FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Prediction of Shell Finite-Element Stresses using Convolutional Neural Networks (CNN)

Nuno Gaspar

Mestrado em Engenharia e Ciência de Dados

Supervisor: Prof. Vera Miguéis

Co-Supervisor: Prof. José Luís Borges

September 30, 2022

Prediction of Shell Finite-Element Stresses using Convolutional Neural Networks (CNN)

Nuno Gaspar

Mestrado em Engenharia e Ciência de Dados

September 30, 2022

Abstract

Structural mechanics calculation methods can be computationally demanding and time-consuming. An example of that is the finite-elements method (FEM), which is the most popular method for solving such problems.

As a step to widen available approaches for solving these problems, the present work explores a deep learning based approach for predicting bending stress fields in plan 2D structures. These structures may be subject to different patterns and magnitudes of external out-of-plane loads, geometry and support conditions.

To that end, a dataset of finite-element structural models was firstly developed by means of commercial software, followed by the design, train and test of the deep learning models. This research is conducted using Convolutional Neural Networks (CNN), benefiting from its specialization in processing data that has a grid-like topology.

Results show that the developed DL models are able to produce good results in the vast majority of the test samples within considerably lower computation times. Nevertheless, the fact that the models still do not perform well against some indistinguishable instances makes it unfeasible to consider its direct utilization in practice.

Keywords: finite-element, structural engineering, convolutional neural networks, deep learning

Resumo

Métodos de cálculo da mecânica estrutural podem ser computacionalmente exigentes e demorados. Um exemplo disso é o método de elementos finitos (FEM), que se trata do método mais utilizado para resolver este género de problemas.

Com vista a ampliar as abordagens disponíveis para resolver tais problemas, o presente trabalho explora uma abordagem baseada em *deep learning* para prever mapas de esforços de flexão em estruturas planas 2D. Essas estruturas podem estar sujeitas a diