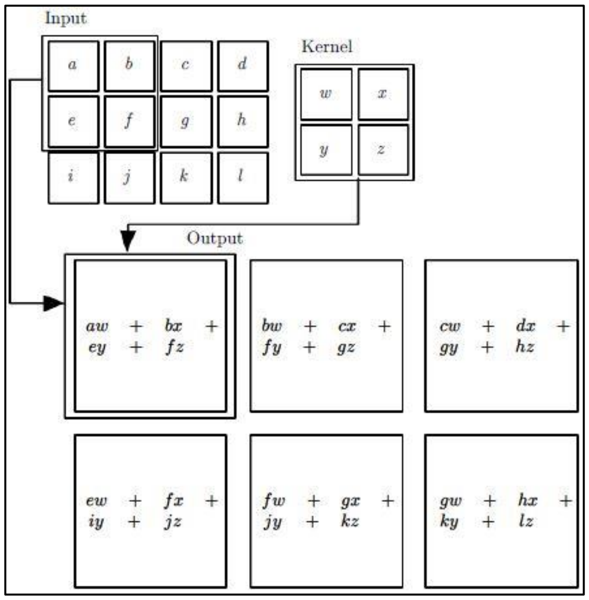


Figure 2.2 Convolution operation [55]

In the figure, it can be seen how the kernel operates. It takes a subset of the input with its same size and performs a matrix operation between that subset and the weights of the kernel. Then it moves to the right (the amount of positions that the kernel moves is called *stride*. In this case, its value is 1 in both axis) and performs the same operation. The process described before can be repeated many times, each repetition generating a different set of weights. These different set of weights learn different features of the input image. These features are presented in *feature maps*. After a convolution, a bias is added, which is also learned through training. An activation function is then applied, just like a NN.

Generally, the number of parameters used to construct a CNN can be very large. This makes the computation process very expensive. *Pooling* is basically an operation to address this issue. In a way that's similar to the convolution's kernel, a small "window" inspects the image and summarizes the information in each subset of the image. Typically, this is done by taking the maximum value in the subset (*max-pooling)* or the average (*average pooling*). In image data, it can be seen that max-pooling extracts more significant features than average pooling, whereas average pooling extracts features more smoothly. Figure 2.3 shows an example where both max pooling and average pooling are applied to some values.

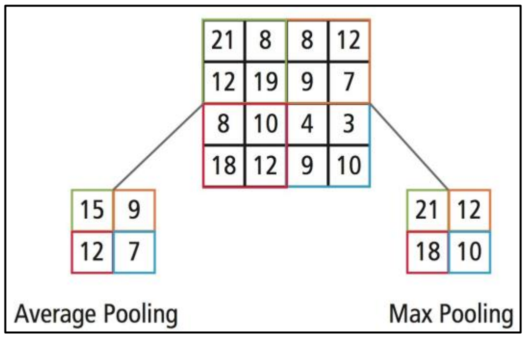


Figure 2.3 Difference between max pooling and average pooling [55]

2.9 Capsule Neural Networks (CapsNets)

Most deep learning algorithms are born as an effort to imitate the way the human brain uses different information and makes the associative processes that lead to learning. Neural Networks is the clearest example. Each “neuron” in the network is called like that because its properties are vaguely similar to actual neurons in the human nervous system. While artificial neurons have a number of input channels, a stage in which input is processed, and an output which can serve as 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 which can be tuned (strides).

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 “work” although they shouldn’t, or that they 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. Because all of the features in that region are represented by one value (that normally is 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 doesn't really *understand* those changes. This is called *invariance*.

To tackle the problem of invariance, a novel architecture has been built. This is, in a way, inspired by cortical microcolumns [56]. Microcolumns are sets of neurons organized in vertical columns. In [57], the authors 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 the article, a single microcolumn being paired to a sensor (a fingertip, for example) generates a location signal as the sensor approaches to an object. This location signal activates the neurons in the microcolumn that can decode the signal to recognize a feature according to the signal. 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 objects containing the sensed feature at the sensed location. This action is repeated at multiple locations to discard possible elements and to identify accurately 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, Capsule Networks are created. A Capsule Network (CapsNet) 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's 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 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 come from a lower-level capsule layer also). These neurons evaluate the most possible output for the next layer's capsules. The "final" output is decided based on a measure of agreement between all the predictions. This last step is called *routing by agreement* and is an iterative process. The predictions are used to compute the actual output, and the agreement between this two determines the predictions done by capsules in the next iteration, which are used to update the output. This is done a few times (typically, 3 to 5).

Capsule Networks have been presented in two papers. In one [42] capsules are represented by vectors (very similar to a microcolumn), in which every unit is a neuron and represents a feature value. The routing algorithm is called *Dynamic Routing* and is similar to a clustering algorithm using Euclidean distance. Predictions within this cluster will have greater activation value, meaning their predictions will have more importance.

In the second paper [44] capsules are no longer seen as one entity, but two: a *pose matrix* and an *activation unit*. Also, the routing algorithm is modified with respect to the first paper and is called *EM Routing*. It is also similar to a clustering algorithm, but each cluster is governed by a gaussian distribution.

In the following subsections, capsules and routing are explained according to both publications.

2.9.1 Capsules and Routing according to "Dynamic Routing Between Capsules"

A capsule in the work of Sabour [42] 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's activation probability. For the vector to represent a probability, its length value must be between 0 and 1, which is why a "squashing" function is introduced:

$$v_j = \frac{\|s_j\|^2}{1 + \|s_j\|^2} \frac{s_j}{\|s_j\|^2} \tag{2.19}$$

, where v_j is the "squashed" value of the capsule's output s_j.

The first layer of capsules comes from the output of an already known convolution. This output is rearranged into vectors with a previously specified dimension (and 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's 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 measure between predictions.

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

$$\hat{\boldsymbol{u}}_{j|i} = \boldsymbol{W}_{ij}\boldsymbol{u}_i \tag{2.20}$$

, 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 the 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 in layer *l+1* s_j corresponds to a weighted sum over all $\hat{\boldsymbol{u}}_{j|i}$ (and shrunk using the squashing function):

$$s_j = \sum_i c_{ij}\hat{\boldsymbol{u}}_{j|i} \tag{2.21}$$

$$v_j = \frac{\left\|s_j\right\|^2}{1 + \left\|s_j\right\|^2} \frac{s_j}{\left\|s_j\right\|^2} \tag{2.22}$$

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

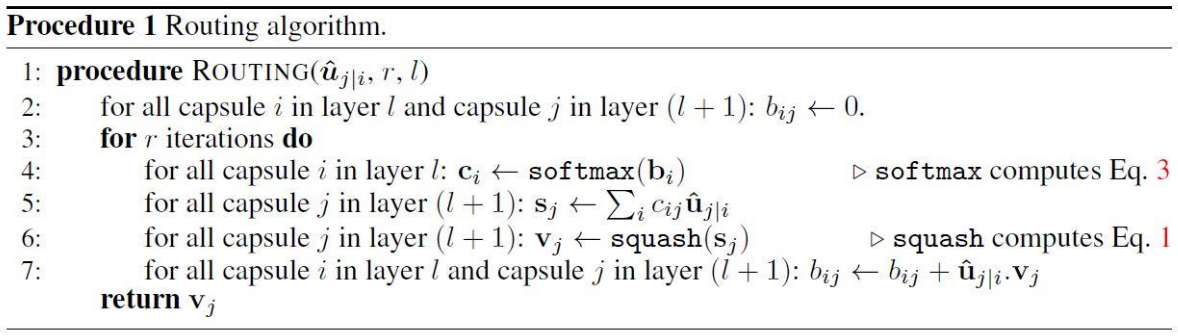
$$c_{ij} = \frac{\exp(b_{ij})}{\sum_k exp\ (b_{ik})} \tag{2.23}$$

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

$$b_{ij} \leftarrow b_{ij} + \hat{\boldsymbol{u}}_{j|i} \cdot v_j \tag{2.24}$$

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

To summarize, dynamic routing computes the output v_j of a capsule in layer *l+1* by performing a weighted sum over the predictions of capsules in layer *l* and "squashing" that vector. Those weights are refined in each inner iteration according to the agreement between the previous v_j and the prediction done by capsules in layer *l*. The number of iterations must be defined beforehand, being 3, 4 and 5 the recommended ones. The coupling coefficients are refined during the computation of v_j and have no incidence in the training process (i.e, training doesn't take into account the tuning of coupling coefficients), while the weights associated to the prediction calculation vary during training. Figure 2.4 shows a pseudo-code of the algorithm.



```
Procedure 1 Routing algorithm.
1: procedure ROUTING($\hat{\mathbf{u}}_{j|i}$, $r$, $l$)
2:     for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$: $b_{ij} \leftarrow 0$.
3:     for $r$ iterations do
4:         for all capsule $i$ in layer $l$: $\mathbf{c}_i \leftarrow \mathtt{softmax}(\mathbf{b}_i)$        ▷ softmax computes Eq. 3
5:         for all capsule $j$ in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}$
6:         for all capsule $j$ in layer $(l+1)$: $\mathbf{v}_j \leftarrow \mathtt{squash}(\mathbf{s}_j)$        ▷ squash computes Eq. 1
7:         for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
       return $\mathbf{v}_j$
```

Figure 2.4 Dynamic routing algorithm

After this procedure, the output of capsules in layer *l+1* may serve for two main purposes: input to a NN or for another routing procedure to compute the output for capsules in a *l+2* layer.

2.9.2 Capsules and Routing according to "Matrix Capsules with EM Routing"

A matrix capsule, like seen in figure 2.5 is composed of two entities: a *pose matrix* and an *activation unit*. Like it says, the pose matrix captures the pose of the image in space. This is done in order for the algorithm to recognize rotated images one image rotated in different angles and not different objects, thus requiring less data to train and 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.

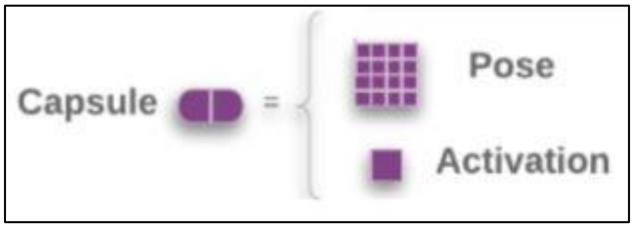


Figure 2.5 Matrix capsule representation

Similar to the previous work on capsule networks, the first instance capsules appear is after a convolution. In this case though, not only a rearrangement (*reshape* operation) is necessary. To get 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 get the activation value, another parallel convolution is performed to get a single value output.

To get the votes (predictions done by the capsules in layer *l* about capsules in layer *l+1*), the capsules are multiplied by a weights matrix which is modified during training process. These votes are used to compute the output for capsules on layer *l+1*, using a novel 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 cluster (or capsule) j in layer l+1. This is done through the following equations:

$$\boldsymbol{p}_j = \frac{1}{\sqrt{\prod_h^H 2\pi\left(\sigma_j^h\right)^2}} \exp\left(-\sum_h^H \frac{\left(V_{ij}^h - \boldsymbol{\mu}_j^h\right)^2}{2\left(\sigma_j^h\right)^2}\right) \tag{2.25}$$

$$\boldsymbol{R}_{ij} = \frac{\boldsymbol{a}_j \boldsymbol{p}_j}{\sum_k \boldsymbol{a}_k \boldsymbol{p}_k} \tag{2.26}$$

In the expression for $\boldsymbol{p}_j$, V_{ij}^h is the h-th dimension of the vote from capsule i to capsule j, $\boldsymbol{\mu}_j^h$ and σ_j^h are the h-th 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 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:

$$\boldsymbol{R}_{ij} = \boldsymbol{R}_{ij} * \boldsymbol{a}_i \tag{2.27}$$

$$\boldsymbol{\mu}_j^h = \frac{\sum_i \boldsymbol{R}_{ij} * V_{ij}^h}{\sum_i \boldsymbol{R}_{ij}} \tag{2.28}$$

$$\left(\sigma_j^h\right)^2 = \frac{\sum_i \boldsymbol{R}_{ij} * \left(V_{ij}^h - \boldsymbol{\mu}_j^h\right)^2}{\sum_i \boldsymbol{R}_{ij}} \tag{2.29}$$

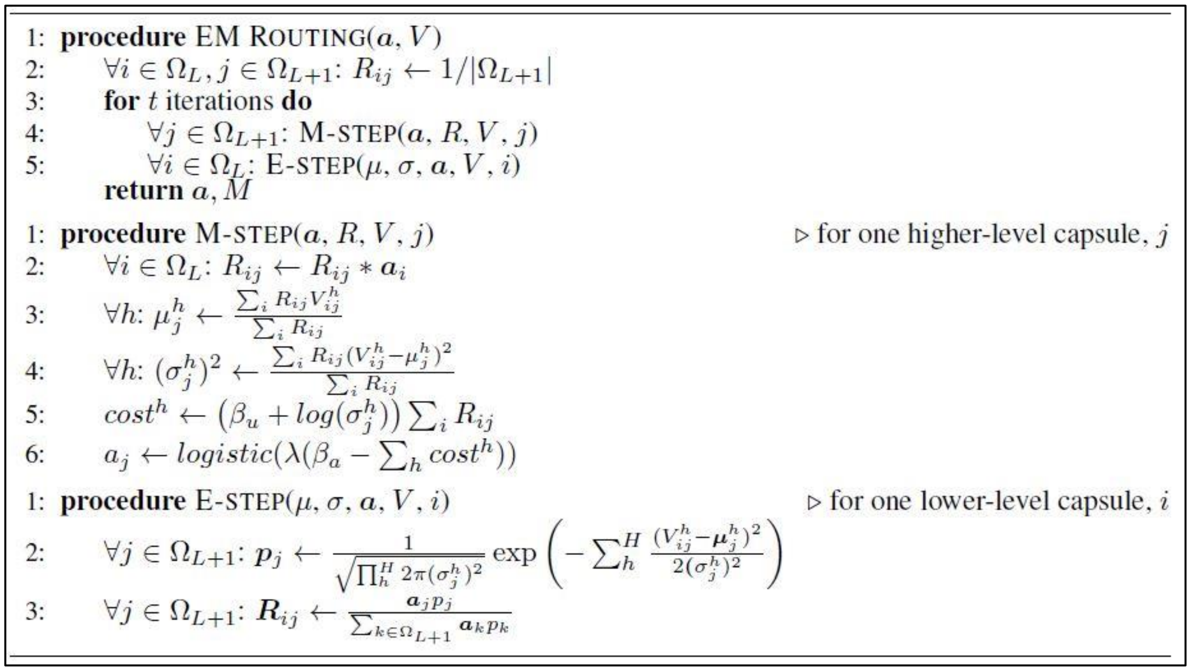
The clusters are redefined in order to minimize a *cost function* that takes into account the probabilities that each cluster generates the values shown by the capsules in layer *l*. That cost function is used to update the activation value for each capsule in layer *l+1*:

$$cost^h = \left(\beta_u + \log\left(\sigma_h^h\right)\right) \sum_i \boldsymbol{R}_{ij} \tag{2.30}$$

$$\boldsymbol{a}_j = logistic\left(\lambda\left(\beta_a - \sum_h cost^h\right)\right) \tag{2.31}$$

, 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.

To summarize, EM Routing is used to calculate the capsules' output $\boldsymbol{\mu}_j$ and its' activation values $\boldsymbol{a}_j$. The E-step is used to calculate (or update) the assignment probabilities, which are then used to calculate/update the mean and variance for each capsule in layer *l+1* during the M-step. With this information, the activations are calculated. Figure 2.6 shows the procedure:



```
procedure EM ROUTING($a$, $V$)
    $\forall i \in \Omega_L, j \in \Omega_{L+1}$: $R_{ij} \leftarrow 1/|\Omega_{L+1}|$
    for $t$ iterations do
        $\forall j \in \Omega_{L+1}$: M-STEP($a$, $R$, $V$, $j$)
        $\forall i \in \Omega_L$: E-STEP($\mu$, $\sigma$, $a$, $V$, $i$)
    return $a$, $M$
procedure M-STEP($a$, $R$, $V$, $j$)                              ▷ for one higher-level capsule, $j$
    $\forall i \in \Omega_L$: $R_{ij} \leftarrow R_{ij} * a_i$
    $\forall h$: $\mu_j^h \leftarrow \frac{\sum_i R_{ij} V_{ij}^h}{\sum_i R_{ij}}$
    $\forall h$: $(\sigma_j^h)^2 \leftarrow \frac{\sum_i R_{ij} (V_{ij}^h - \mu_j^h)^2}{\sum_i R_{ij}}$
    $cost^h \leftarrow \left(\beta_u + log(\sigma_j^h)\right) \sum_i R_{ij}$
    $a_j \leftarrow logistic(\lambda(\beta_a - \sum_h cost^h))$
procedure E-STEP($\mu$, $\sigma$, $a$, $V$, $i$)                    ▷ for one lower-level capsule, $i$
    $\forall j \in \Omega_{L+1}$: $p_j \leftarrow \frac{1}{\sqrt{\prod_h^H 2\pi(\sigma_j^h)^2}} \exp\left(-\sum_h^H \frac{(V_{ij}^h - \mu_j^h)^2}{2(\sigma_j^h)^2}\right)$
    $\forall j \in \Omega_{L+1}$: $R_{ij} \leftarrow \frac{a_j p_j}{\sum_{k \in \Omega_{L+1}} a_k p_k}$
```

Figure 2.6 EM Routing algorithm

2.10 Performance Metrics

To evaluate the models' performance, proper metrics must be defined. Since the tasks are to locate and quantify damage, performance must be measured in terms of how the different models quantify damage and locate it.

To measure the models' capacity to quantify damage, *Mean Sizing Error* (MSE) is used:

$$MSE = \frac{1}{NO} \sum |y_i - o_i| \tag{2.32}$$

, 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) measure the fraction of damaged elements unnoticed and undamaged elements tagged as damaged, respectively. DME is given by:

$$DME = \frac{1}{NT}\sum \epsilon_i^l \tag{2.33}$$

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

$$\epsilon_i^l = \begin{cases} 1, & y_i \leq \alpha_c, o_i \geq 0 \\ 0, & \sim \end{cases} \tag{2.34}$$

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:

$$FAE = \frac{1}{NF}\sum \epsilon_i^u \tag{2.35}$$

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

$$\epsilon_i^u = \begin{cases} 1, & y_i \geq \alpha_c, o_i = 0 \\ 0, & \sim \end{cases} \tag{2.36}$$

3. Working Methodology

In order to achieve the objectives stated in the Introduction chapter, a working methodology must be followed. This chapter presents that methodology.

To understand the problem and techniques to solve it, an extensive literature review is necessary, beginning with machine learning topics, neural networks, convolutional neural networks, and finally capsule networks.

To properly train the models, the datasets must be available. Transmissibility images are generated through Finite Element (FE) models that are calibrated with experimental images. In this way, the generated images represent the experimental setups and are seen as an extension of possible cases that, due to time and costs, cannot be generated.

After both datasets are defined, an adequate architecture is proposed. This is done based on prior knowledge of the problem, the datasets, and CapsNets.

With the proposed architecture defined, the most suitable hyperparameters for the architecture are found using prior knowledge and inspection. The obtained results are compared with CNN, trying to keep the architecture close to the one defined for CapsNets, for a proper comparison.

The working methodology is summarized in the following figure:

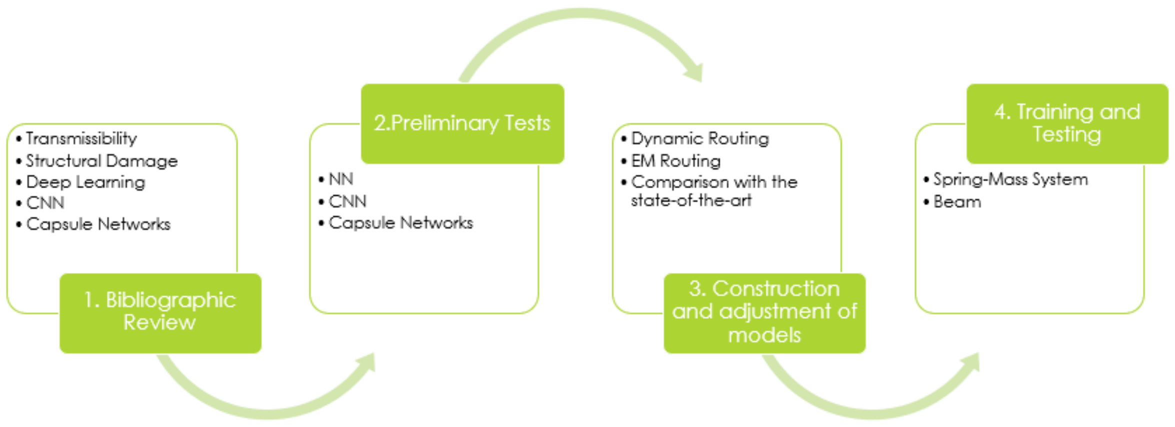


Figure 3.1 Working methodology

4. Proposed Capsule Networks for Damage Localization and Quantification

As exposed in the Introduction chapter, the objective of this thesis is to develop a framework that is suitable not only for structural damage localization, but also quantification. To do this, it is necessary to adapt Capsule Networks for classification and regression.

Vanilla Capsule Networks have been used only for classification. The architecture has two main parts: convolutions and routing. (in [42], a multilayer perceptron is added for reconstruction, with the purpose of avoiding overfitting). 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's output, which are classes. The capsule with the highest activation value represents the predicted class (multiple classes could be accepted also. In that case, the loss function should be modified). This works for structural damage localization, because each capsule would represent an element within the structure. If the capsule's activation value is near to 1, the element presents damage. The method described before doesn't work for regression purposes, because the capsules' activations represent the probability of existence of a property, in this case, structural damage. There is no output value that could represent the amount of damage each value. To fix this, the proposed Capsule Networks includes a MLP that uses the last level capsules' output as input. The MLP's 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.

In the figure below, the proposed model for damage localization and quantification is shown. The input image is submitted to two convolution processes. Then, 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), the secondary capsules are obtained. The number of secondary capsules depends on what they are meant to represent. Finally, information from the secondary capsules is used as input for a MLP, which calculates the amount of damage per element in the structure.

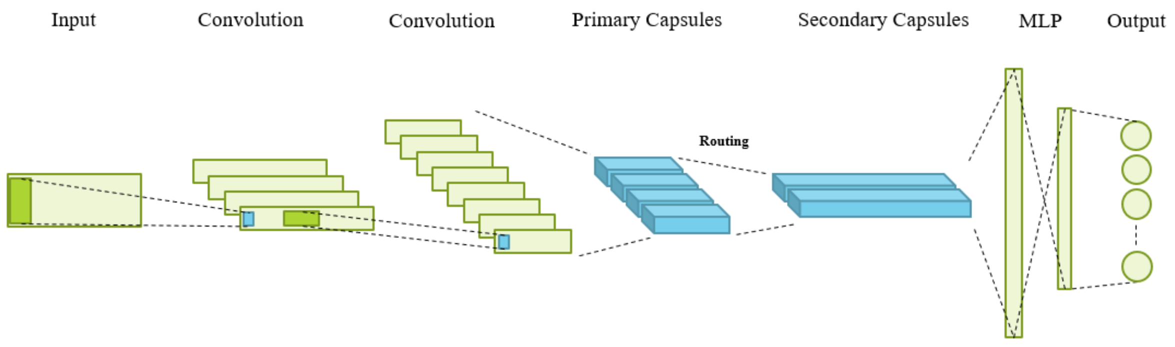


Figure 4.1 Proposed Capsule Networks model

In this thesis, both routing algorithms are studied, each of them through one corresponding architecture. Since there are two case studies, there is a total of four models.

For CapsNets with dynamic routing, the following architecture is proposed:

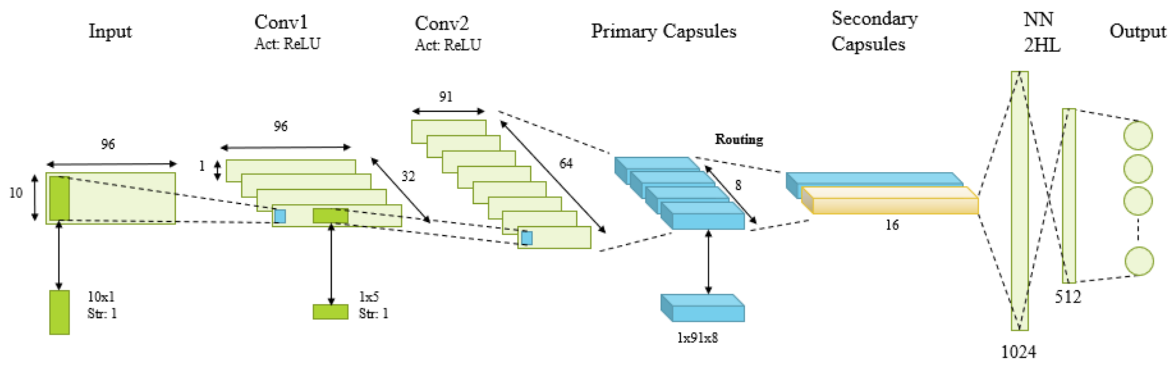


Figure 4.2 Capsule Networks with Dynamic routing, proposed architecture

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 size of 10x1 pixels, and a stride of 1. After the first convolution, another convolution is performed, this time generating 64 feature maps. The filters' size is 1x5, with a stride of 1.

After both convolutions, a ReLU activation function is applied, to inject nonlinearity to the model.

From the 64 feature maps obtained by the last convolution to the primary capsules, a reshape operation is done, which turns the 1x91x64 neurons with a scalar output each into 1x91x8 8-dimensional vectors. Squashing function is applied to all capsules to ensure their lengths represent activation probabilities, and routing is done to compute the secondary capsules' values.

The number of secondary capsules depends on the model. For model 1, there are only two types of images: no damaged elements and one damage element. Therefore, there are only two secondary capsules. Using the same logic, model 2 has three secondary capsules, and model 3 has four. The objective is for capsules to detect damage.

With the secondary capsules' output, the one with larger length is fed to a neural network with two hidden layers, the first with 1024 hidden units and the second one with 512. The final output is a N-dimensional vector, with N being the number of nodes in the system (7 or 18). The cost function to be minimized during training is:

$$C = \frac{1}{N}\sum(y_i - o_i)^2 \tag{3.1}$$

, where y_i is the estimated output, and o_i is the real one.

In the case of EM routing, the architecture used is the one shown in figure 4.3. The first convolution is equal to the one used for dynamic routing. The second convolution is done with 32 1x6 filters, with a stride of 3.

To get the activations and pose matrices for all primary capsules in the next layer, two convolutions are performed simultaneously. The first one doesn't have an activation function, so it is only a linear combination of the Conv2 layer. It outputs a 4x4 matrix for each capsule. The second one outputs the activation logits for each capsule, and to achieve this, a sigmoid activation function is set after the convolution. In this way, the outputs are set to be between 0 and 1.

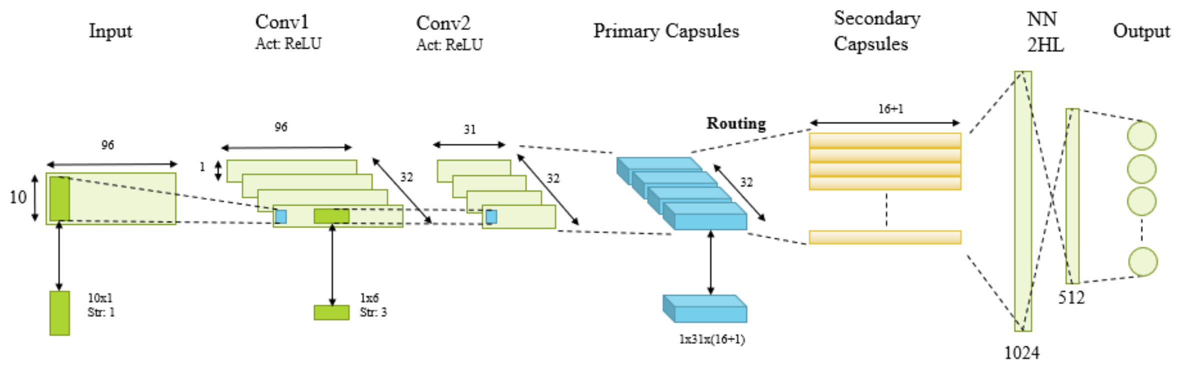


Figure 4.3 Capsule Networks with EM routing, proposed architecture

At first, no dropout is used in any layer of the network, neither convolutions nor fully connected layers. A second model is trained using dropout in the first hidden layer of the neural network for regression, with $p = 0{,}5$. This is done only with the dynamic routing type architecture.

5. Results

5.1 Case Study 1: Spring-mass system

The first case study corresponds to a spring-mass system, which consists of eight aluminum disk masses separated by seven springs. The excitation force comes from a shaker connected to the first mass. One accelerometer is connected to each mass. These accelerometers measure horizontal acceleration data and are used to calculate transmissibility functions. Data is acquired with frequency resolution of 0.125 [Hz], in the range of 10-110 [Hz]. Figure 5.1 shows a representation of the system.

The system is adequate for this kind of analysis because damage can easily (and accurately) be represented by a spring having less stiffness than the others. Damage quantification will correspond to the stiffness reduction rate, and the localization will be the position of the "damaged" element, being one element for each spring. For example, a stiffness reduction of 10% in the third spring means the third element is damaged with $y_3 = 0.1$.

After training, models are tested with one experimental image, in which the 5th element suffers a stiffness reduction of 55% in its spring.

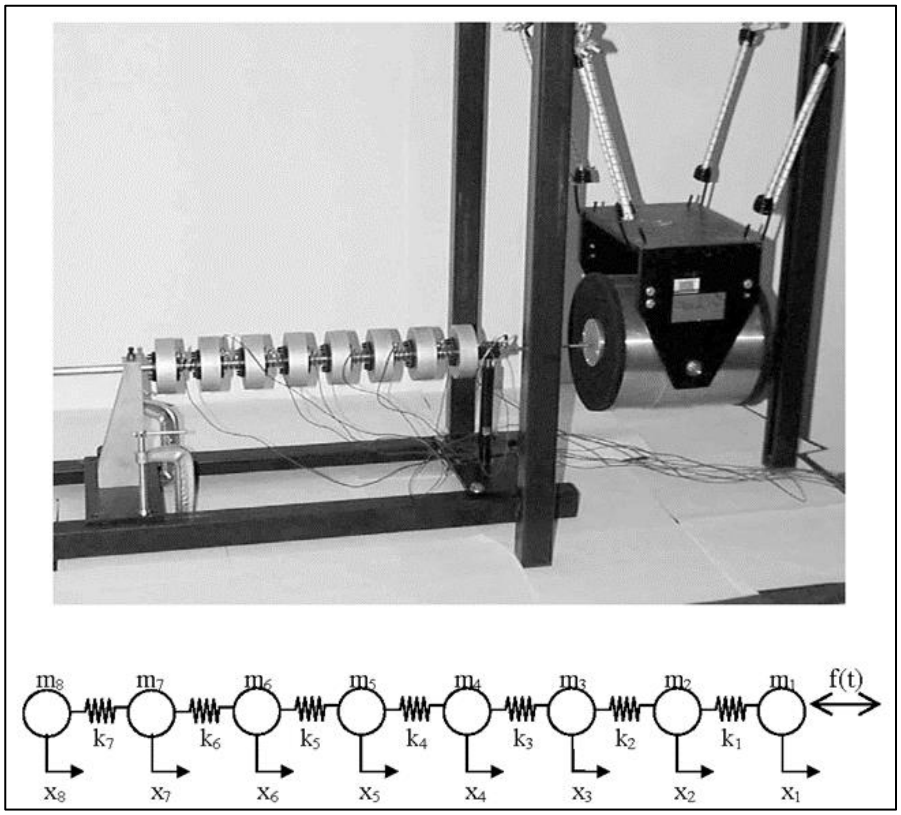


Figure 5.1 8 DOF spring-mass system

For this case study, four types of images are generated, according to the number of damaged elements.

Table 5.1 Types of training images

Damaged Elements	Number of Images
0	10,000
1	30,000
2	30,000
3	30,000

With these images, three models are proposed:

Table 5.2 Model configurations

Model	Dataset	Number of Images
Model 1	0 and 1 damaged elements	40,000
Model 2	0, 1, and 2 damaged elements	70,000
Model 3	0, 1, 2, and 3 damaged elements	100,000

5.1.1 Training Results, Dynamic Routing

For the first case study, performance metrics and training time are shown in table 5.3:

Table 5.3 Training performance metrics for capsule networks models with dynamic routing, first case study

Model	MSE	DME	FAE	Time per epoch [s]
1_DR	0,045 %	0,37 %	22,8 %	22,6
2_DR	0,091 %	0,94 %	16,2 %	49,0
3_DR	0,209 %	1,58 %	9,77 %	101,4

5.1.2 Experimental Results: Dynamic Routing

Model 1

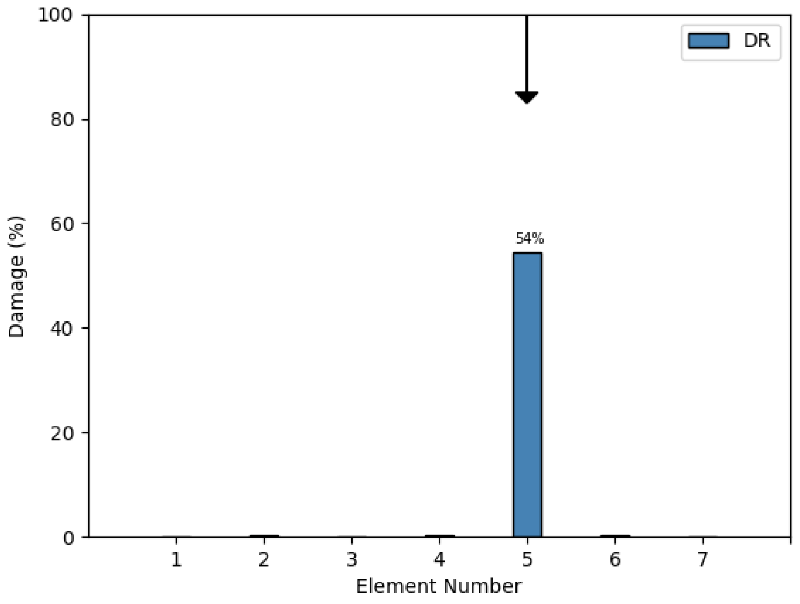


Figure 5.2 Experimental test with model 1, dynamic routing

Model 2:

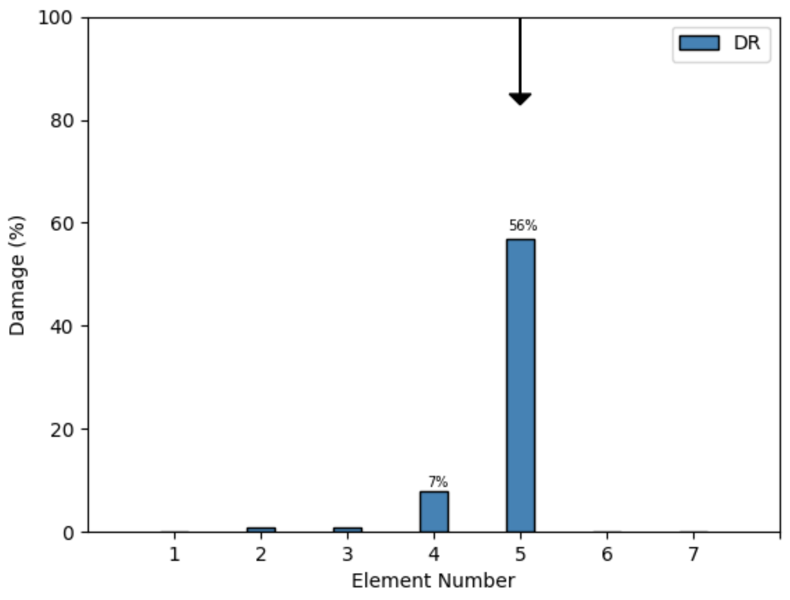


Figure 5.3 Experimental test with model 2, dynamic routing

Model 3:

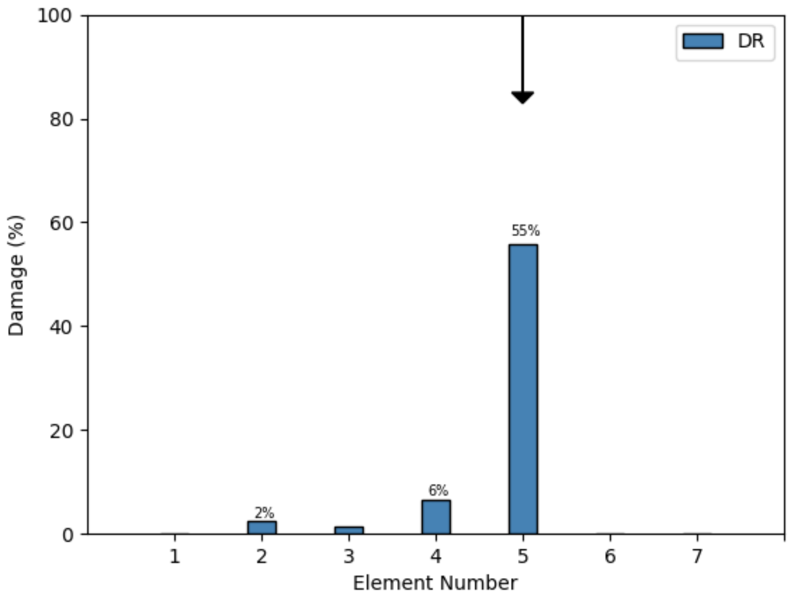


Figure 5.4 Experimental test with model 3, dynamic routing

5.1.3 Training Results: EM Routing

Table 5.4 Training performance metrics for capsule networks models with EM routing, first case study

Model	MSE	DME	FAE	Time per epoch [s]
1_EM	0,067 %	0,33 %	48,92 %	46,1
2_EM	0,17 %	1,36 %	18,83 %	81,7
3_EM	0,25 %	1,46 %	16,45 %	116,8

5.1.4 Experimental Results: EM Routing

Model 1:

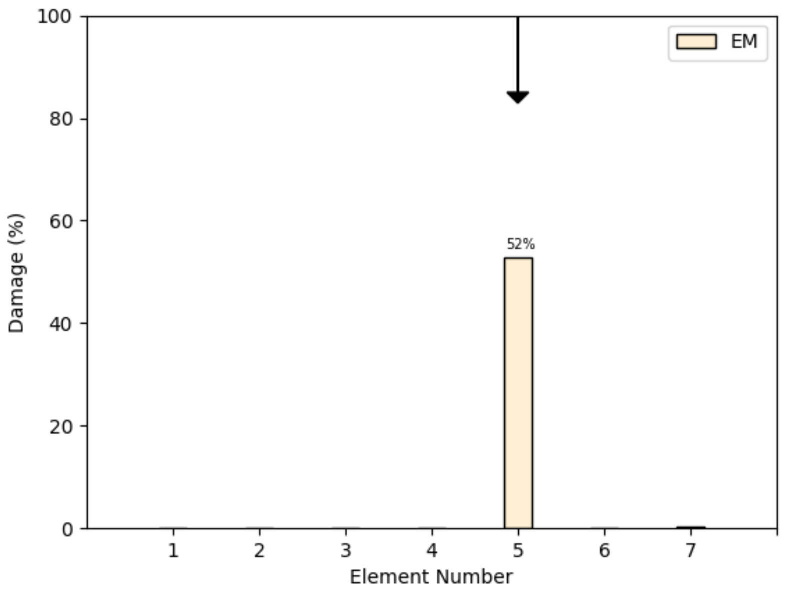


Figure 5.5 Experimental test with model 1, EM routing

Model 2:

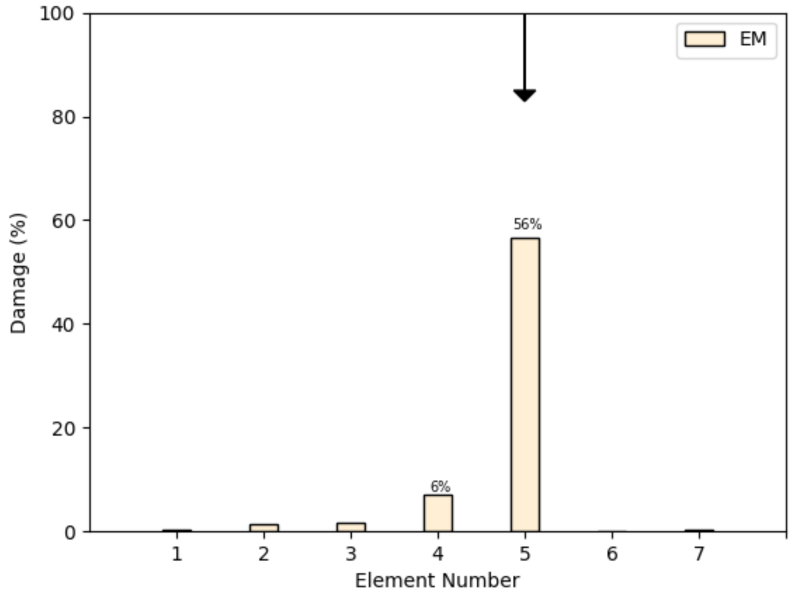


Figure 5.6 Experimental test with model 2, EM routing

Model 3:

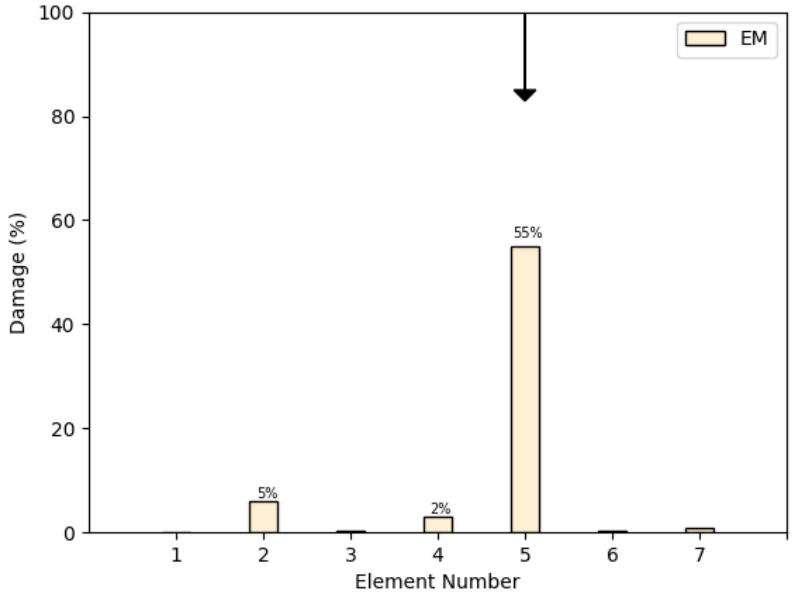


Figure 5.7 Experimental test with model 3, EM routing

5.1.5 Comparison Between Models

In figures 5.8 to 5.10, a comparison between DR, EM and CNN is shown.

Model 1:

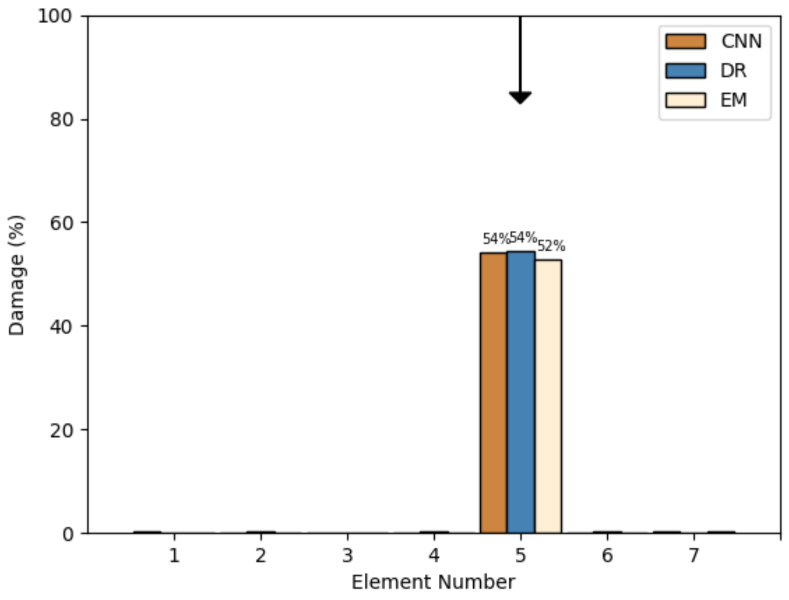


Figure 5.8 Comparison between architectures, model 1

Model 2

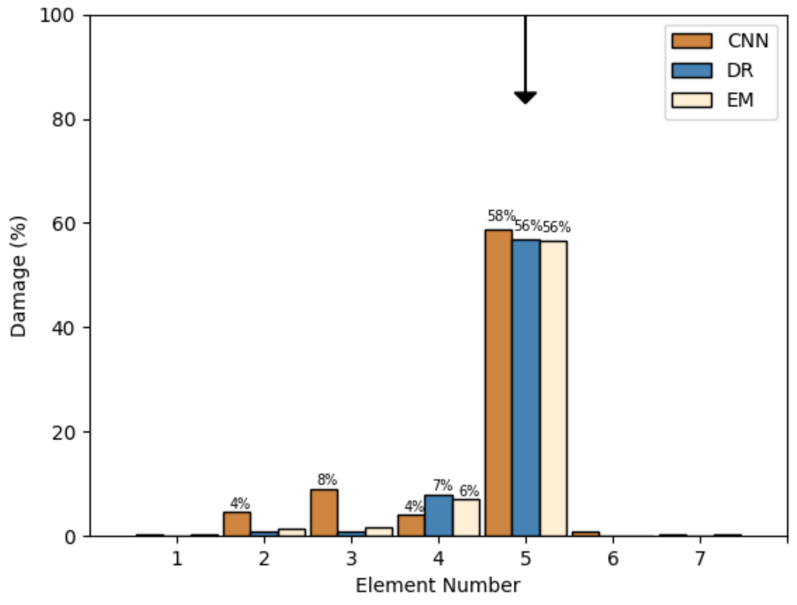


Figure 5.9 Comparison between architectures, model 2

Model 3:

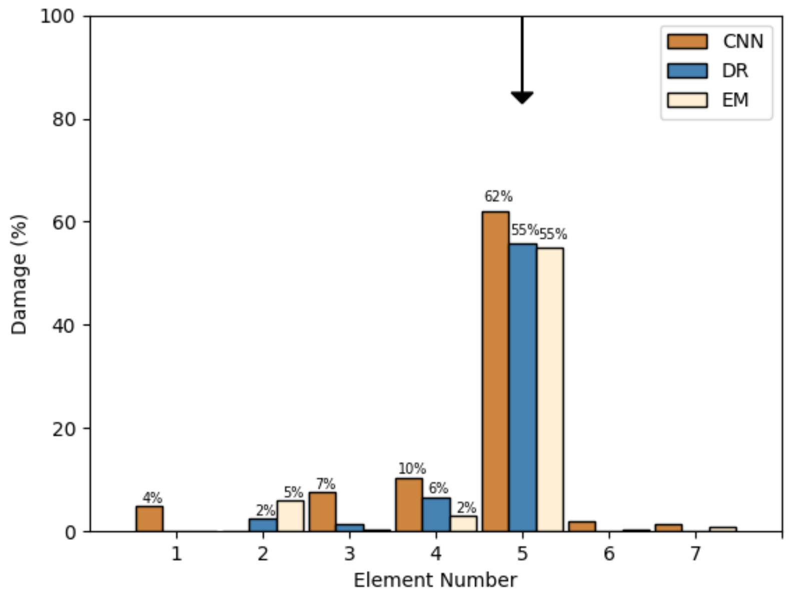


Figure 5.10 Comparison between architectures, model 3

5.2 Case Study 2: Structural Beam

The second case is a beam with damaged elements. Damage is generated by saw cuts at different locations of the beam. Just like the first case, a shaker is connected to one end of the beam to generate the 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 calculate transmissibilities. Data is acquired with frequency resolution of 1 [Hz], in the range of 1-2000 [Hz]. To build the FE model, unidimensional beam elements are used, with two nodes per element and two degrees of freedom per node. The beam is divided into 20 elements of 5 [cm] each. Figures 5.11, 5.12 and 5.13 show the setup, a diagram of the beam's elements, and cut examples for the beam.

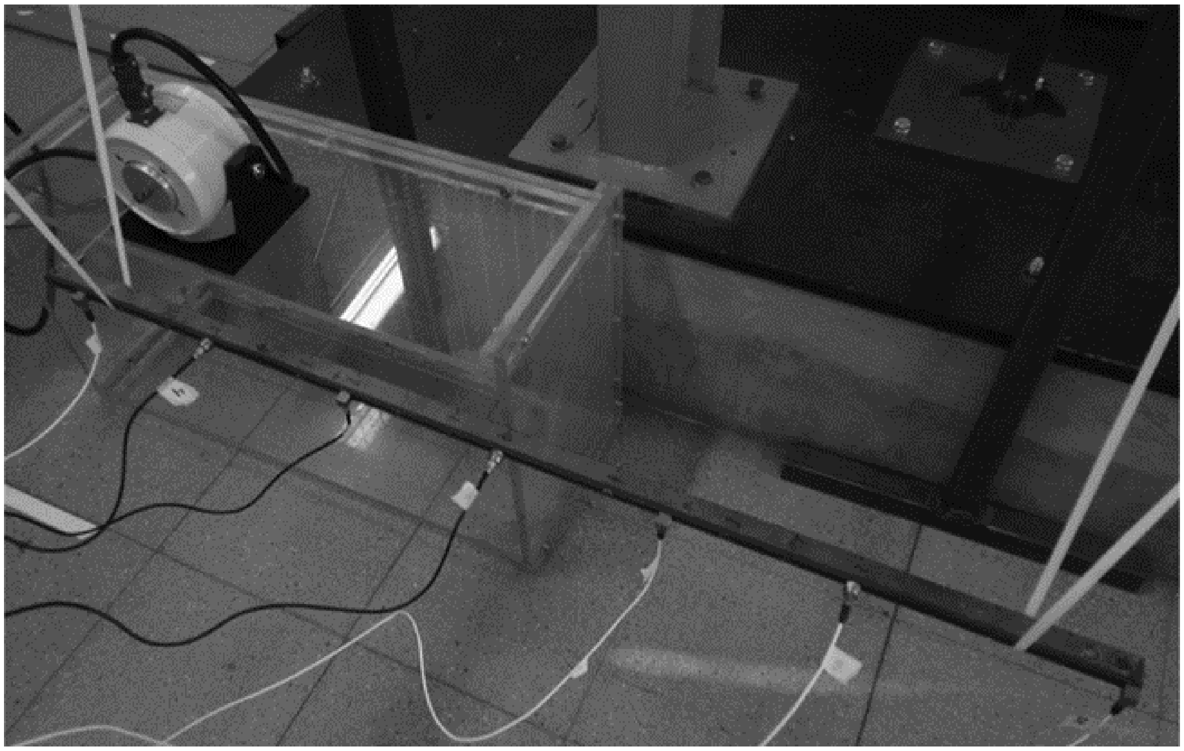


Figure 5.11 Beam setup

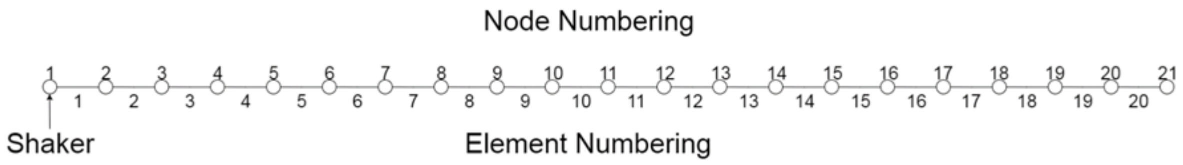


Figure 5.12 Beam element numbering

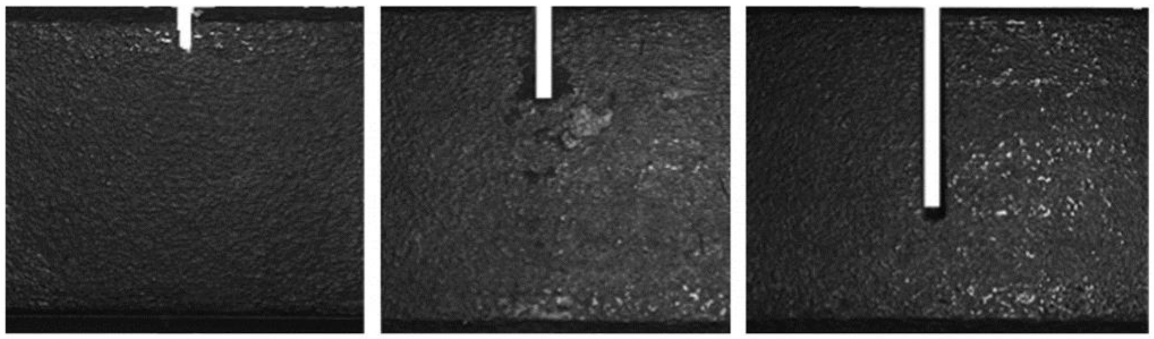


Figure 5.13 Beam cut examples

After training, models are tested with four experimental cases, which are described in table 5.5:

Table 5.5 Damage scenarios for beam experimental cases

Damage Scenario	**Number of cuts**	**Distance from the left side [mm]**	**Damaged Element**	**Cut depth [mm]**
1	1	313	7	7
2	1	637	13	9
3	2	361	8	8
		812	17	15
4	3	363	8	13
		574	12	8
		696	14-15	6

While in the first case study, a stiffness reduction of a spring implies a damage in the same amount of the reduction, in this case the saw cuts inflict stiffness reduction and therefore damage, but this damage's value is unknown. For example, a cut depth of 15

[mm] (of a total depth of 25 [mm]) doesn't necessarily mean the element suffers a 60% stiffness reduction.

For this case study, the configuration in terms of images and models is the same as the one used for the first case study (see tables 5.1 and 5.2)

5.2.1 Training Results, Dynamic Routing

For the structural beam case, performance metrics are presented in table 5.6:

Table 5.6 Training performance metrics for capsule networks models with dynamic routing, second case study

Model	MSE	DME	FAE	Time per epoch [s]
1_DR	0,038%	1,17%	65,30%	22,3
2_DR	0,415%	5,33%	55,28%	30,1
3_DR	0,783%	6,17%	44,70%	132,2

In figures 5.14, 5.15, and 5.16, DME and FAE results are shown, for each of the 3 trained models.

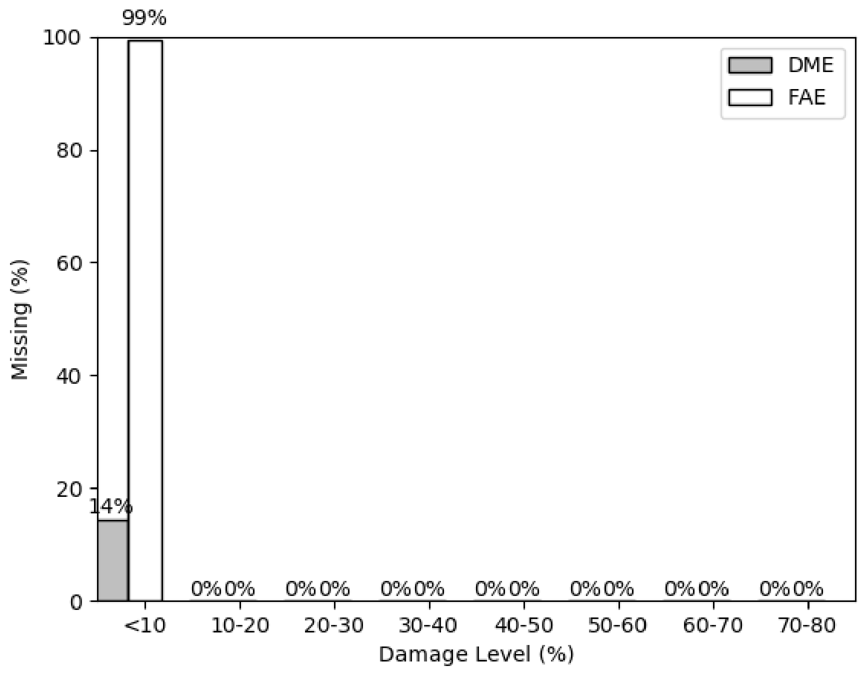


Figure 5.14 DME and FAE versus damage level for structural damage using model 1

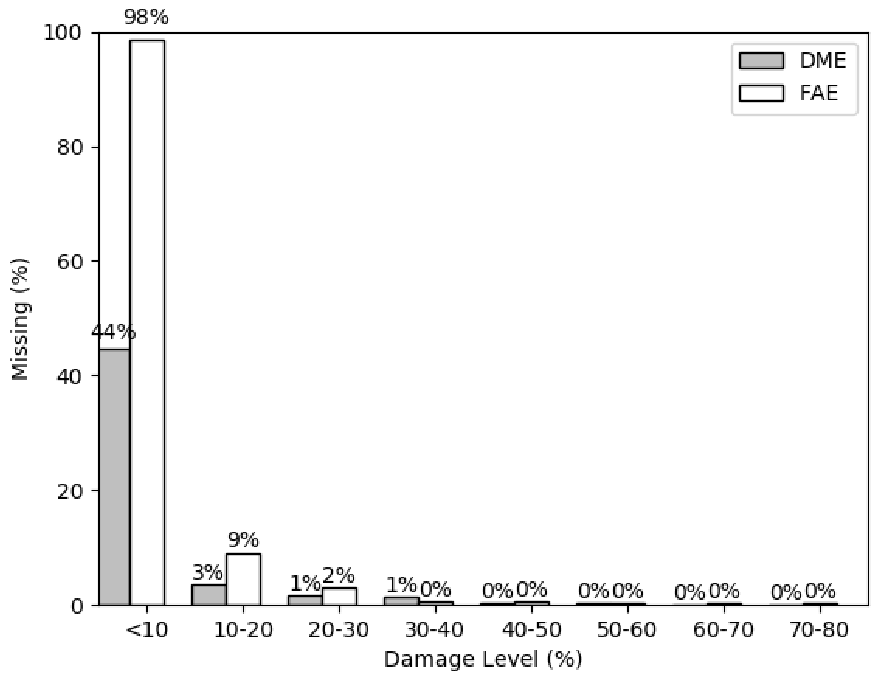


Figure 5.15 DME and FAE versus damage level for structural damage using model 2

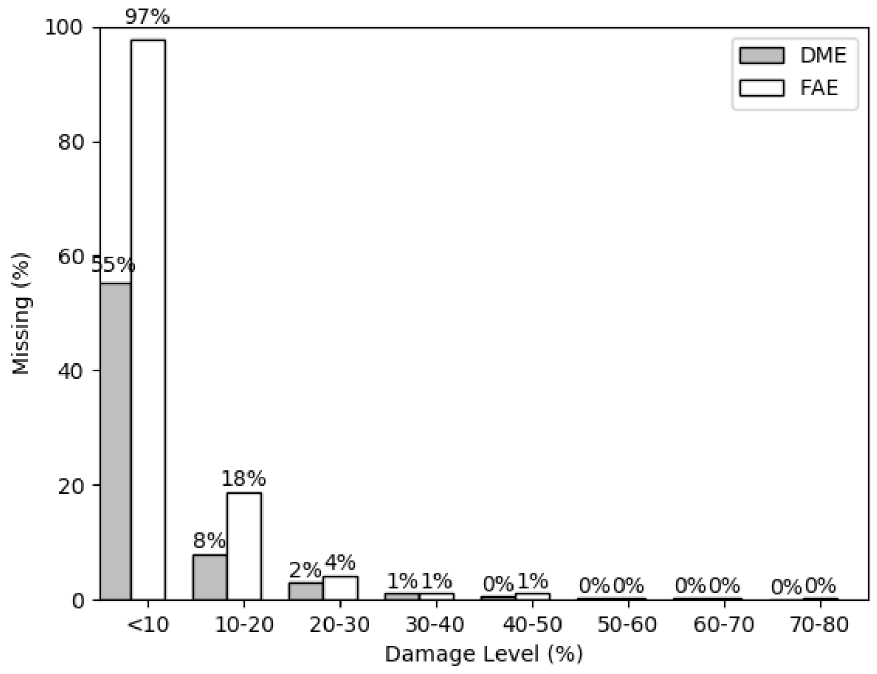


Figure 5.16 DME and FAE versus damage level for structural damage using model 3

5.2.2 Experimental Results: Dynamic Routing

Trained models are tested on experimental images. The results are shown in the following images:

Model 1:

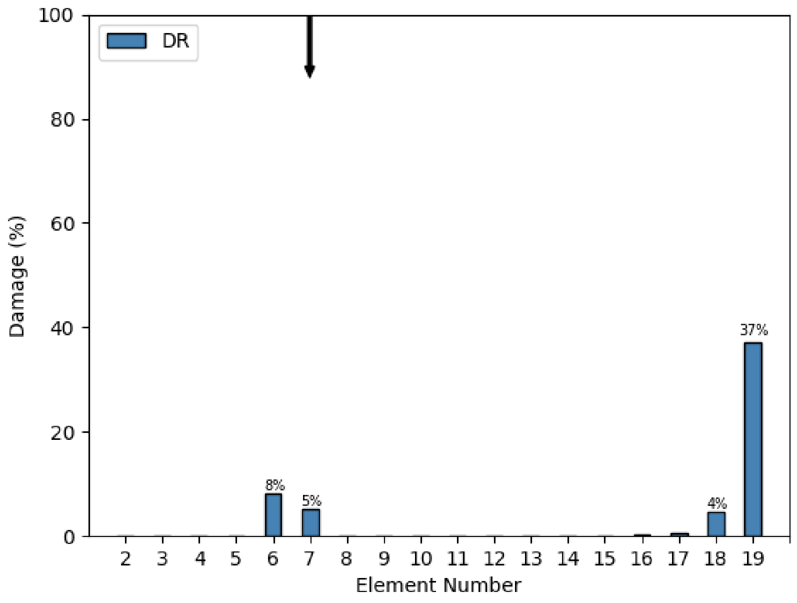


Figure 5.17 Experimental test 1 with model 1, dynamic routing

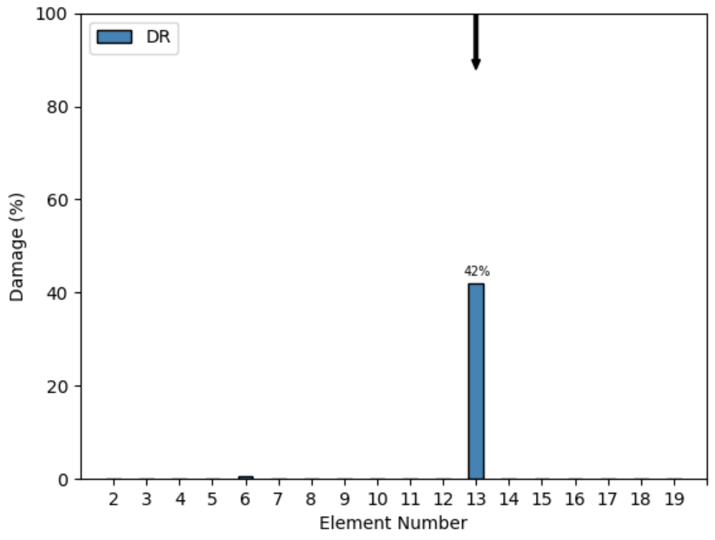


Figure 5.18 Experimental test 2 with model 1, dynamic routing

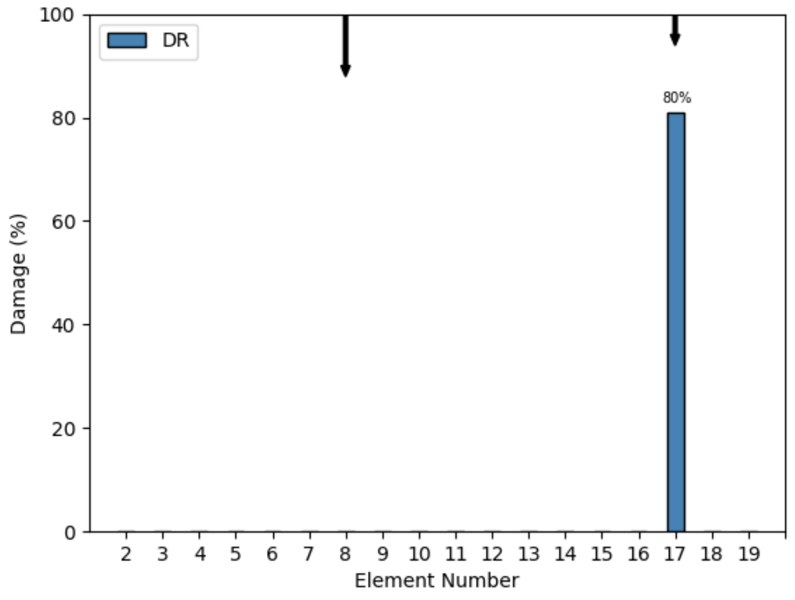


Figure 5.19 Experimental test 3 with model 1, dynamic routing

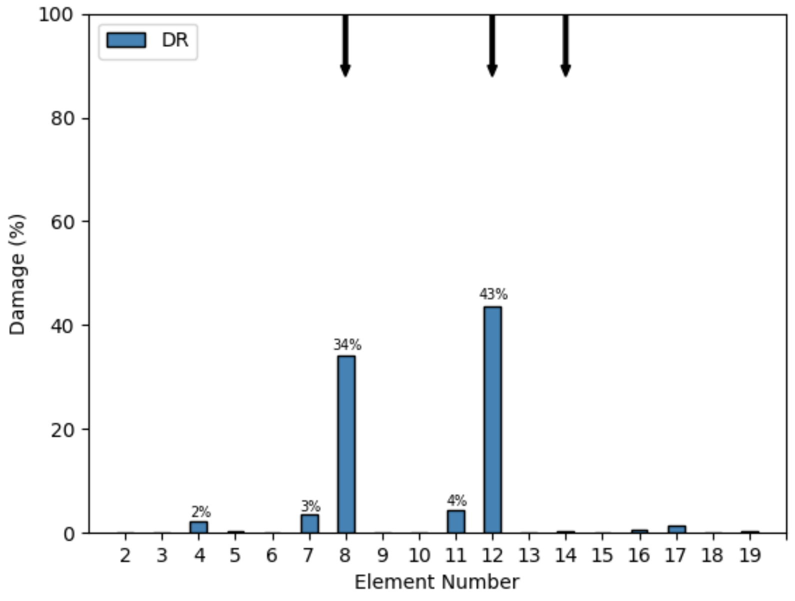


Figure 5.20 Experimental test 4 with model 1, dynamic routing

Model 2:

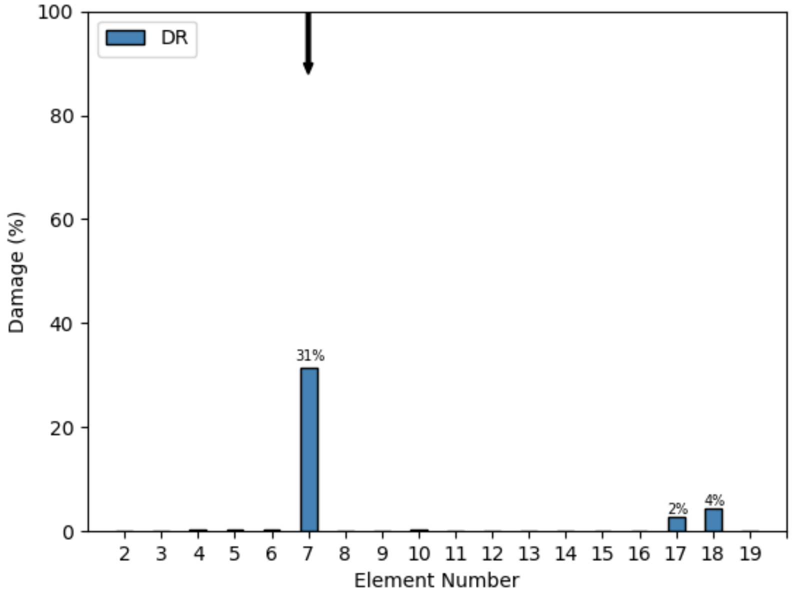


Figure 5.21 Experimental test 1 with model 2, dynamic routing

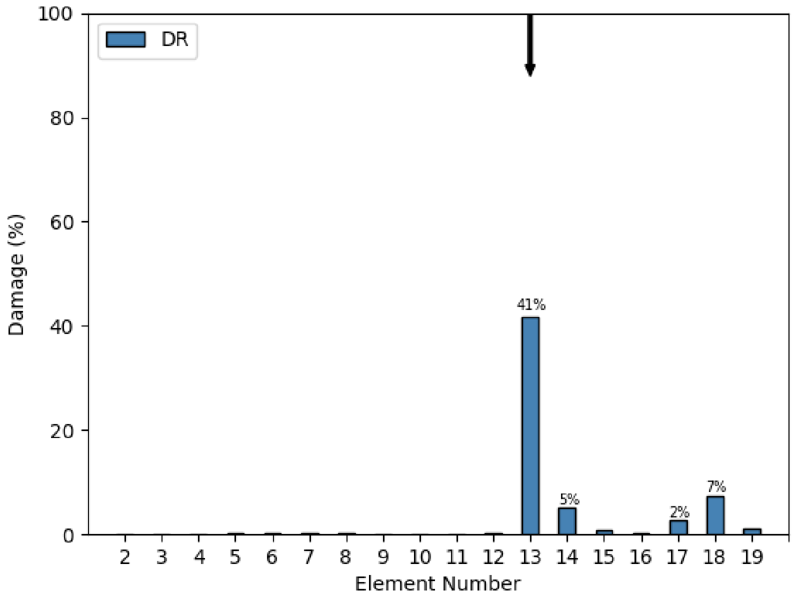


Figure 5.22 Experimental test 2 with model 2, dynamic routing

Model 3:

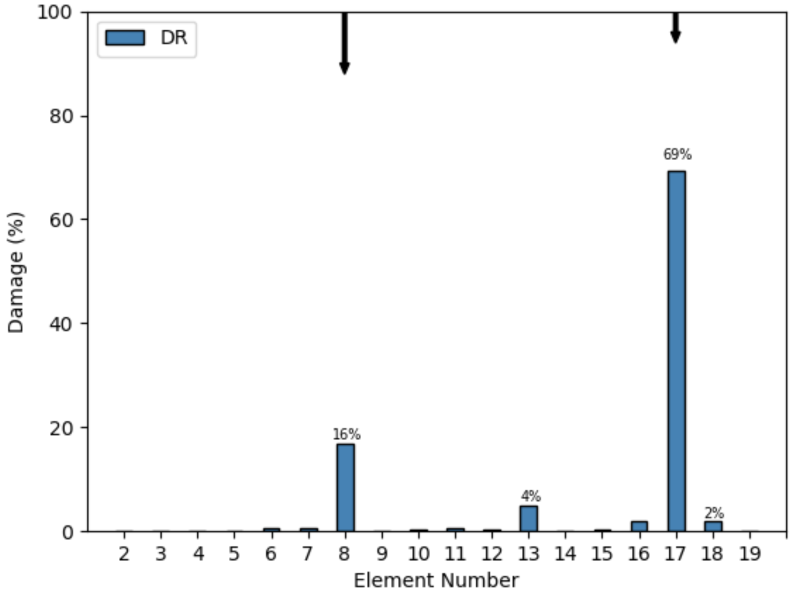


Figure 5.23 Experimental test 3 with model 2, dynamic routing

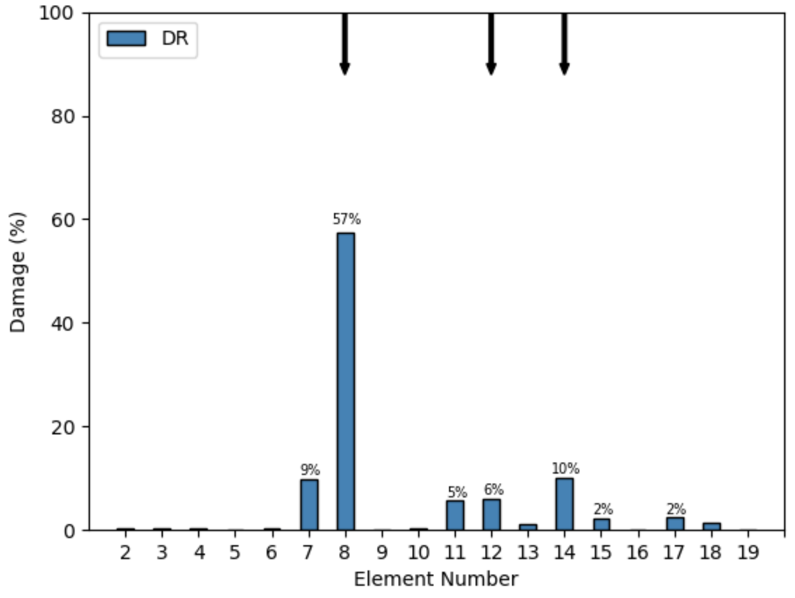


Figure 5.24Experimental test 4 with model 2, dynamic routing

Model 3:

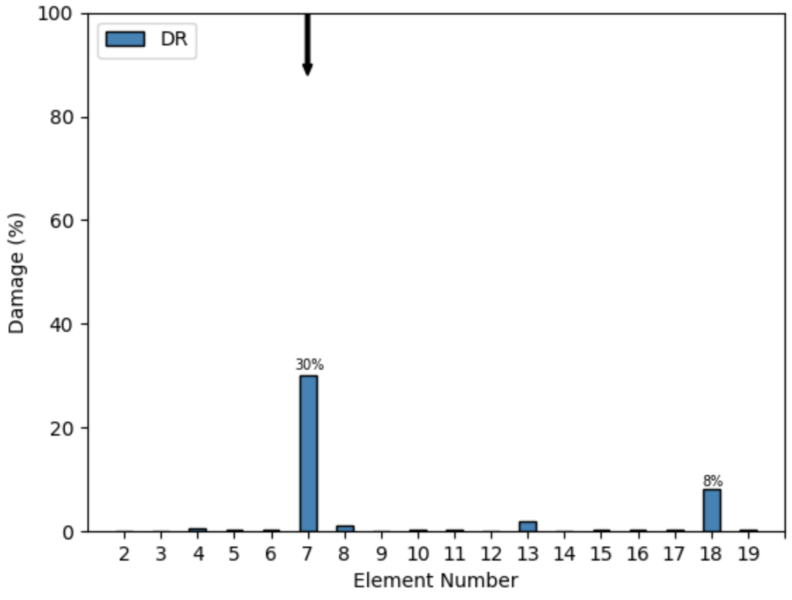


Figure 5.25 Experimental test 1 with model 3, dynamic routing

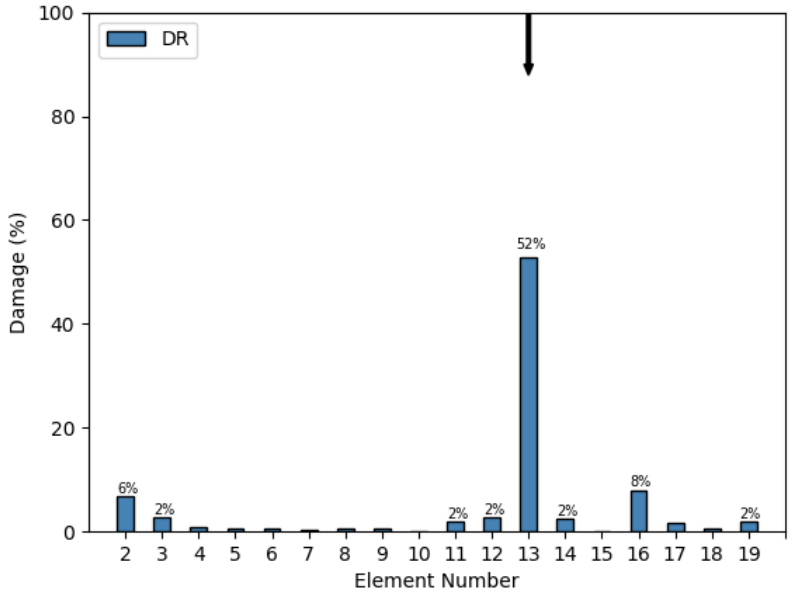


Figure 5.26 Experimental test 2 with model 3, dynamic routing

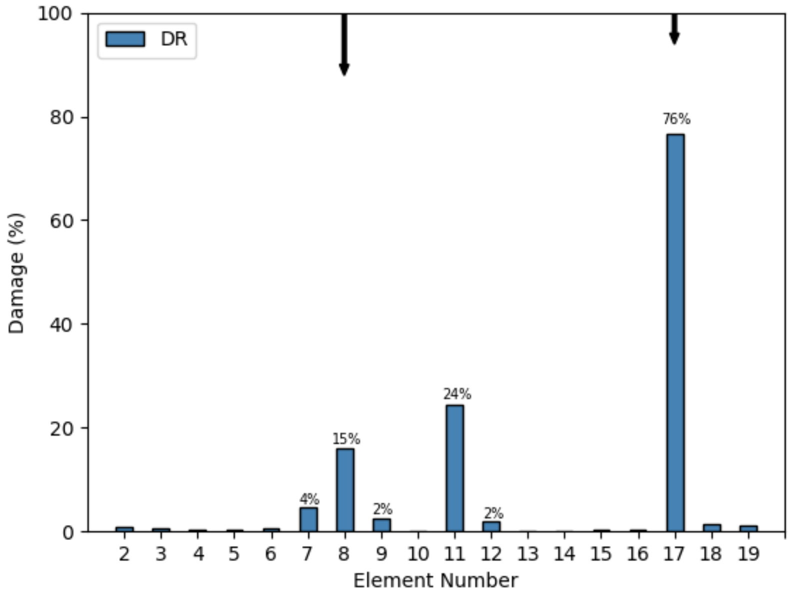


Figure 5.27 Experimental test 3 with model 3, dynamic routing

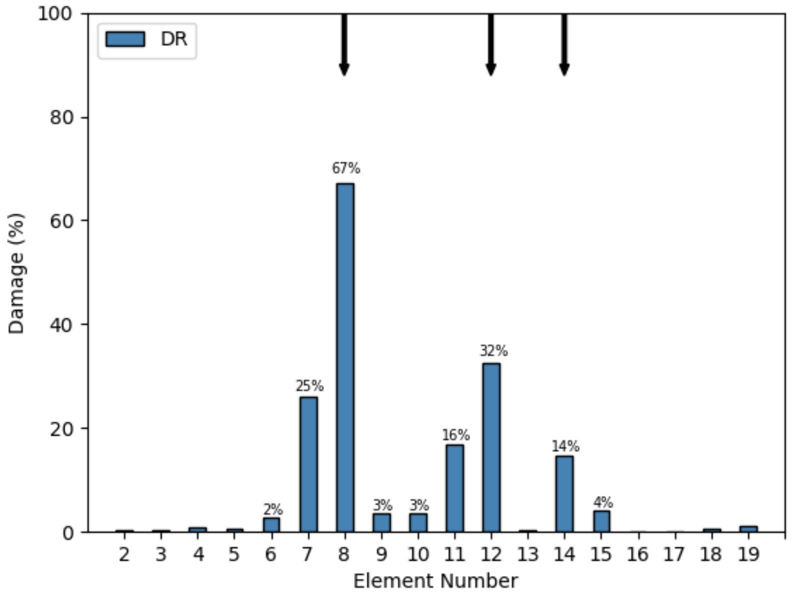


Figure 5.28 Experimental test 4 with model 3, dynamic routing

5.2.3 Training Results: EM Routing

Table 5.7 Training performance metrics for capsule networks models with EM routing, second case study

Model	MSE	DME	FAE	Time per epoch [s]
1_EM	0.132%	1.59%	77.43%	153.8
2_EM	0.464%	4.36%	52.08%	260.1
3_EM	0.948%	6.14%	45.24%	375.9

In figures 5.29, 5.30 and 5.31, DME and FAE results are shown, for each of the 3 trained models.

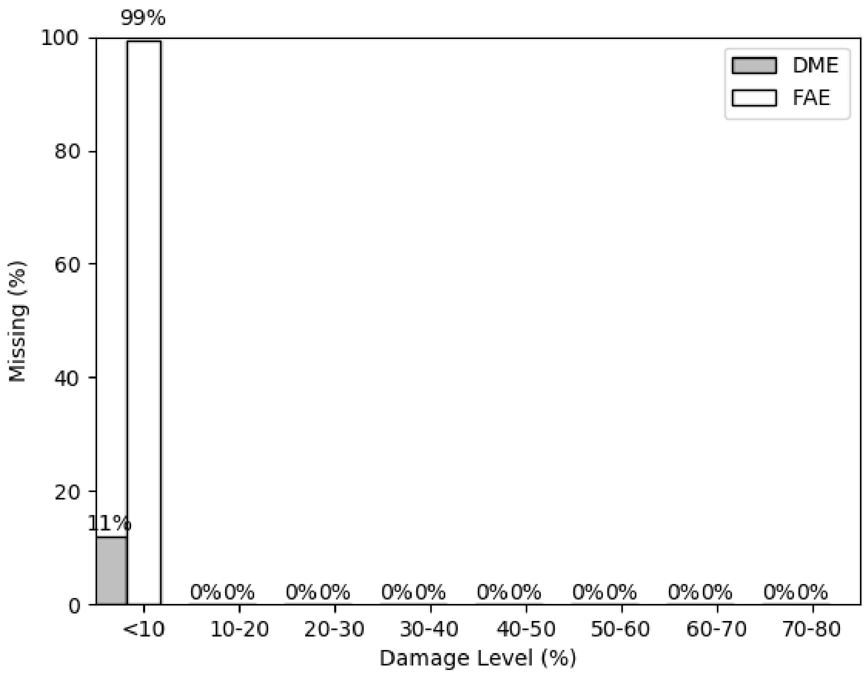


Figure 5.29 DME and FAE versus damage level for structural damage using model 1

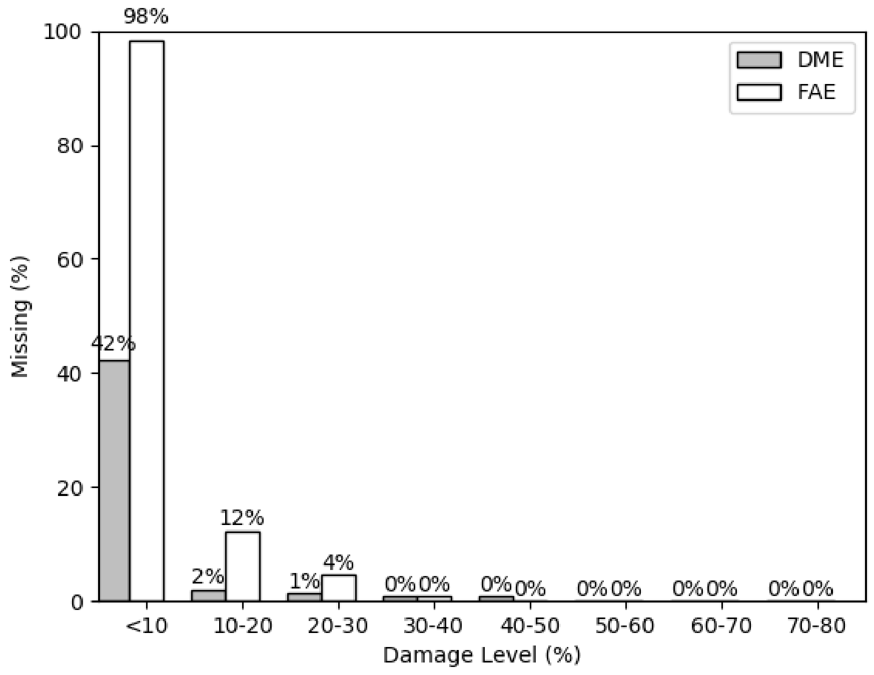


Figure 5.30 DME and FAE versus damage level for structural damage using model 2

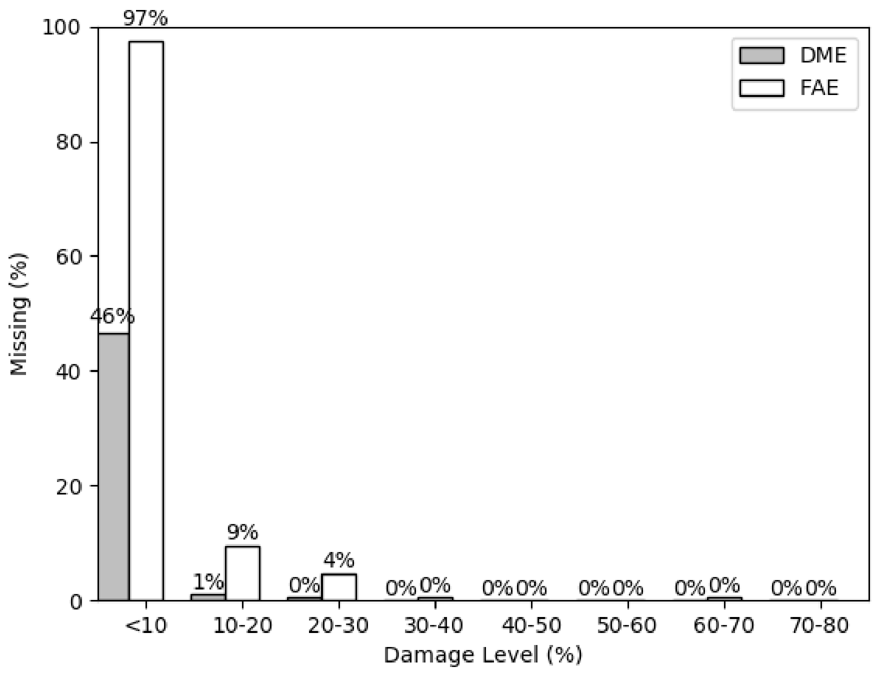


Figure 5.31 DME and FAE versus damage level for structural damage using model 3

5.2.4 Experimental Results: EM Routing

Trained models using CapsNets with EM routing are tested with experimental cases. Results are shown in the figures below.

Model 1:

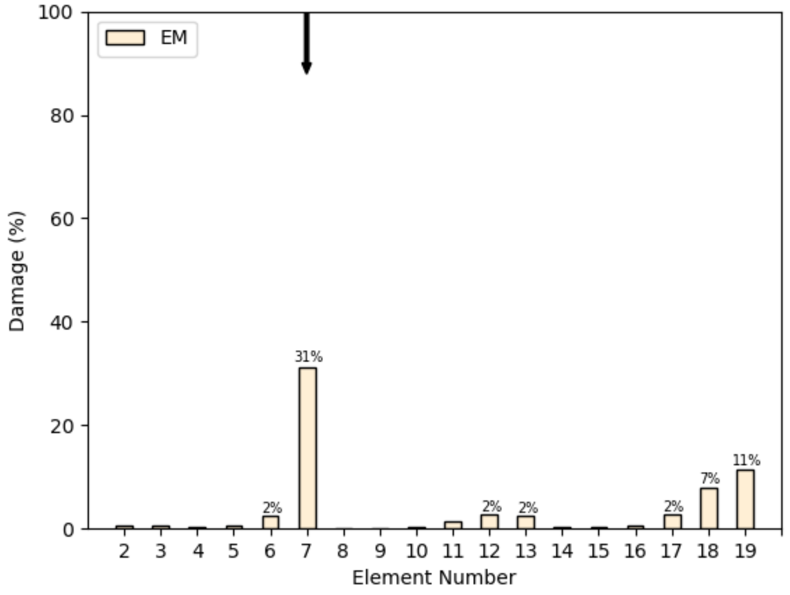


Figure 5.32 Experimental test 1 with model 1, EM routing

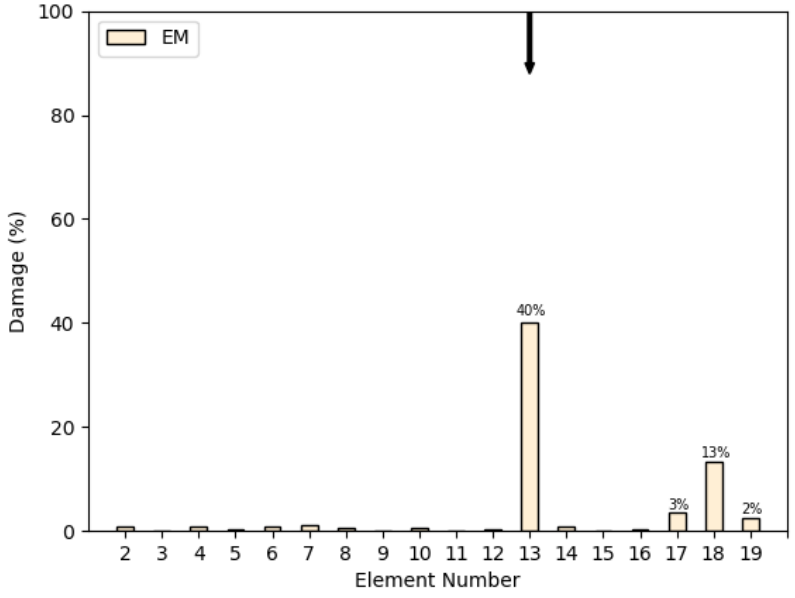


Figure 5.33 Experimental test 2 with model 1, EM routing

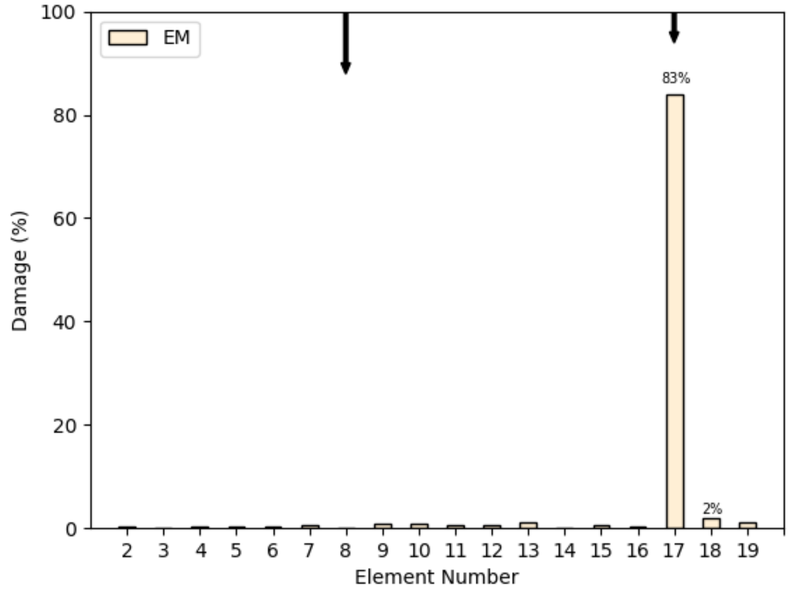


Figure 5.34 Experimental test 3 with model 1, EM routing

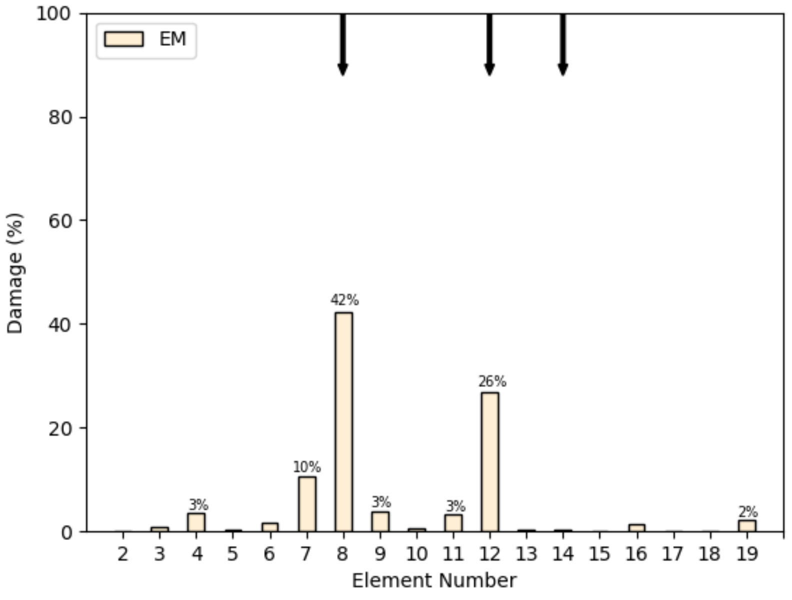


Figure 5.35 Experimental test 4 with model 1, EM routing

Model 2:

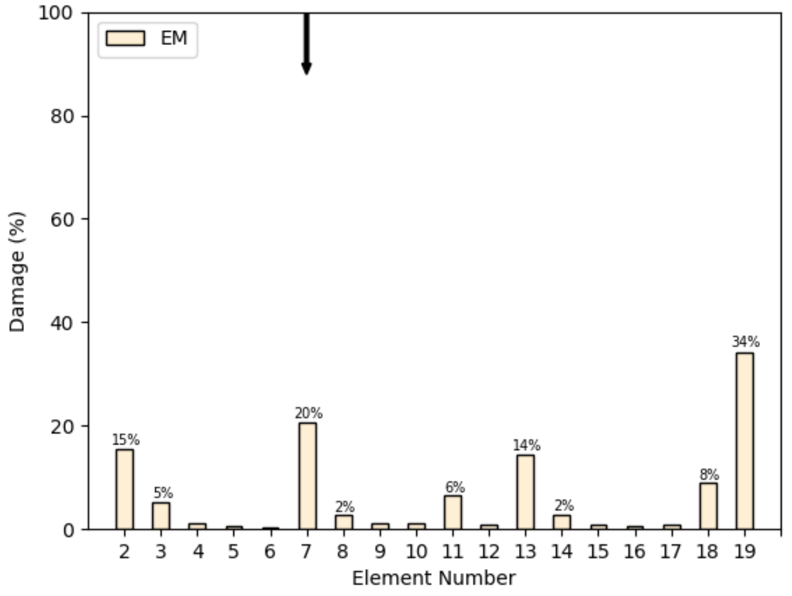


Figure 5.36 Experimental test 1 with model 2, EM routing

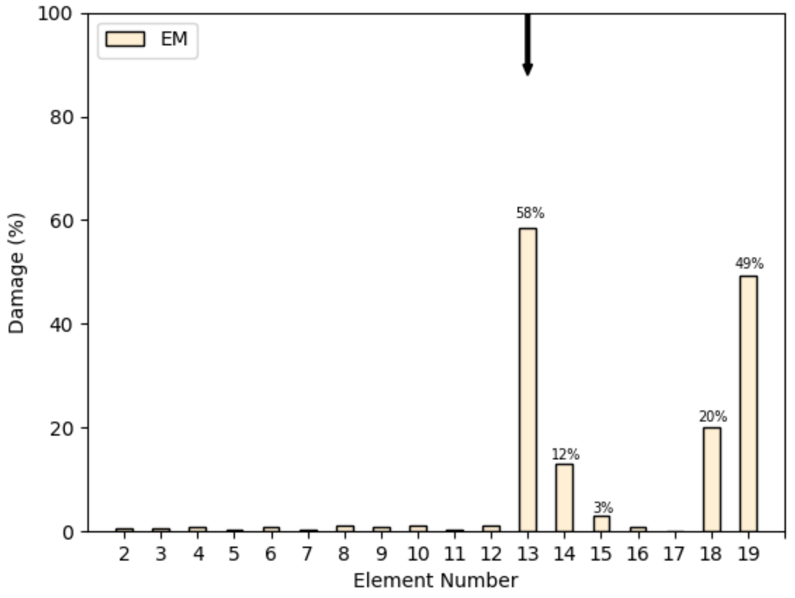


Figure 5.37 Experimental test 2 with model 2, EM routing

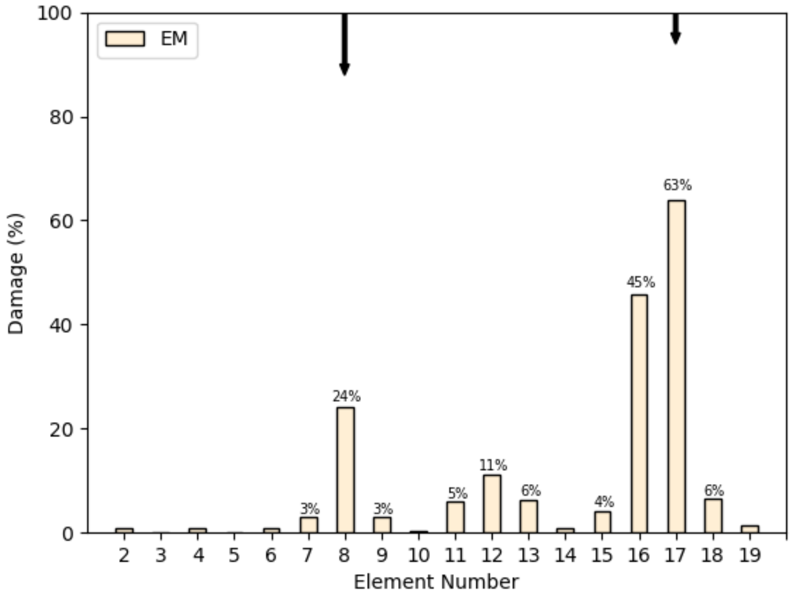


Figure 5.38 Experimental test 3 with model 2, EM routing

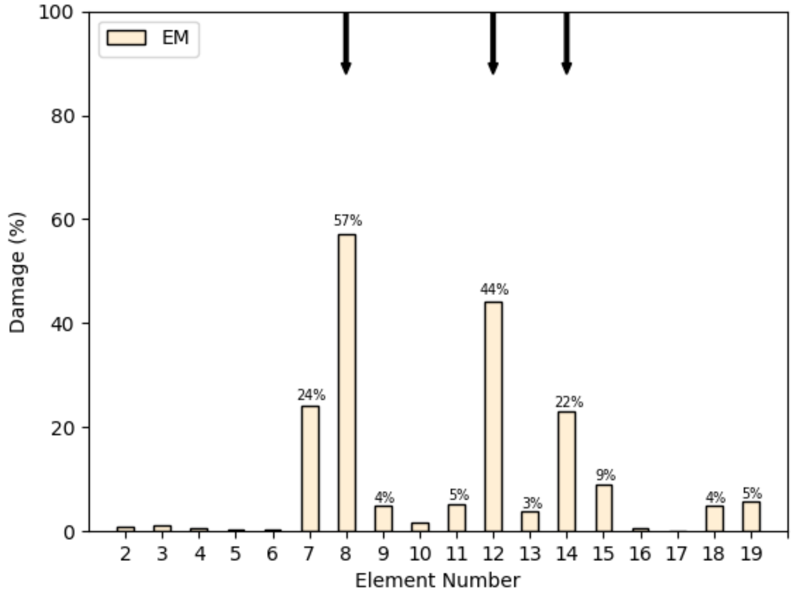


Figure 5.39 Experimental test 4 with model 2, EM routing

Model 3:

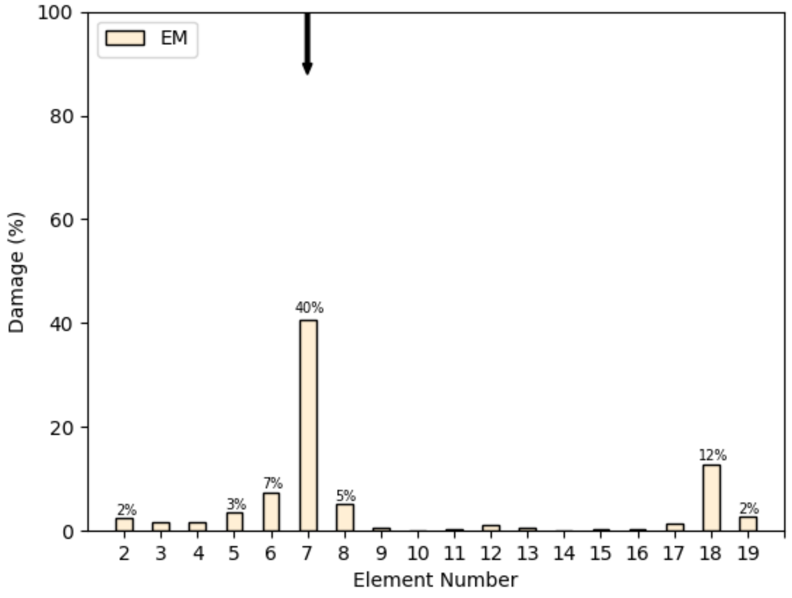


Figure 5.40 Experimental test 1 with model 3, EM routing

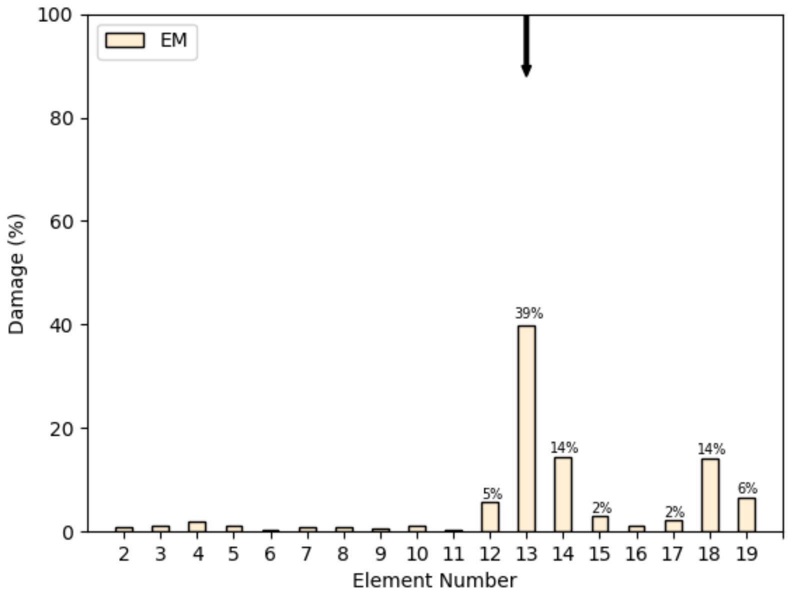


Figure 5.41 Experimental test 2 with model 3, EM routing

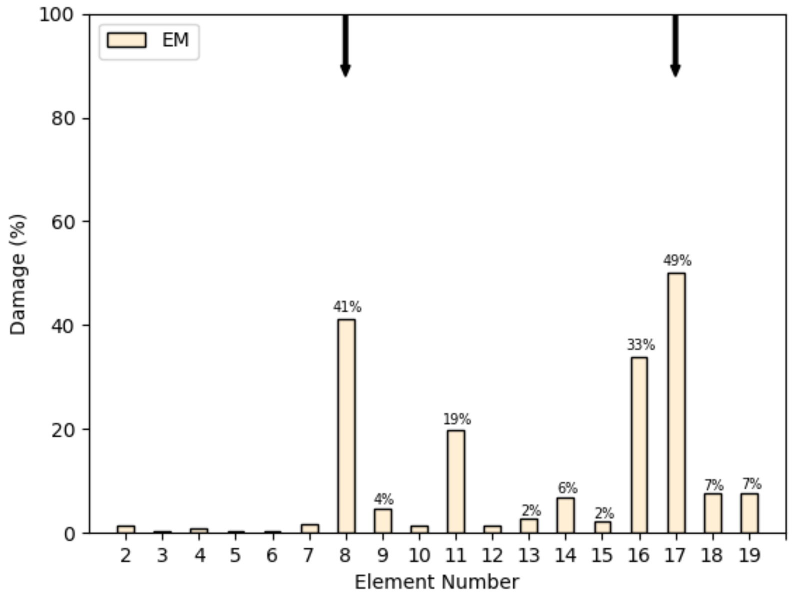


Figure 5.42 Experimental test 3 with model 3, EM routing

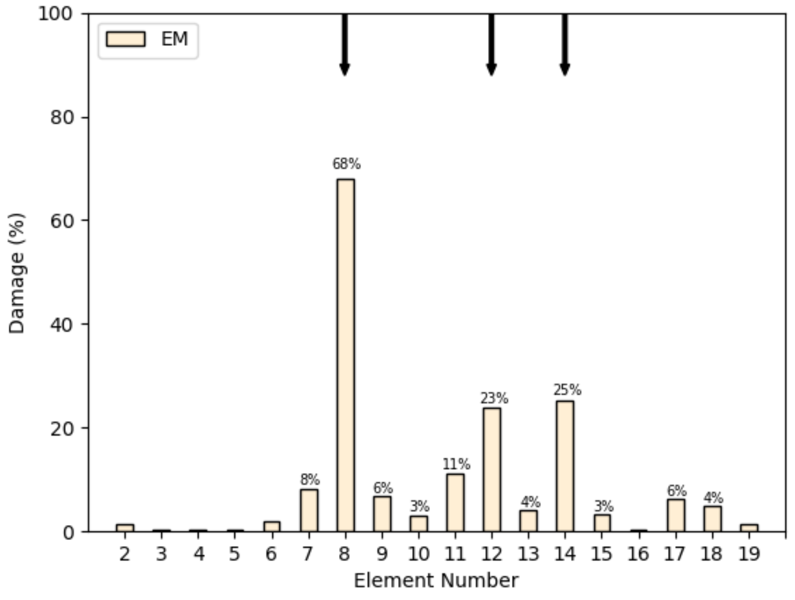


Figure 5.43 Experimental test 4 with model 3, EM routing

5.2.5 Comparison Between Models

In figures 5.44 to 5.55, a comparison between DR, EM and CNN is shown.

Model 1:

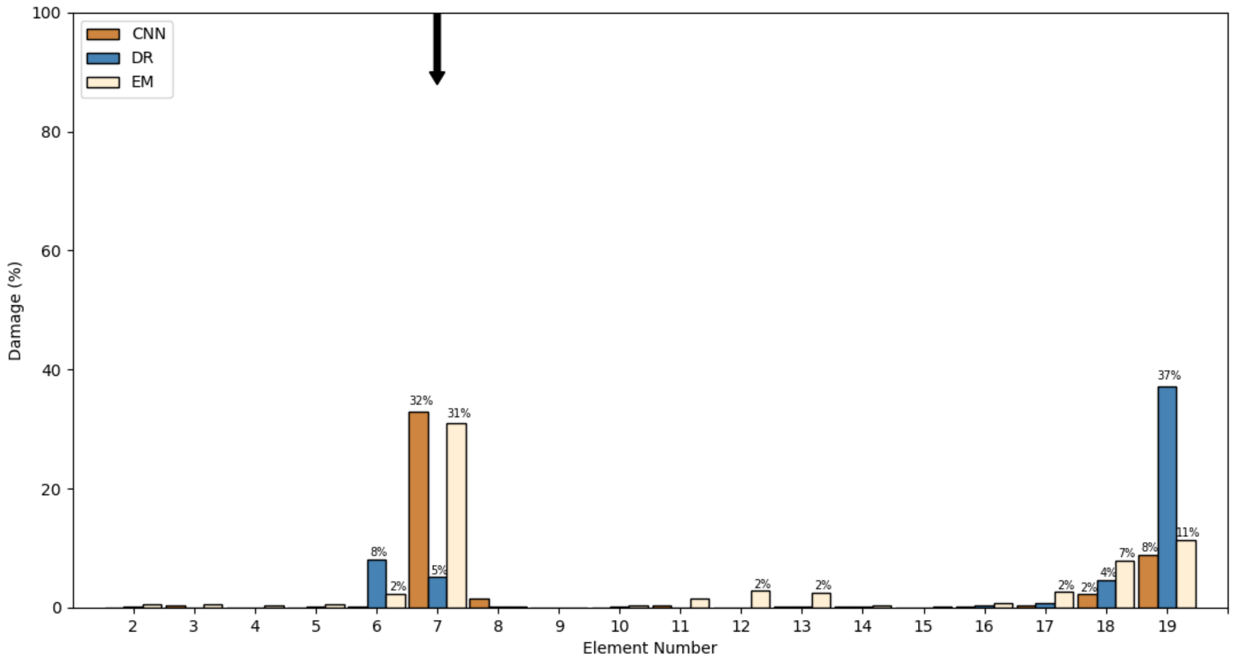


Figure 5.44 Comparison between models, test 1, model 1

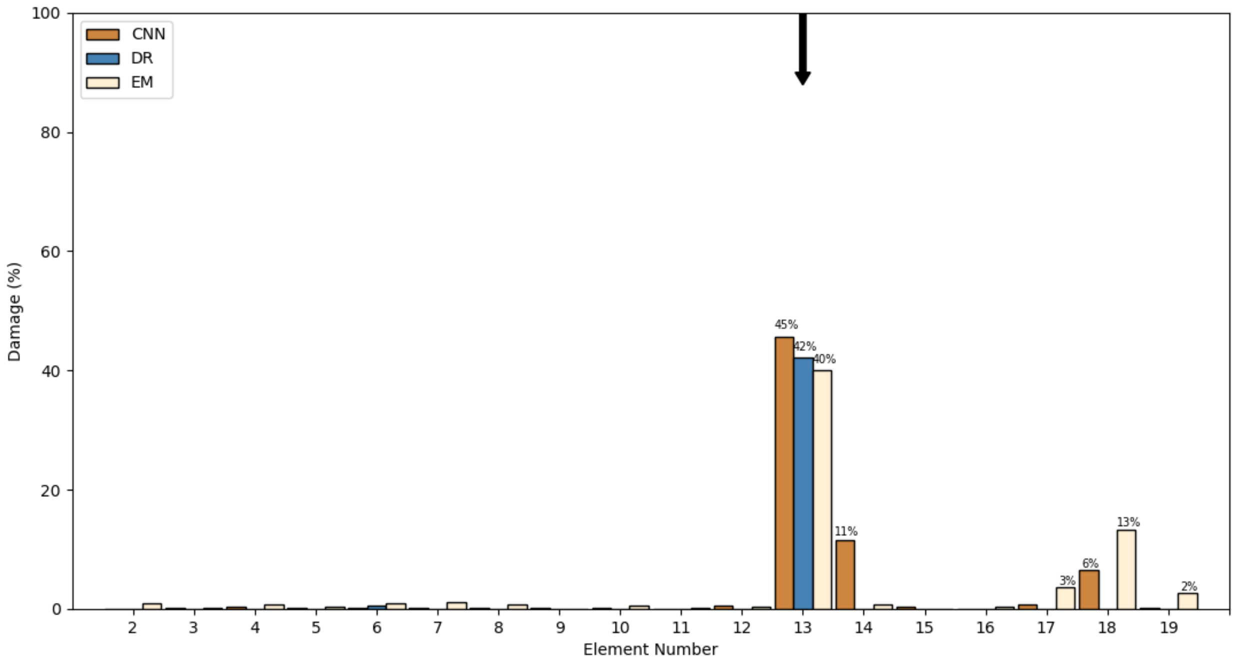


Figure 5.45 Comparison between models, test 2, model 1

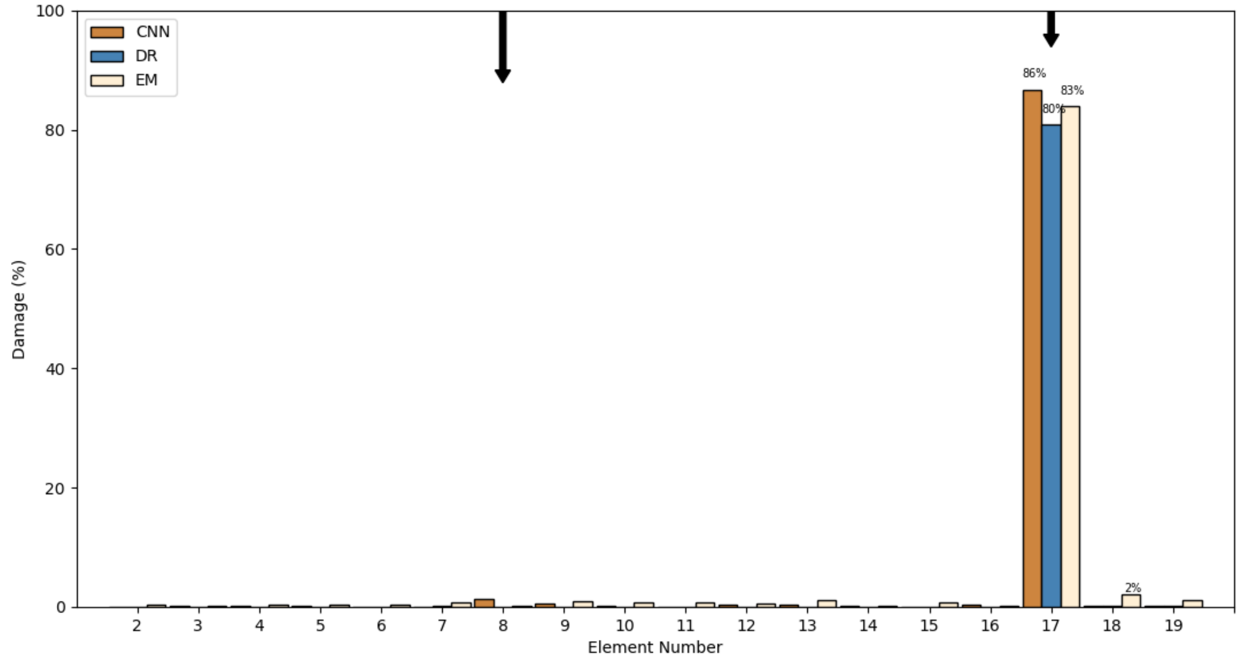


Figure 5.46 Comparison between models, test 3, model 1

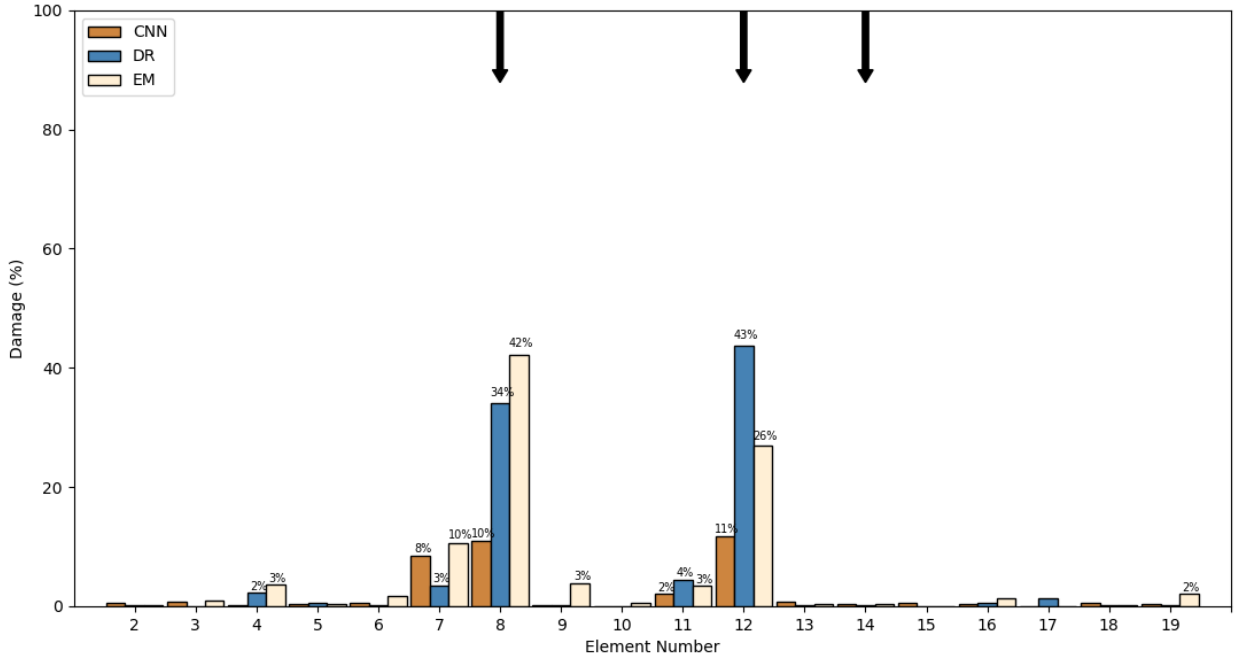


Figure 5.47 Comparison between models, test 4, model 1

Model 2:

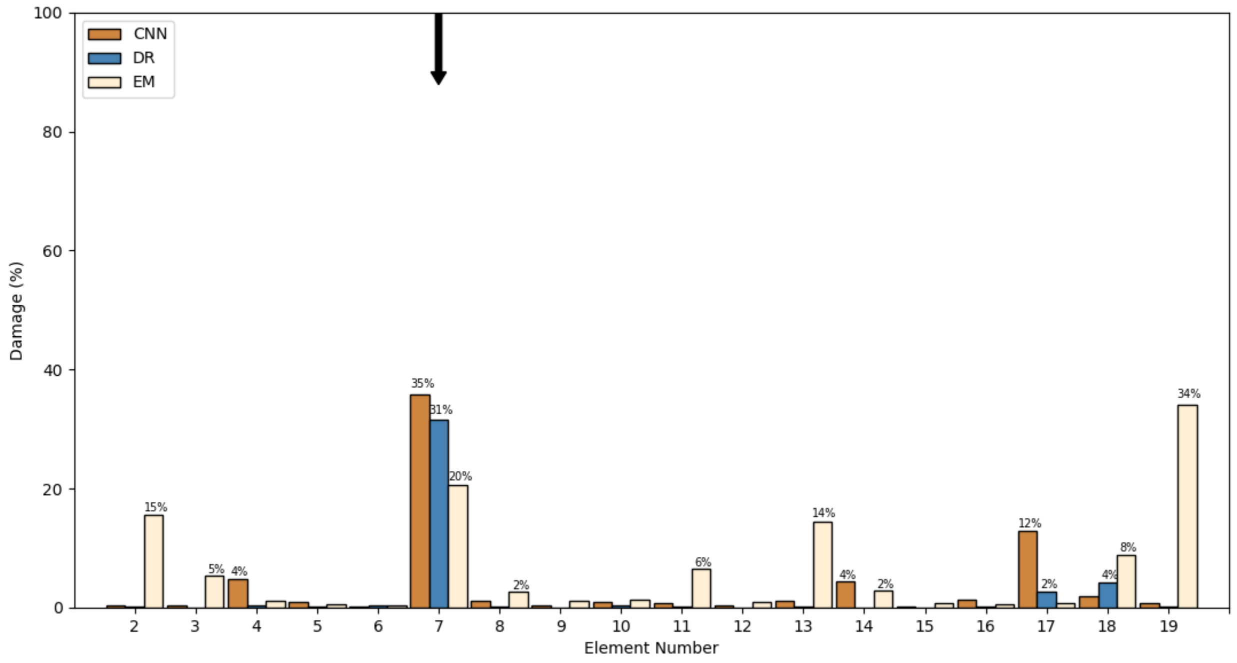


Figure 5.48 Comparison between models, test 1, model 2

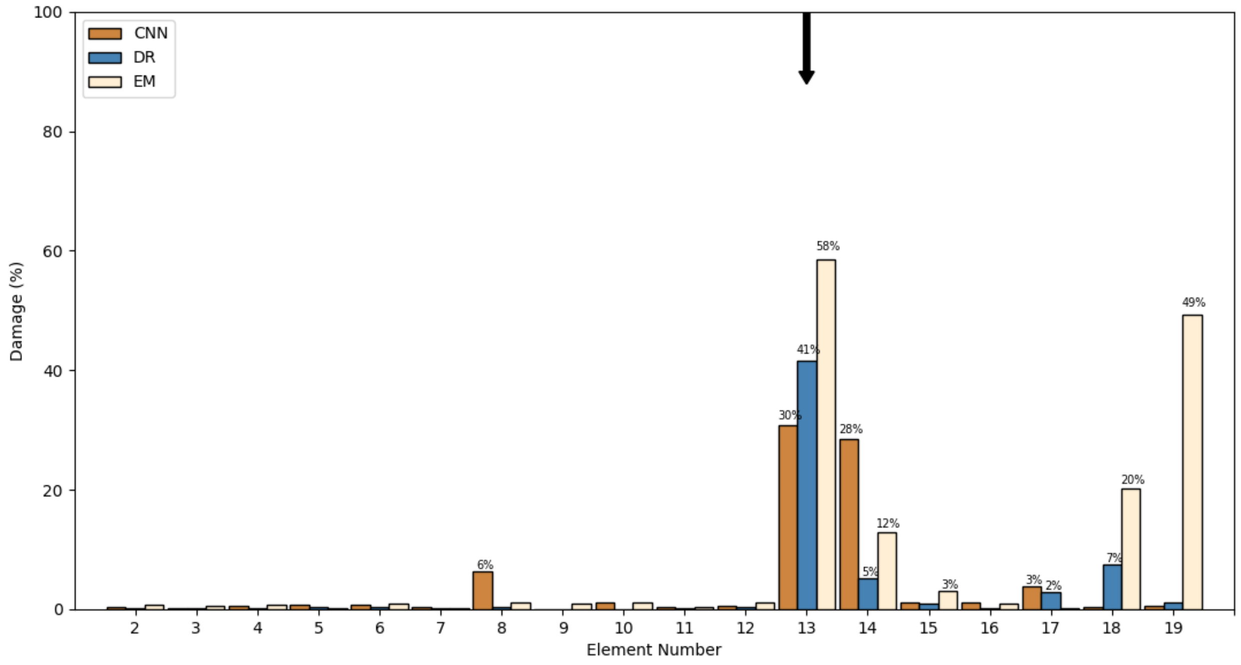


Figure 5.49 Comparison between models, test 2, model 2

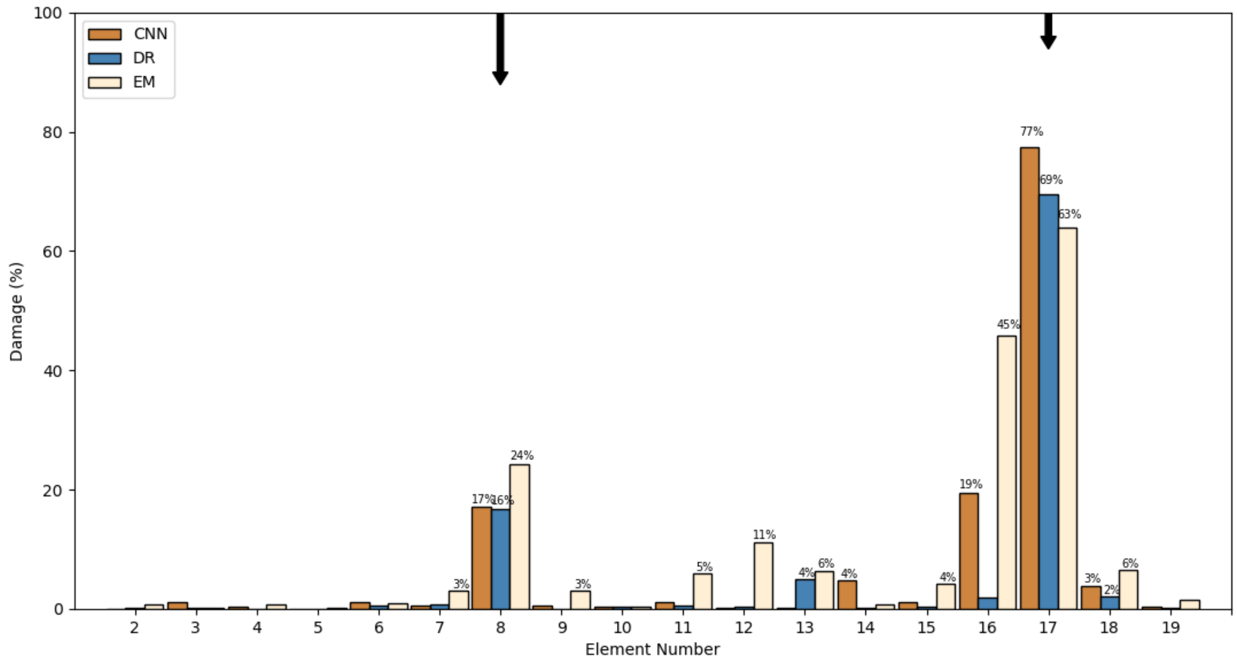


Figure 5.50 Comparison between models, test 3, model 2

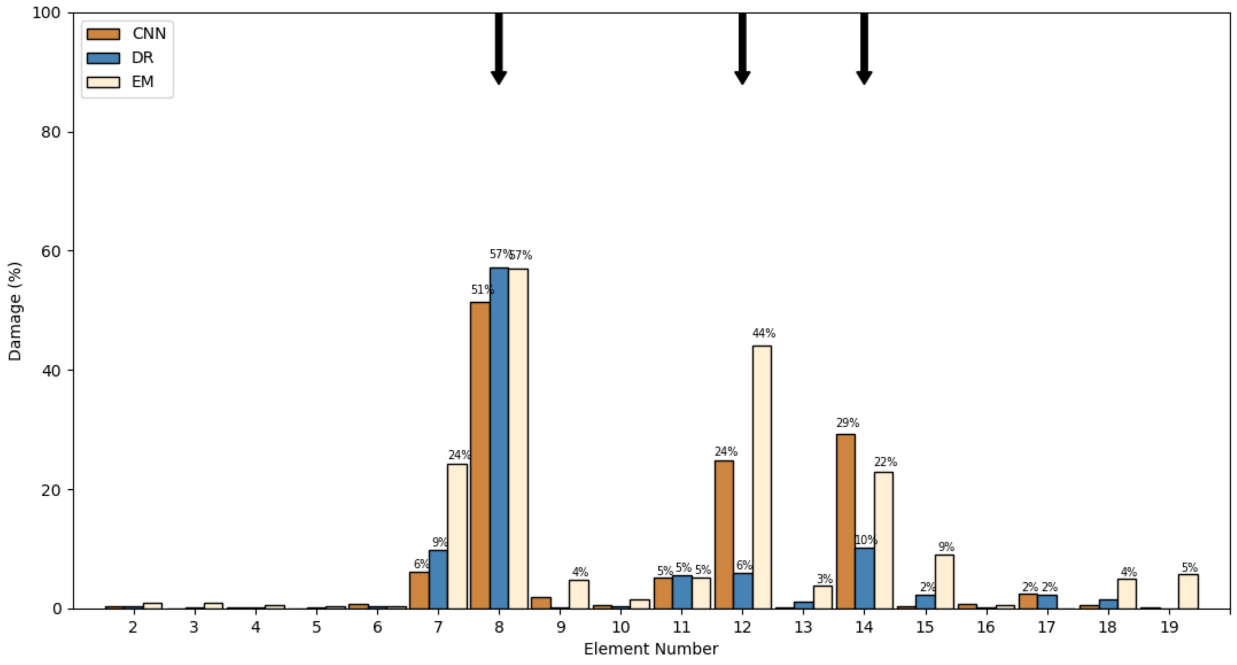


Figure 5.51 Comparison between models, test 4, model 2

Model 3:

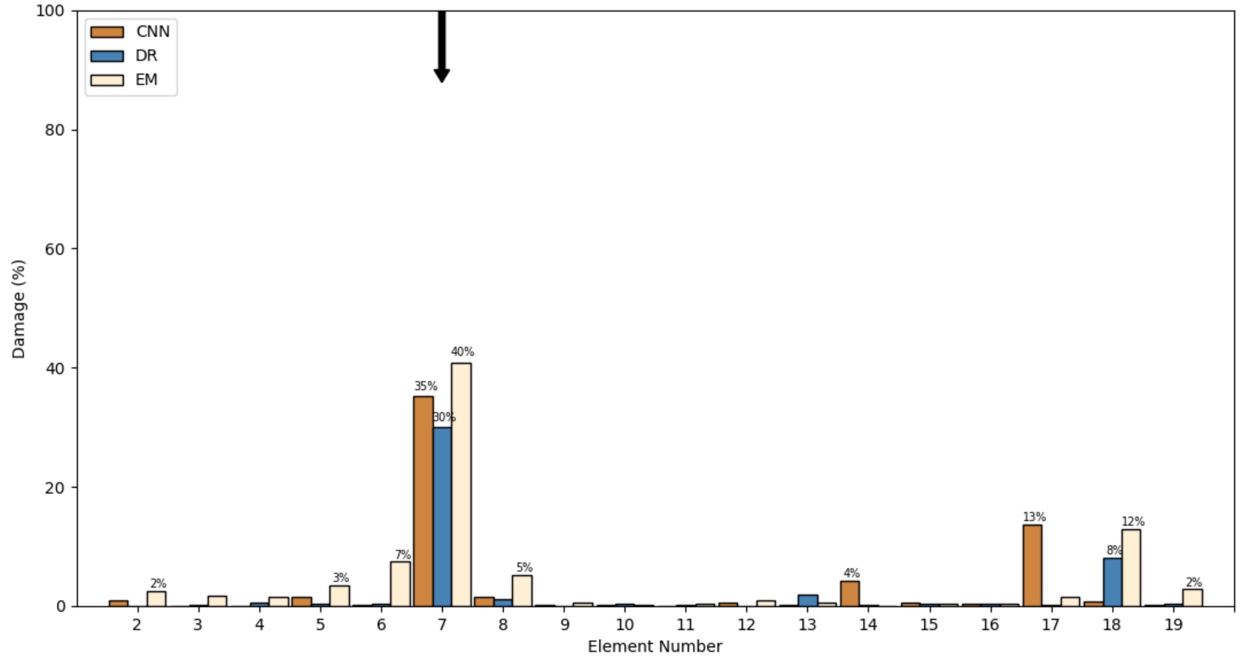


Figure 5.52 Comparison between models, test 1, model 3

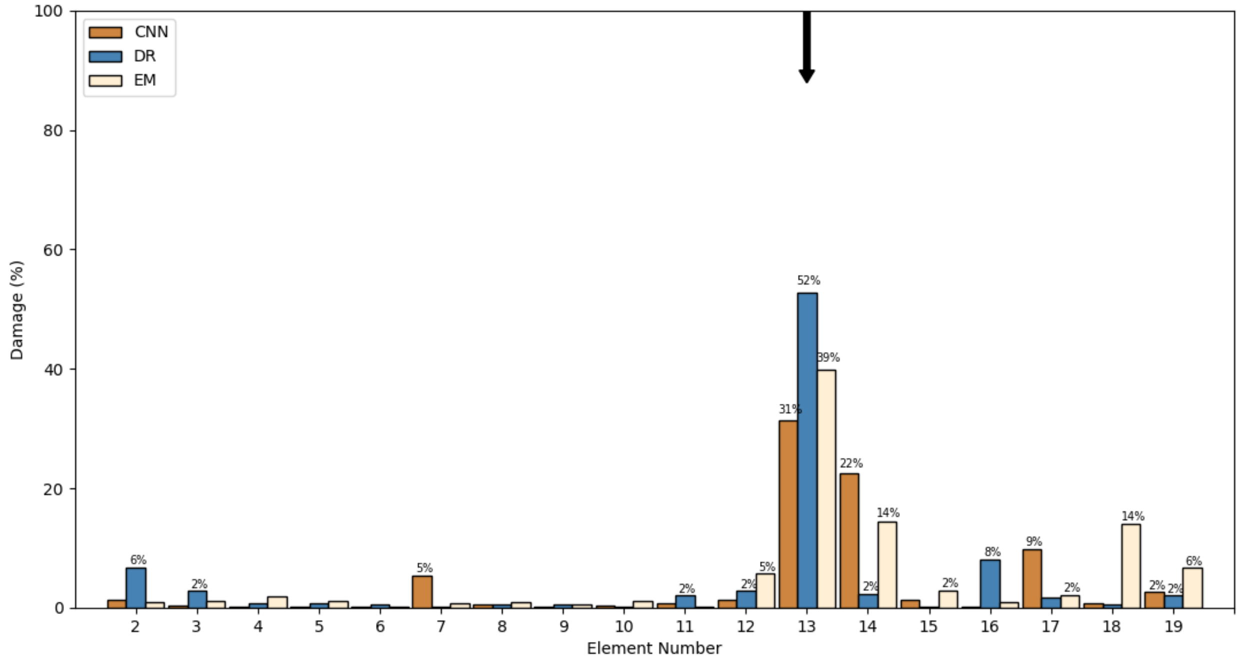


Figure 5.53 Comparison between models, test 2, model 3

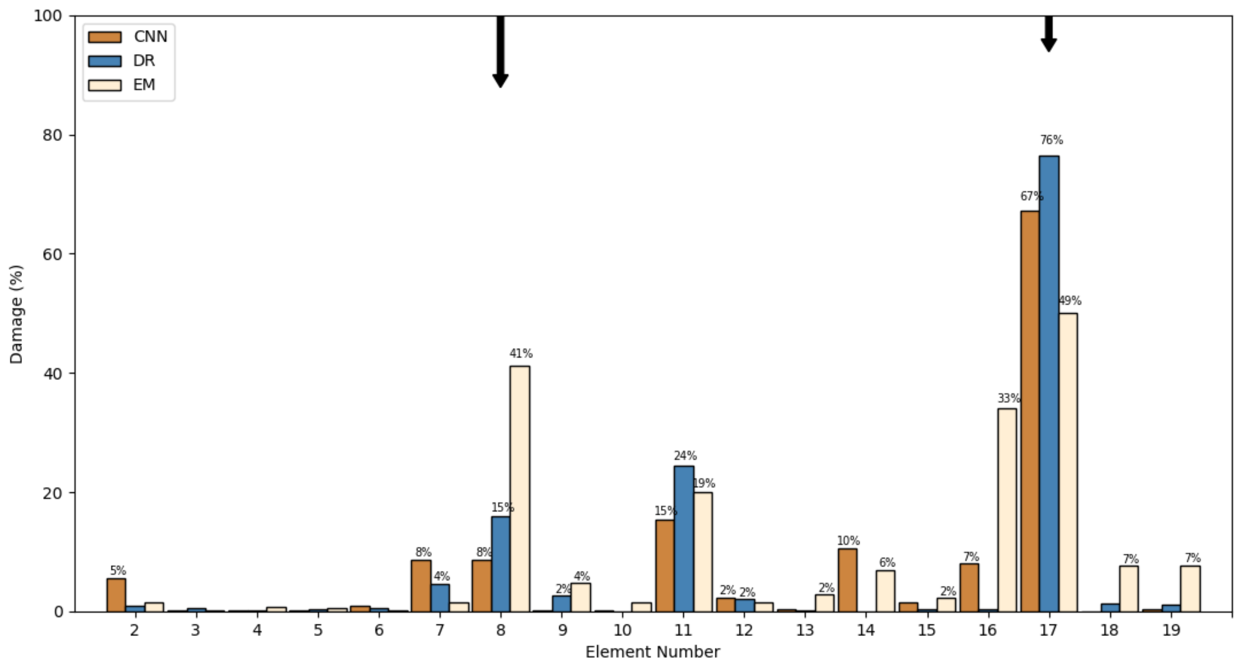


Figure 5.54 Comparison between models, test 3, model 3

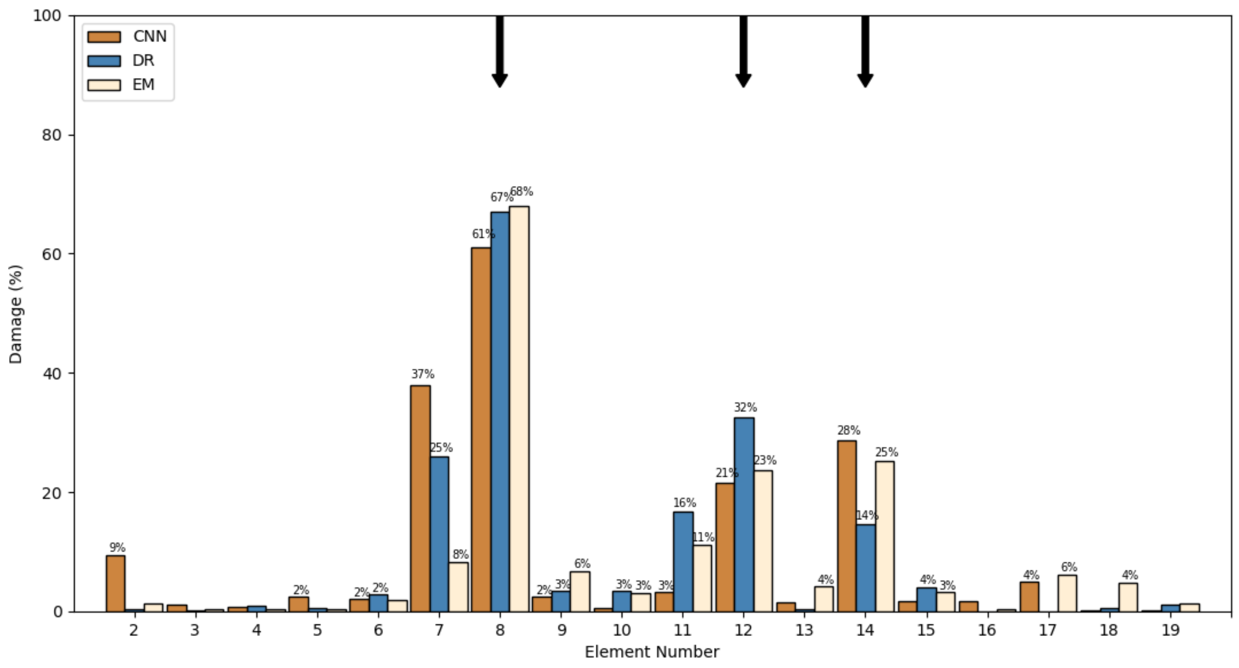


Figure 5.55 Comparison between models, test 4, model 3

5.2.6 Influence of dropout, dynamic routing

With the purpose of achieving better generalization between the training sets (which are generated through a FE model) and the experimental cases, dropout with $p = 0.5$ is added in the first hidden layer, for dynamic routing models only. Training results are shown in table 5.8:

Table 5.8 Comparison between models with and without dropout, training phase

Model	Dropout	MSE [%]	DME [%]	FAE [%]	Time per epoch [s]
1	No	0.038	1.17	65.30	22.3
	Yes	0.138	3.79	51.67	22.5
2	No	0.415	5.33	55.20	30.1
	Yes	0.610	9.96	47.96	48.11
3	No	0.783	6.17	44.70	132.2
	Yes	0.949	10.53	46.99	137.4

Each model is tested using the experimental cases. The results are shown in the following figures:

Model 1:

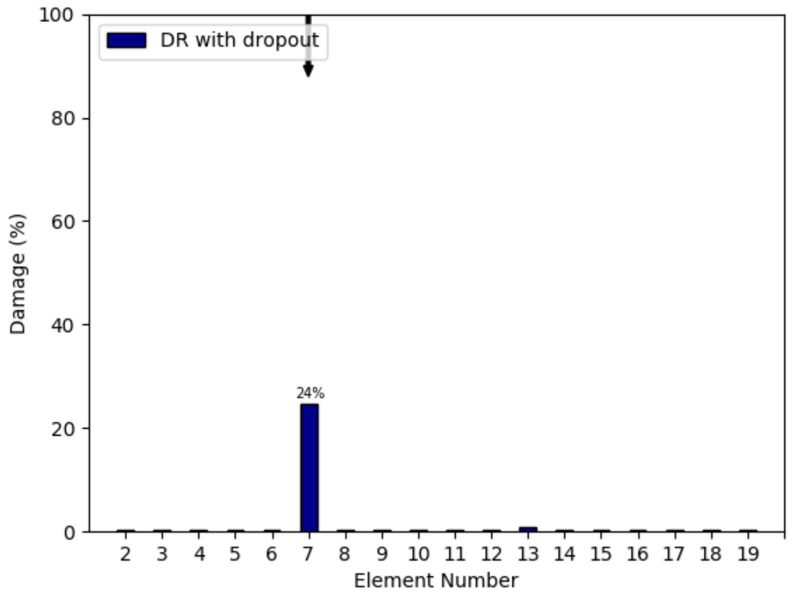


Figure 5.56 Experimental test 1 with model 1, dynamic routing with dropout

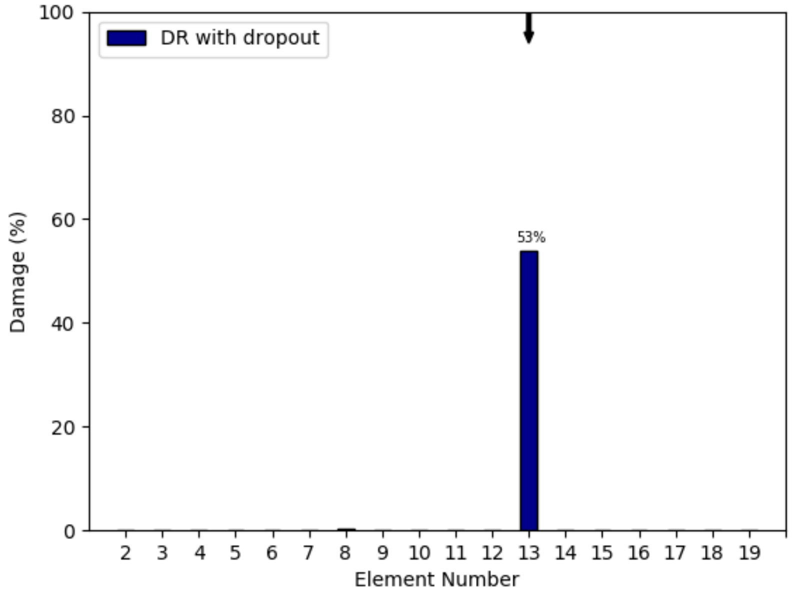


Figure 5.57 Experimental test 2 with model 1, dynamic routing with dropout

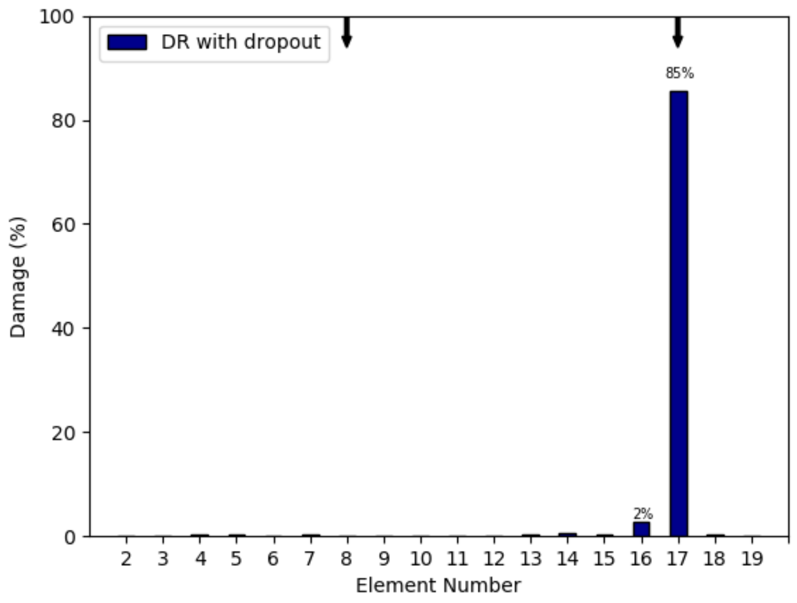


Figure 5.58 Experimental test 3 with model 1, dynamic routing with dropout

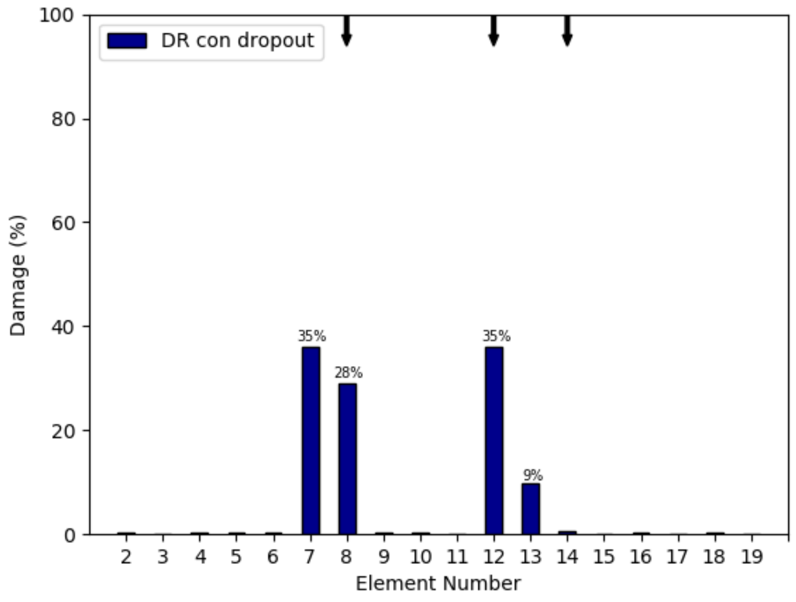


Figure 5.59 Experimental test 4 with model 1, dynamic routing with dropout

Model 2:

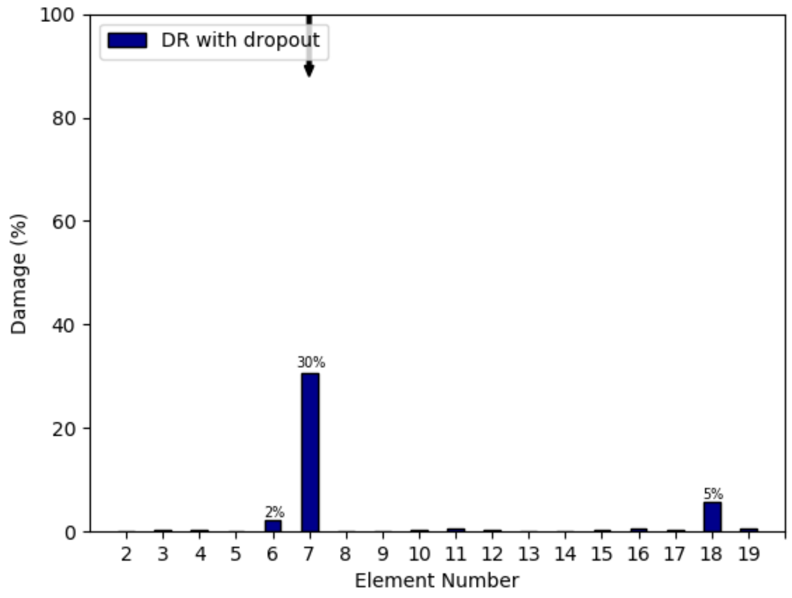


Figure 5.60 Experimental test 1 with model 2, dynamic routing with dropout

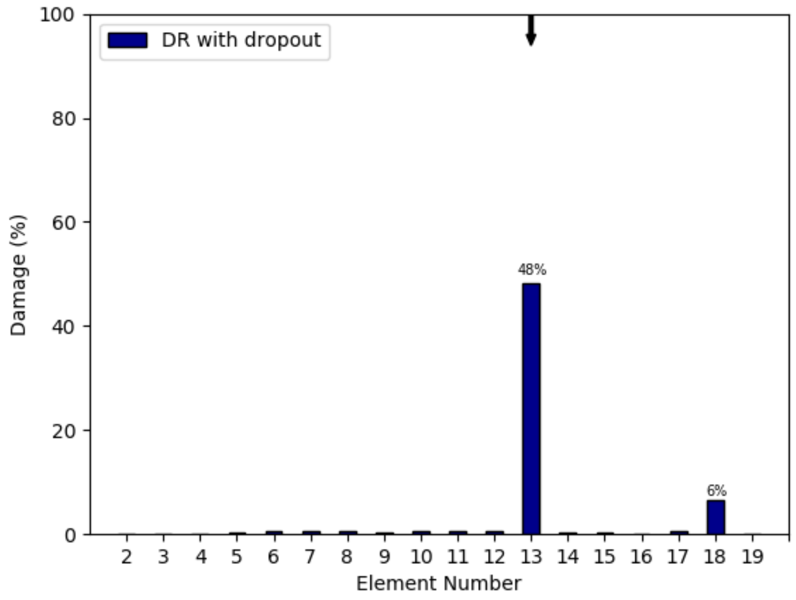


Figure 5.61 Experimental test 2 with model 2, dynamic routing with dropout

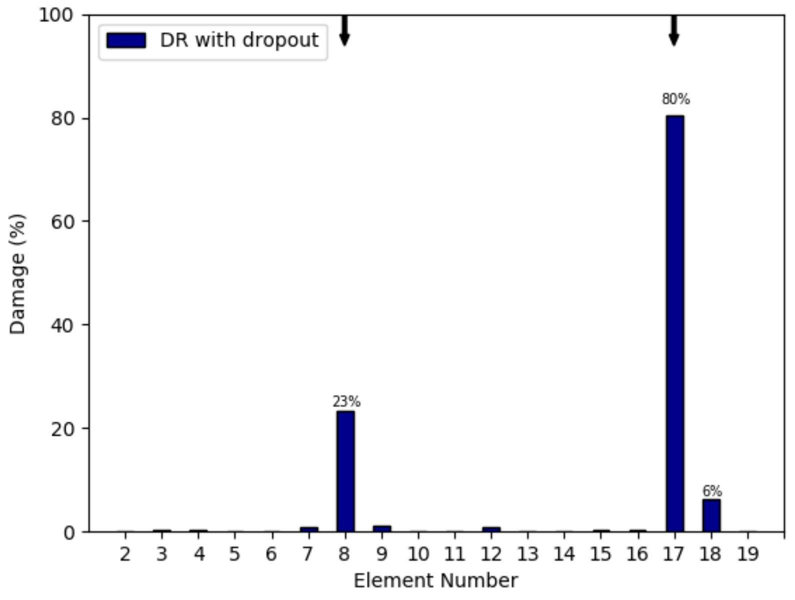


Figure 5.62Experimental test 3 with model 2 , dynamic routing with dropout

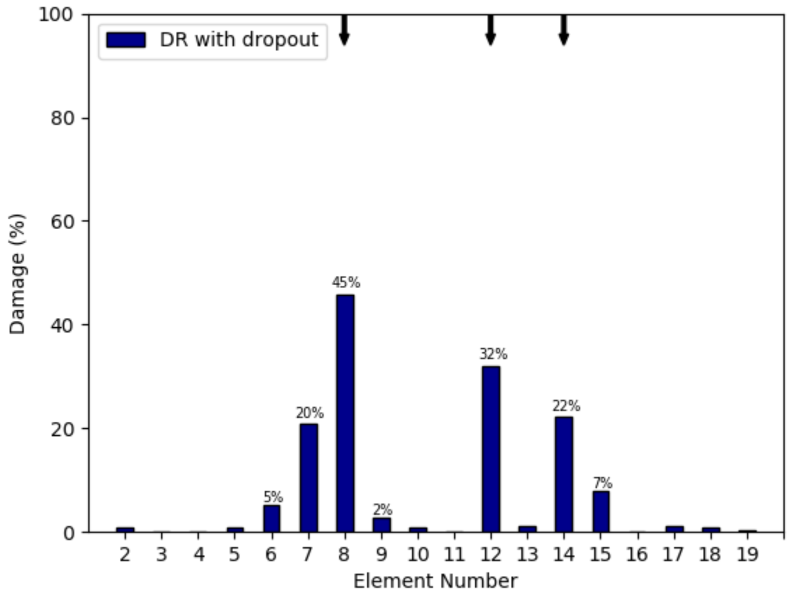


Figure 5.63 Experimental test 4 with model 2, dynamic routing with dropout

Model 3:

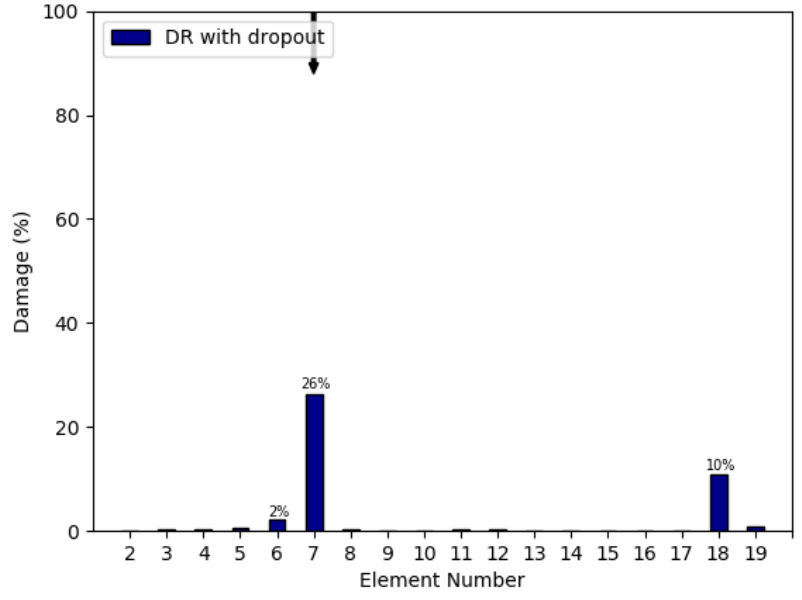


Figure 5.64 Experimental test 1 with model 3, dynamic routing with dropout

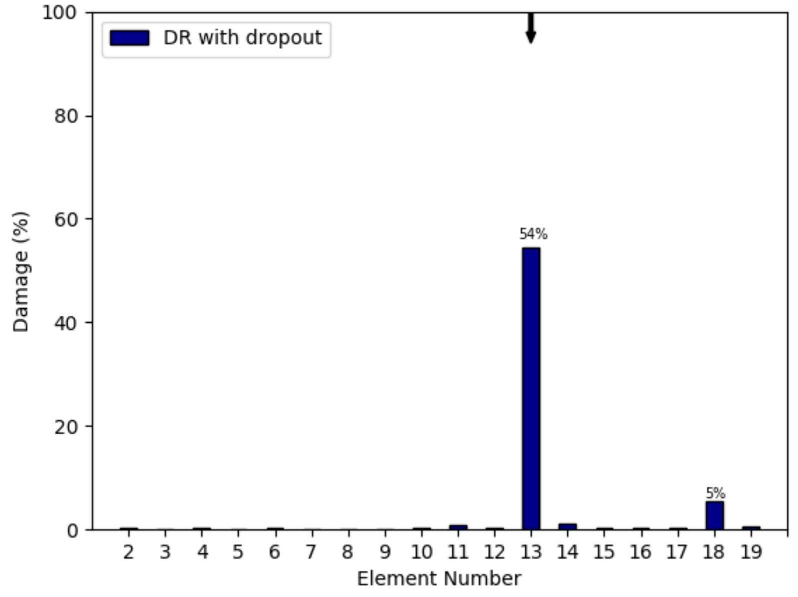


Figure 5.65Experimental test 2 with model 3, dynamic routing with dropout

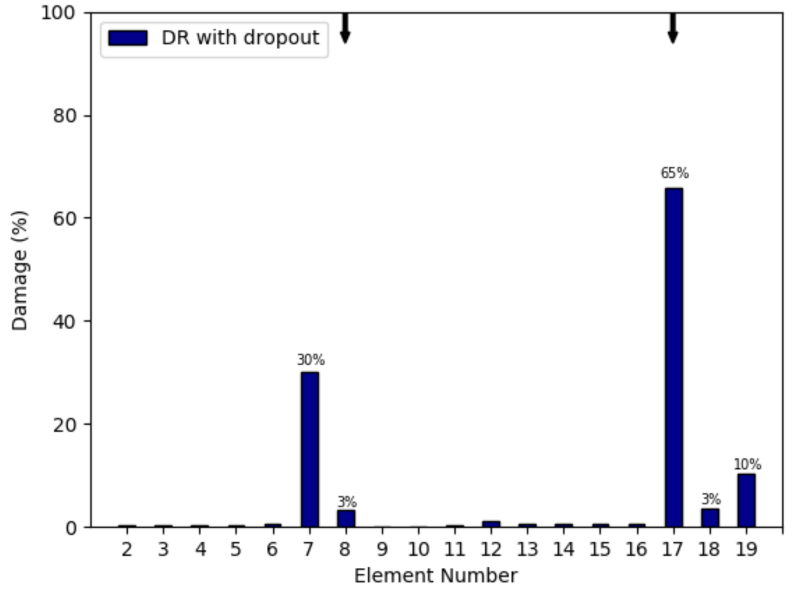


Figure 5.66Experimental test 3 with model 3, dynamic routing with dropout

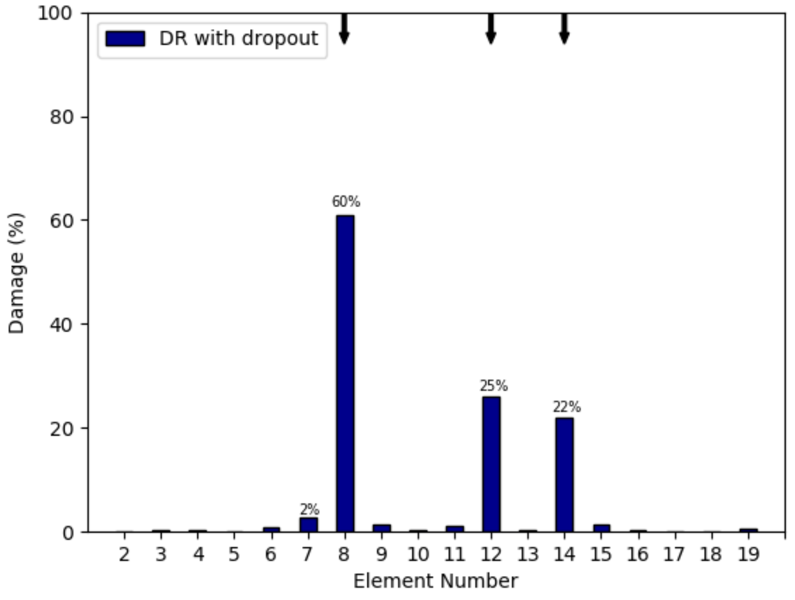


Figure 5.67 Experimental test 4 with model 3, dynamic routing with dropout

Every model is compared with the better result previously achieved, which is dynamic routing without dropout.

Model 1:

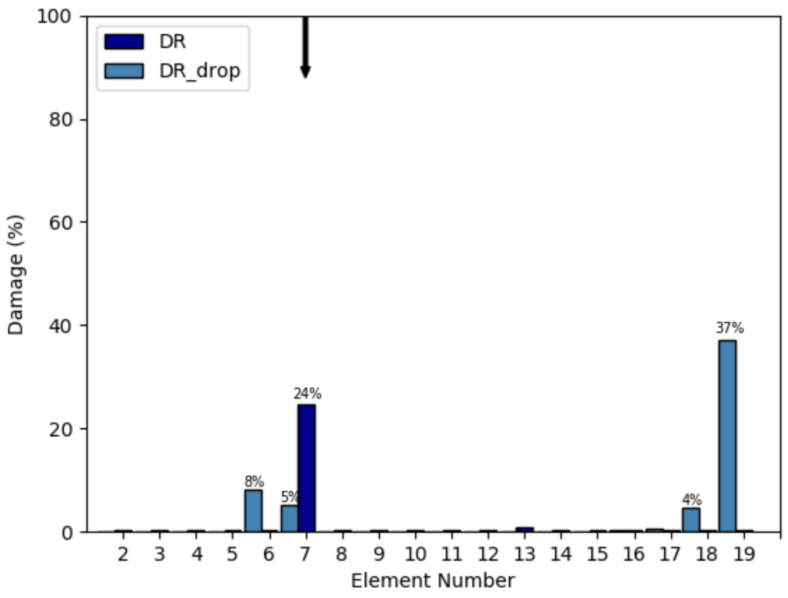


Figure 5.68 Effects of dropout on test 1, model 1

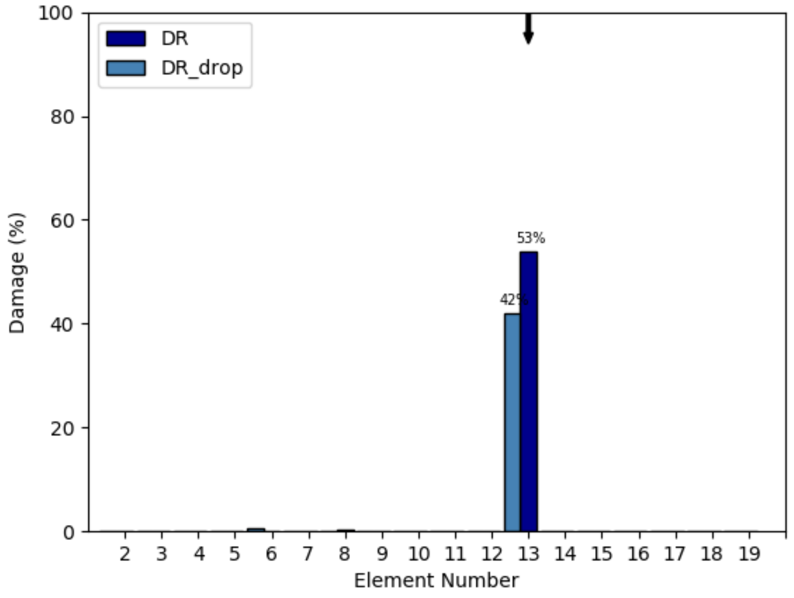


Figure 5.69 Effects of dropout on test 2, model 1

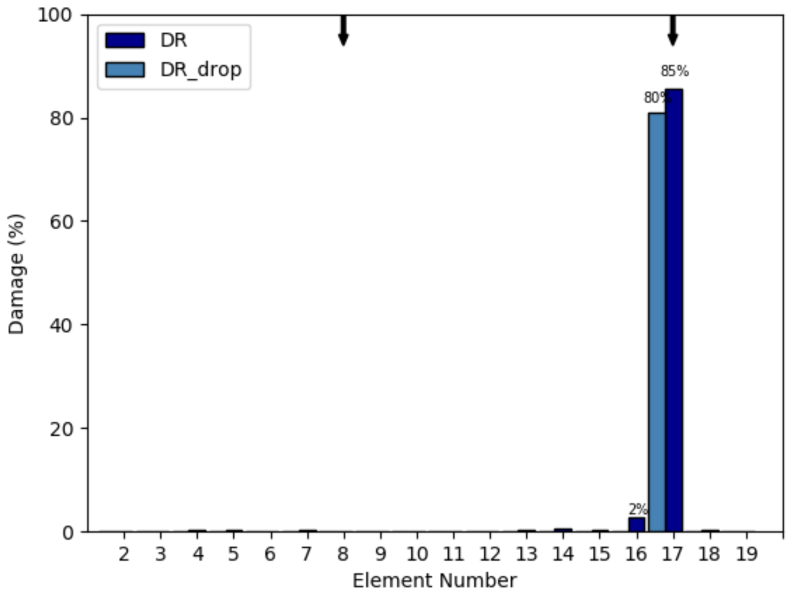


Figure 5.70 Effects of dropout on test 3, model 1

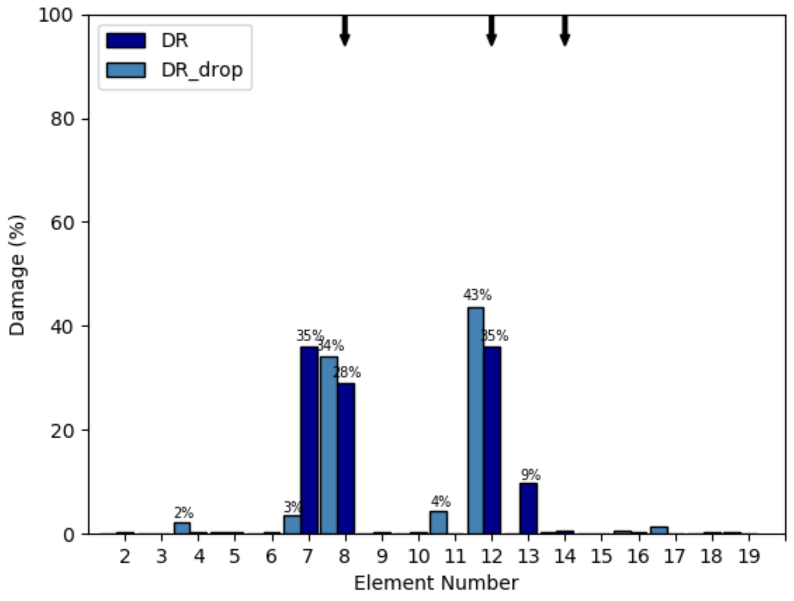


Figure 5.71 Effects of dropout on test 4, model 1

Model 2:

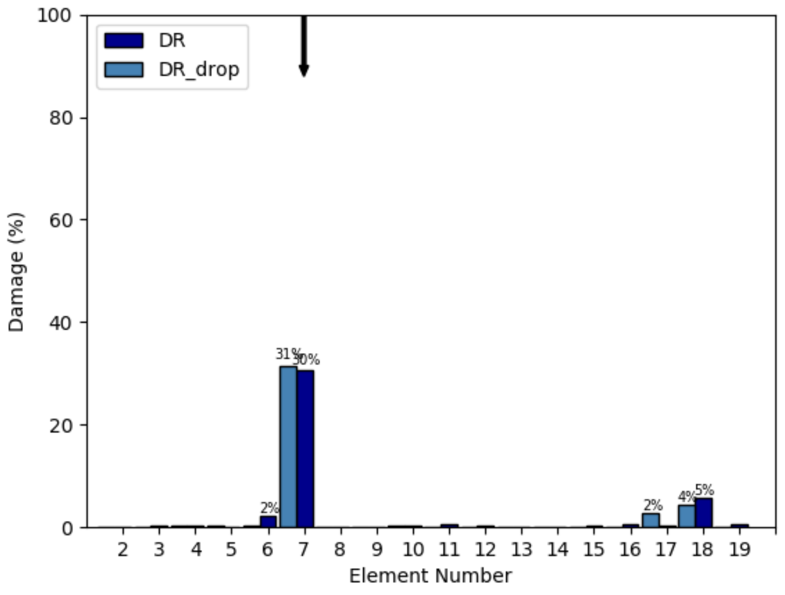


Figure 5.72 Effects of dropout on test 1, model 2

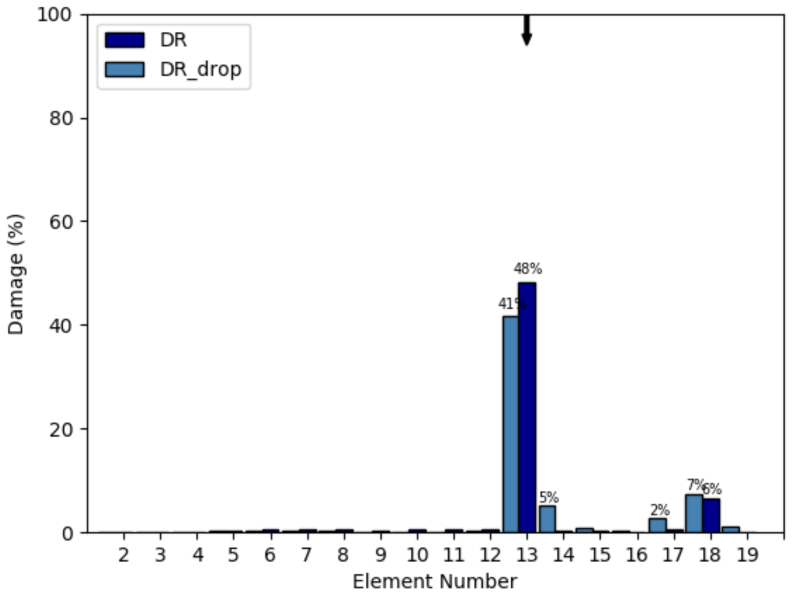


Figure 5.73 Effects of dropout on test 2, model 2

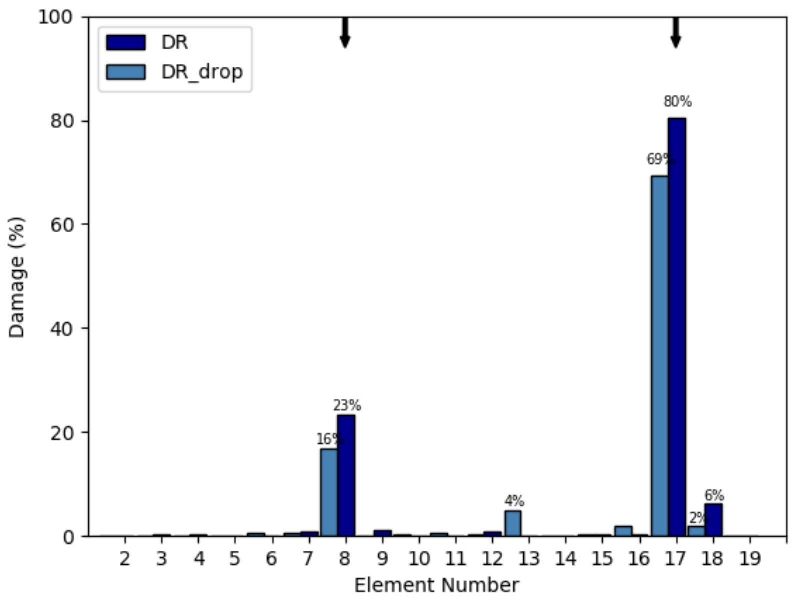


Figure 5.74 Effects of dropout on test 3, model 2

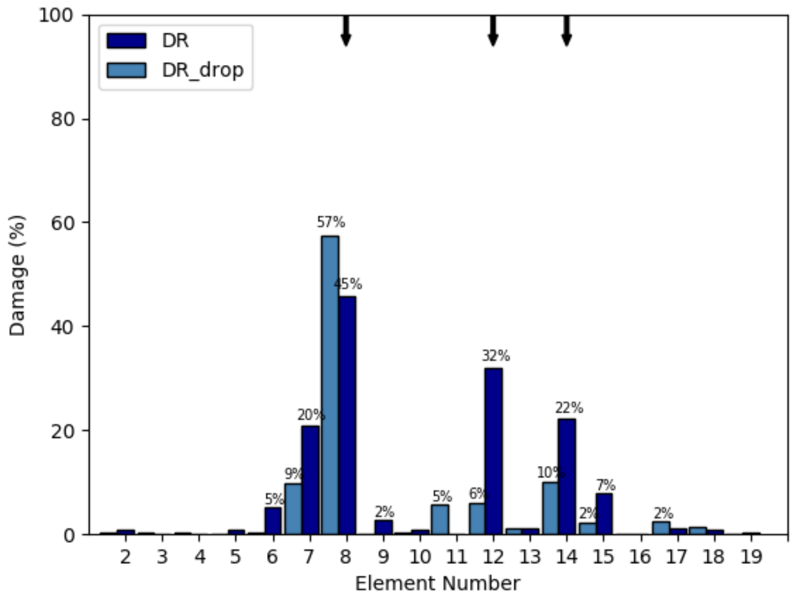


Figure 5.75 Effects of dropout on test 4, model 2

Model 3:

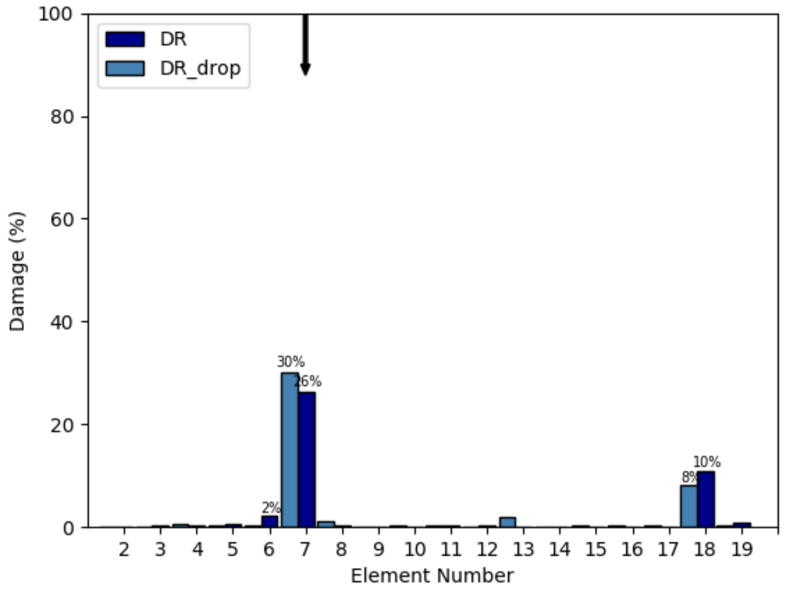


Figure 5.76 Effects of dropout on test 1, model 3

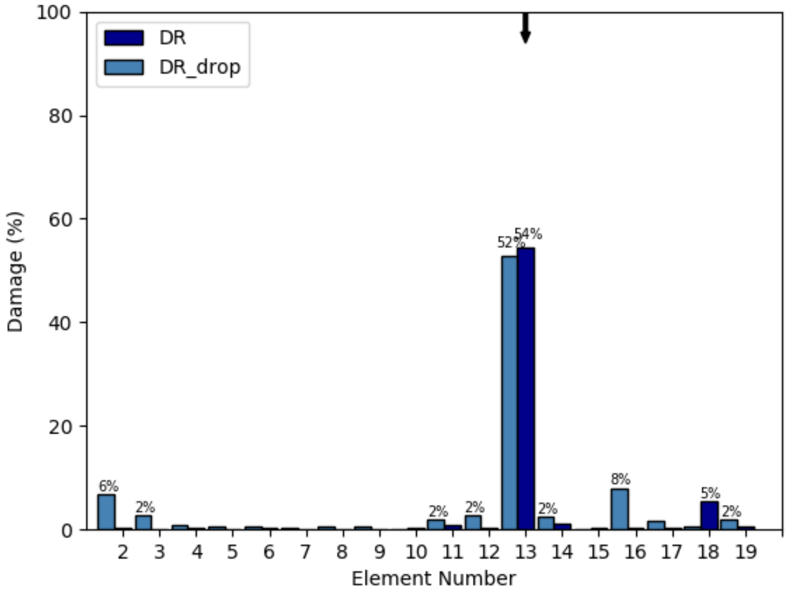


Figure 5.77 Effects of dropout on test 2, model 3

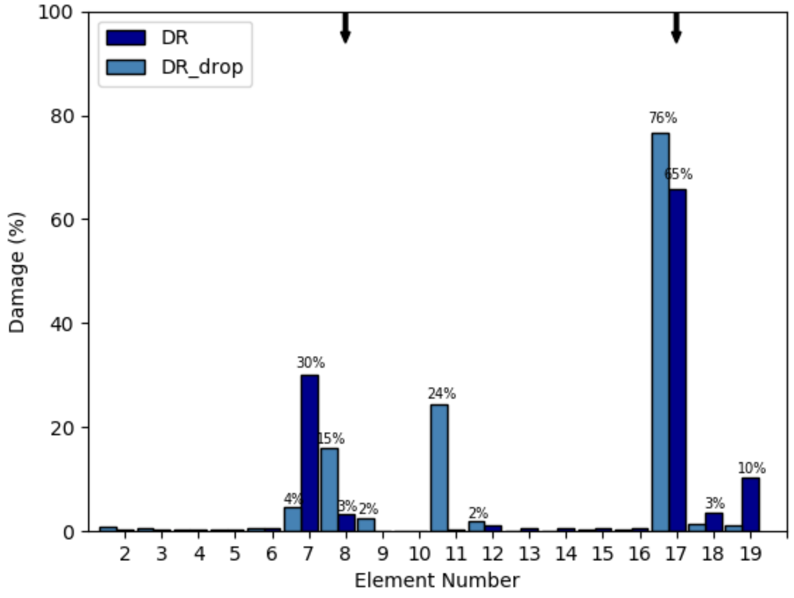


Figure 5.78 Effects of dropout on test 3, model 3

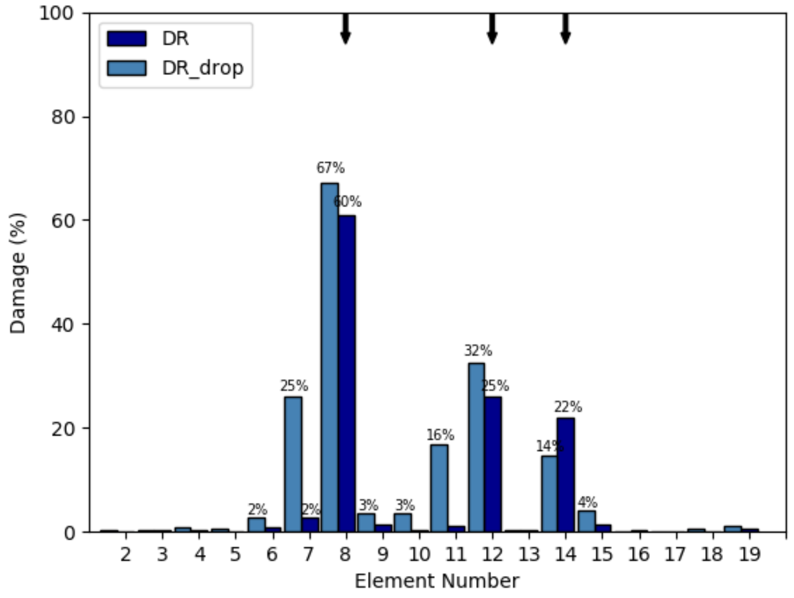


Figure 5.79 Effects of dropout on test 4, model 3

6. Discussion and Analysis

In the previous chapter, results for both case studies are shown. For each case, CapsNets type architectures are tested with two routing algorithms: dynamic routing and EM routing. These two are compared with the (next) best result, which is achieved using a CNN based architecture. In this chapter, results are discussed and analyzed by cases.

6.1 Case Study 1: Spring-mass system

During training, tables 5.3 and 5.4 show that capsule networks achieve great training results with both routing alternatives. Particularly, it can be seen that the *MSE* metric reaches the top value of 0.25%, which is promising. This is explained by the fact that the cost function (shown in equation 3.1) and the *MSE* metric are very similar. In the first, the squared difference between each prediction and the real output is used, while in the latter, the absolute value of the same difference is used.

DME and *FAE* metrics are strongly related to the capability of the model to locate damaged elements. *DME* is particularly important. It measures the false negatives given by the model. It is highly expected that every model presents a small number of false negatives, for a false negative can lead to serious consequences. In this system, each of the 6 models show *DME* values under 1.6%. This shows that the model considers localization of damage during the optimization stage, even though the cost function's nature is primarily related to quantification. The minimum value the cost function can achieve is when not only the quantity of damage in the system 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 node*.

On the other hand, the *FAE* metric measures the amount of false positives, and in the 6 models, the maximum value reaches 49%. As models get more complex (in terms of the kind of images used for training), *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 capsule networks with dynamic routing and with EM routing, tables 5.3 and 5.4 show that training results are very similar in terms of performance metrics, except for *FAE*. Capsule networks 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. Models with dynamic routing take much less time in training that those with EM routing.

When analyzing experimental results, it can be noticed that capsule networks models with the two kinds of routing successfully identify and quantify the 5th damaged element (see figures 5.2 to 5.7), however, as models are trained with more kinds of images, more relevant false positive values appear. It seems that the *FAE* value increases, although training results show the opposite. This is explained by the fact that false positive values increase in their magnitude, not in their quantity. Is not that there are more false positive values, it is that those fewer values are more perceptible.

When comparing capsule networks-based models with CNN, it can be noted that all three models achieve similar results. The three are able to quantify correctly (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 system is not a continuous one, each spring is connected to two masses, this meaning that a stiffness reduction in one spring will affect the vibrations of two masses. The model may interpret this as a stiffness reduction in the adjacent element, showing then a false positive. Other false positive values that are not near the area correspond to a defect of the model (although every kind of false positive is a deficiency of the model, the latter kind is associated to a deficiency only in the algorithm, not explained by physical reasons). CNN based models present more false negatives than the capsule network based ones. In models 2 and 3 (figures 5.9 and 5.10) the total percentages of false negatives in CNN add up to 16% and 21% respectively, while in capsule networks with dynamic routing they add 7% and 8%, and using EM routing, 7% and 7%. Apart from summing larger quantities, false positives coming from CNN based models are more distributed along the spring-mass system than those using capsule networks. As mentioned above, this is not a good result and shows a poorer performance from CNN. For example, in figure 5.10, a 4% false positive is shown in the first element of the system. The two capsule network based models don't show any amount of damage in that element.

In terms of generalization, neither CNN nor capsule networks based models suffer from overfitting, or at least not in a relevant way. All models are capable of identifying the correct damaged element and to measure it accurately. All models present false negative values, but since they are mainly in the range of 0-10% damage, they are not too relevant. In this way, experimental results match with training results.

Although the analyzed system exists merely for research purposes, this outcome shows that localization and quantification of structural damage can be done through capsule networks and aim for promising results.

6.2 Case Study 2: Beam

In this case, tables 5.6 and 5.7 show promising results for all models, especially in case 1. *MSE* values increase as models get more complex, regardless the routing algorithm. In the case of dynamic routing, the minimum *MSE* reaches a value of 0,038% and the maximum, a value of 0,783%. In the case of EM routing, the minimum and maximum are 0,132% and 0,948%, respectively. This occurs because, as more kinds of images are fed to an algorithm, the total damage in an individual image has a high probability of increasing, and there are more damaged elements. The model must hold more information from the images in its weights and biases. Thus, the task of recognizing exactly the same amount of damage and assigning it to the correct positions in the beam is harder than in, for example, model 1. This also explains the increase in *DME*. As with the first case study, *FAE* values decrease because, as images contain more damaged elements, there are less possibilities for false alarms.

In figures 5.14, 5.15, 5.16, 5.29, 5.30 and 5.31, *FAE* and *DME* values are presented in detail, according to their damage percentage. This is important considering that it is not the same to have, for example, a 70% damage false negative than a 3% one. The same happens for false alarms. Results show that, for models 1, false alarms and false negatives only accumulate in the range of 0-10%. For models 2 and 3, those values accumulate in the 0-10%, 10-20%, and 20-30%. This means that, for any model, training results suggest that any element with more than a 30% damage is properly identified (meaning that 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). It is also noted that false negatives are far more scarce than false positives, without the need of punishing a false negative more than a false positive in the cost function. This indicates that the approach is conservative, which is a concept to aim at in structural damage assessment. Perfect precision is something no model can achieve. Thus, it is better to overestimate damage than to underestimate it.

Comparing both routing algorithms, it can be observed that dynamic routing clearly outperforms EM routing in model 1. All of the three metrics present better results than those with EM routing. That difference is not so clear in models 2 and 3. In model 2, dynamic routing is better at quantifying damage (i.e better *MSE*), but less accurately when localizing damage (i.e, lower *DME* and *FAE*) than EM routing. In model 3, both routing algorithms present virtually the same results. This is explained by the types of architectures used for each routing algorithm. The architecture used along with the dynamic routing algorithm has 2, 3 or 4 secondary capsules, depending on the model. Those capsules are used to identify the kind of image, in terms of damaged elements. Model 1 has 2 secondary capsules, for there are only two kinds of images (undamaged beam and 1 damaged element). The 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 routing algorithm are very separated, unlike those in the other models. The algorithm can isolate undamaged images with relative ease, however, segregation between damaged images results a harder task for the algorithms. In the case of capsule networks 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 the secondary capsules' layer comprises the task of locating damage, and not only identifying the number of damaged elements like capsules in dynamic routing. The number of secondary capsules doesn't change from model to model, so the architecture is the same for all. This is why results, although they change from model to model, they don't vary as much as with dynamic routing models.

All six models are evaluated on the experimental cases. Each of the 4 cases is tested in each of the 6 models. For models 1, experimental cases 1 and 2 present the best results. This is an expected result, since models 1 are trained to detect up to 1 damaged element. Cases 3 and 4 show that the algorithms do not present great capability of generalization. This is also an awaited result, mainly because models do not see this kind of images during training. In the case of dynamic routing, model 1 is intrinsically built to recognize only 0 or 1 damaged elements due to its secondary capsules. This affects its facility at generalizing.

In figure 5.17, it is shown that, even though the model can correctly identify the damaged element, it clearly doesn't quantify it correctly (see table 5.5) and displays an important false negative at the right end of the beam. That particular false negative is presented several times in all models.

Testing shows different results than those obtained by training process. According to figures 5.14, 5.15, 5.16, 5.29, 5.30 and 5.31, false negative values should be much smaller (in terms of damage size) than they are. Taking as an example figure 5.17 again, that false negative at the right end of the beam is incoherent with the information conveyed by figure 5.14. There should be no false positives with damage size over 10%. This discrepancy is far more substantial in those results achieved by EM routing than those with dynamic routing. Figures 5.37 and 5.38 show false positives with damage size over the 40%, which again, is inconsistent with the fact that, according to figure 5.30, there should be no false positives with that damage size.

Although experimental results yield a count for false positives that could be reduced, from a safety point of view, it is better to have false positives than false negatives. In this sense, all models perform almost impeccably. Figure 5.19 shows a false negative value, but it is explained by the fact that the case presents two damaged elements, whereas that particular model is trained to recognize beams with up to one damaged element. This also occurs at figures 5.20, 5.34, and 5.35. All four images correspond to cases 3 and 4, and in all of them, the element that isn't recognized is the one with the least damage percentage. When the model is trained to recognize (for example) up to one damaged element, the maximum amount of damaged distributed along the beam is 100%. In the case of figures 5.19 and 5.34, since there is already a damaged element with a considerable size (cut depth: 15/25 [mm]), the model “concludes” that there is no other damaged element because the limit amount of damage is to be reached. Something similar occurs with figures 5.20 and 5.35, where the maximum total damage is 200%. The rest of the cases present no false negative, neither using dynamic routing nor EM routing.

All experimental results are compared to the (previously) best result, which is obtained using CNN. From figures 5.44 to 5.55, the first and main conclusion is that capsule networks with dynamic routing present better results that CNN and capsules with EM routing. With exception of figure 5.44, where it presents the worst result (by wrongly assigning the damage amount to the end of the beam instead of the 7th element), all the experimental cases are best represented by DR. Most accurate cases are shown by figures 5.45, 5.50 and 5.55, where DR correctly measures damaged elements and exhibits the least amount of false positives. Furthermore, 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.

As established before, DR shows acceptable performance, surpassing CNN obtained results for training and test sets. On the other hand, EM routing has the problem of computing too many false positives. Examples of this are figures 5.36 and 5.37, where false positives are considerable and have an important size. For diagnostics purposes, these values can generate confusion in analysis, acting as distractors and thus making the model inaccurate and unreliable. This occurs because models using EM routing are too complex and this affects generalization ability. The fact of using 18 secondary capsules, one for each element in the beam, builds a complexity for the models that proves to be unnecessary. Since the model is more complex than the dataset, its capacity is used to overanalyze the dataset and then adjust itself in a way that loses generalization. Even though the same models are used with the spring-mass system, in that case overfitting doesn’t occur because of two main reasons. The first one is mentioned before and relates to the fact that each element of the spring-mass system is independent, and the system itself is a discrete one, unlike the beam which is continuous. The second is that only 7 capsules are used and not 18, meaning the model is less complex.

To detect overfitting, a data subset is used only to compute the loss function (or accuracy) value. This subset is not used for training purposes. This validation loss function is compared to the training set loss function to detect overfitting. In this case, no signs of overfitting where 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 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.

To tackle the aforementioned issue, dropout technique was used in the last layer of the fully connected neural network. Even though overfitting is more relevant in EM routing than in dynamic routing, dropout was applied only on the results obtained through the latter, because they were the best results and had greater potential to achieve even better results. By using dropout, the network lowers its results but has a greater generalization capability. Table 5.8 shows a comparison between results obtained with dropout and without it. 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 from figures 5.56 to 5.67. Model 1 achieves to correctly locate 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 3 and 4, the model cannot recognize the element with the less amount of damage. However, figure 5.71 shows that the model with dropout 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 model 2, experimental results also improve with the use of dropout. The model still performs well on cases 1 and 2, showing only small percentage false positives. On case 3 the model is capable of recognizing the element with the small amount of damage as well as the one with great amount, showing only a 6% of false positive value located in the 18th element, next to the truly damaged element. On case 4, despite the model being trained with up to 2 damaged elements images (thus allowing some mistakes), the algorithm detects correctly the three damaged elements, with the drawback of showing an important number of false negatives. 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 quantify more damaged elements than those with which the algorithm learned the task, per beam.

For model 3, in general terms the algorithm correctly assesses damage, better than without using dropout. Particularly, figures 5.76 and 5.77 show that the use of dropout affects directly on the quantity of false positive values. Figure 5.78 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 though, this isn't a highly relevant mistake, because the element is displaced only one element, but its quantity is correctly assigned. Finally, figure 5.79 shows that the model recognizes 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 most damage percentage. Comparing with the same model but without using dropout, there is a big difference in terms of false positives. A 25% false positive, located at element 7, and a 16% one located at element 11 act as big distractors in terms of assessment, which are not present when using dropout.

7. Conclusion and Comments

7.1 Conclusions

In the present work, a Capsule Network based model was developed, with the objective of locating and quantifying damage in structural elements. The idea is to study the application of Capsule Networks to this task and to evaluate and compare its performance with the state-of-the-art. Two types of routing algorithms within Capsule Networks are studied: Dynamic Routing (DR) and EM Routing (EMR). Both architectures are analyzed with two case studies: a spring-mass system and an experimental beam. In both cases, models are trained using images containing 10 transmissibility functions, each image representing various damage scenarios. Images are created using a FE model tuned to represent the experimental cases. Trained models are validated using experimental cases and compared with CNNs, which are the state-of-the-art in that kind of problems.

Results show that during training, Capsule Networks outperform CNN, with no signs of overfitting. During testing there were difference between the case studies. In the first one (spring-mass system), training results match with experimental results, there are no false negative values and few false positives. In the second case study, training results differ from experimental results, leading to the conclusion that overfitting is occurring, although there were no signs when observing the validation set. For this reason, dropout is applied to the network with DR, obtaining worse results during training but better generalization capacity, which is seen in experimental results. These final results clearly outperform CNN, notably reducing false positive values, while maintaining a correct damage estimation at the correct locations.

7.2 Comments and Future Work

The present work shows that Capsule Networks, a new Deep Learning algorithm, can be used to locate and estimate structural damage. The Dynamic Routing algorithm is best suited for this task, whilst the EM Routing algorithm, even though it presents similar results to DR, clearly overfits due to its complexity.

Although it is out of this work's scope, a way to improve the way of visualizing results and measuring performance is to give the "real" damage percentage at experimental images, to the beam case study, to compare results. Although the cut depth gives a first estimate and serves for comparison, that estimation is very broad and lacks precision. This can be performed simulating the real beam with a FE model and comparing its natural frequencies and vibration modes with those obtained through a simplified model with a stiffness reduction in the desired element or elements.

Capsule Networks have been a major upgrade to CNNs, and this work shows they can be used in damage detection tasks. With the application of dropout, the model overcomes overfitting which arises from the fact that the model integrally learns the equation that generates the FEM data images. That equation cannot represent authentically what occurs in reality, and that discrepancy makes it necessary to include dropout. The routing algorithm still needs to be improved, to compete with CNNs in terms of computational time. That is the great drawback of Capsule Networks, and it is why still there are no major applications of the algorithm, unlike CNNs.

Bibliography

[1] D. E. Siskind, M. S. Stagg, J. W. Kopp, e C. H. Dowding, “Structure response and damage produced by ground vibration from surface mine blasting”, 1981.

[2] W. I. Duvall e D. E. Fogelson, “Review of Criteria for Estimating Damage to Residences from Blasting Vibrations - RI 5968”, 1962.

[3] Z. Hou, M. Noori, e R. St. Amand, “Wavelet-Based Approach for Structural Damage Detection”, *J. Eng. Mech.*, vol. 126, nº 7, p. 677–683, jul. 2000.

[4] Z. Q. Lang *et al.*, “Transmissibility of non-linear output frequency response functions with application in detection and location of damage in MDOF structural systems”, *Int. J. Non. Linear. Mech.*, vol. 46, nº 6, p. 841–853, 2011.

[5] K. Worden, “Structural Fault Detection using a Novelty Meassure”, *J. Sound Vib.*, vol. 201, nº 1, p. 85–101, 1997.

[6] V. Meruane e A. Ortiz-Bernardin, “Structural damage assessment using linear approximation with maximum entropy and transmissibility data”, *Mech. Syst. Signal Process.*, vol. 54, p. 210–223, 2015.

[7] S. Chesné e A. Deraemaeker, “Damage localization using transmissibility functions: A critical review”, *Mech. Syst. Signal Process.*, vol. 38, nº 2, p. 569–584, 2013.

[8] M. J. Sc. and F. F. H.Zhang, “Structural Health Monitoring Using Transmittance Functions”, *Mech. Syst. Signal Process.*, vol. 13, nº 5, p. 765–787, 1999.

[9] X. Yi, D. Zhu, Y. Wang, J. Guo, e K.-M. Lee, “Transmissibility-function-based structural damage detection with tetherless mobile sensors”, *Proc. 5th Int. Conf. Bridg. Maintenance, Saf. Manag.*, p. 328–335, 2010.

[10] N. M. M. Maia, R. A. B. Almeida, A. P. V. Urgueira, e R. P. C. Sampaio, “Damage detection and quantification using transmissibility”, *Mech. Syst. Signal Process.*, vol. 25, nº 7, p. 2475–2483, 2011.

[11] K. Worden, L. Y. Cheung, e J. a Rongong, “Damage detection in an aircraft component model”, *Proc. Int. Modal Anal. Conf. - IMAC*, vol. 2, nº October, p. 1234–1241, 2001.

[12] K. Worden, G. Manson, e D. Allman, “Experimental validation of a structural health monitoring methodology: Part I. Novelty detection on a laboratory structure”, *J. Sound Vib.*, vol. 259, nº 2, p. 323–343, 2003.

[13] G. Manson, K. Worden, e D. Allman, “Experimental validation of a structural health monitoring methodology. Part II. Novelty detection on a Gnat aircraft”, *J. Sound Vib.*, vol. 259, nº 2, p. 345–363, 2003.

[14] G. Manson, K. Worden, e D. Allman, “Experimental validation of a structural health monitoring methodology: Part III. Damage location on an aircraft wing”, *J. Sound Vib.*, vol. 259, nº 2, p. 365–385, 2003.

[15] A. P. V. Urgueira, R. A. B. Almeida, e N. M. M. Maia, “On the use of the

transmissibility concept for the evaluation of frequency response functions”, *Mech. Syst. Signal Process.*, vol. 25, nº 3, p. 940–951, 2011.

[16] L. Feng, X. Yi, D. Zhu, X. Xie, e Y. Wang, “Damage detection of metro tunnel structure through transmissibility function and cross correlation analysis using local excitation and measurement”, *Mech. Syst. Signal Process.*, vol. 60, p. 59–74, 2015.

[17] R. P. C. Sampaio, N. M. M. Maia, A. M. R. Ribeiro, e J. M. M. Silva, “Transmissibility techniques for damage detection”, *Proc. Imac-Xix A Conf. Struct. Dyn. Vols 1 2*, vol. 4359, nº February, p. 1524–1527, 2001.

[18] T. J. Johnson e D. E. Adams, “Transmissibility as a Differential Indicator of Structural Damage”, *J. Vib. Acoust.*, vol. 124, nº 4, p. 634, 2002.

[19] R. Perera e A. Ruiz, “A multistage FE updating procedure for damage identification in large-scale structures based on multiobjective evolutionary optimization”, *Mech. Syst. Signal Process.*, vol. 22, nº 4, p. 970–991, maio 2008.

[20] Y.-J. Cha e O. Buyukozturk, “Structural Damage Detection Using Modal Strain Energy and Hybrid Multiobjective Optimization”, *Comput. Civ. Infrastruct. Eng.*, vol. 30, nº 5, p. 347–358, maio 2015.

[21] 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.*, vol. 47, nº 1, p. 1–8, jan. 2012.

[22] N. Khaji e M. Mehrjoo, “Crack detection in a beam with an arbitrary number of transverse cracks using genetic algorithms”, *J. Mech. Sci. Technol.*, vol. 28, nº 3, p. 823–836, 2014.

[23] M. Mehrjoo, N. Khaji, H. Moharrami, e A. Bahreininejad, “Damage detection of truss bridge joints using Artificial Neural Networks”, *Expert Syst. Appl.*, vol. 35, nº 3, p. 1122–1131, 2008.

[24] J. J. Lee, J. W. Lee, J. H. Yi, C. B. Yun, e H. Y. Jung, “Neural networks-based damage detection for bridges considering errors in baseline finite element models”, *J. Sound Vib.*, vol. 280, nº 3–5, p. 555–578, 2005.

[25] J. L. Zapico, K. Worden, e F. J. Molina, “Vibration-based damage assessment in steel frames using neural networks”, *Smart Mater. Struct.*, vol. 10, nº 3, p. 553–559, jun. 2001.

[26] W. T. Yeung e J. W. Smith, “Damage detection in bridges using neural networks for pattern recognition of vibration signatures”, *Eng. Struct.*, vol. 27, nº 5, p. 685–698, 2005.

[27] Szewczyk, Z. Peter, e P. Hajela, “Damage detection in structures based on feature-sensitive neural networks”, *J. Comput. Civ. Eng.*, vol. 8, nº 2, p. 163–178, abr. 1994.

[28] V. Meruane, “Online Sequential Extreme Learning Machine for Vibration-Based Damage Assessment Using Transmissibility Data”, *J. Comput. Civ. Eng.*, vol. 30, nº 3, p. 04015042, 2016.

[29] D. Verstraete, A. Ferrada, E. L. Droguett, V. Meruane, e M. Modarres, “Deep learning enabled fault diagnosis using time-frequency image analysis of rolling

element bearings”, *Shock Vib.*, vol. 2017, 2017.

[30] F. Jia, Y. Lei, J. Lin, X. Zhou, e N. Lu, “Deep neural networks: A promising tool for fault characteristic mining and intelligent diagnosis of rotating machinery with massive data”, *Mech. Syst. Signal Process.*, vol. 72–73, p. 303–315, maio 2016.

[31] M. Gan, C. Wang, e C. Zhu, “Construction of hierarchical diagnosis network based on deep learning and its application in the fault pattern recognition of rolling element bearings”, *Mech. Syst. Signal Process.*, vol. 72–73, p. 92–104, maio 2016.

[32] J. Wang, J. Zhuang, L. Duan, e W. Cheng, “A multi-scale convolution neural network for featureless fault diagnosis”, in *2016 International Symposium on Flexible Automation (ISFA)*, 2016, p. 65–70.

[33] L. Liao, W. Jin, e R. Pavel, “Enhanced Restricted Boltzmann Machine with Prognosability Regularization for Prognostics and Health Assessment”, *IEEE Trans. Ind. Electron.*, vol. 63, nº 11, p. 7076–7083, nov. 2016.

[34] X. Guo, L. Chen, e C. Shen, “Hierarchical adaptive deep convolution neural network and its application to bearing fault diagnosis”, *Meas. J. Int. Meas. Confed.*, vol. 93, p. 490–502, nov. 2016.

[35] C. Szegedy *et al.*, “Going deeper with convolutions”, *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 07–12–June, p. 1–9, 2015.

[36] R. Girshick, J. Donahue, T. Darrell, e J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation”, *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. p. 580–587, 2014.

[37] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, e Y. LeCun, “OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks”, dez. 2013.

[38] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, e C. Bregler, “Efficient object localization using Convolutional Networks”, *2015 IEEE Conf. Comput. Vis. Pattern Recognit.*, p. 648–656, 2015.

[39] Y. Taigman, M. Yang, M. Ranzato, e L. Wolf, “DeepFace: Closing the gap to human-level performance in face verification”, *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. p. 1701–1708, 2014.

[40] W. Sun *et al.*, “An Intelligent Gear Fault Diagnosis Methodology Using a Complex Wavelet Enhanced Convolutional Neural Network”, *Materials (Basel).*, vol. 10, nº 7, p. 790, jul. 2017.

[41] G. E. Hinton, A. Krizhevsky, e 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, nº PART 1, Springer, Berlin, Heidelberg, 2011, p. 44–51.

[42] S. Sabour, N. Frosst, e G. E. Hinton, “Dynamic Routing Between Capsules”, nº Nips, 2017.

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

[44] G. Hinton, S. Sabour, e N. Frosst, “M Atrix Capsules With Em Routing”, 2018, p. 1–15.

[45] P. Afshar, A. Mohammadi, e K. N. Plataniotis, “Brain Tumor Type Classification via Capsule Networks”, 2018.

[46] Y. Upadhyay e P. Schrater, “Generative Adversarial Network Architectures For Image Synthesis Using Capsule Networks”, p. 1–9, 2018.

[47] E. Xi, S. Bing, e Y. Jin, “Capsule Network Performance on Complex Data”, vol. 10707, nº Fall, p. 1–7, 2017.

[48] R. LaLonde e U. Bagci, “Capsules for Object Segmentation”, 2018.

[49] P.-A. Andersen, “Deep Reinforcement Learning using Capsules in Advanced Game Environments”, jan. 2018.

[50] M. T. Bahadori, “Spectral Capsule Networks”, *ICLR Work.*, p. 1–5, 2018.

[51] A. E. W. Johnson *et al.*, “MIMIC-III, a freely accessible critical care database”, *Sci. Data*, vol. 3, p. 160035, maio 2016.

[52] D. Rawlinson, A. Ahmed, e G. Kowadlo, “Sparse Unsupervised Capsules Generalize Better”, 2018.

[53] M. I. Friswell e J. E. T. Penny, “Crack Modeling for Structural Health Monitoring”, *journals.sagepub.com*, vol. 1, nº 2, p. 139–148, 2016.

[54] N. Srivastava, G. Hinton, A. Krizhevsky, e R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, 2014.

[55] I. Goodfellow, Y. Bengio, A. Courville, e Y. Bengio, *Deep learning*. 2016.

[56] R. J. Douglas e K. A. Martin, “A functional microcircuit for cat visual cortex.”, *J. Physiol.*, vol. 440, nº 1, p. 735–769, 1991.

[57] J. Hawkins, S. Ahmad, e Y. Cui, “A Theory of How Columns in the Neocortex Enable Learning the Structure of the World”, *Front. Neural Circuits*, vol. 11, p. 0–17, 2017.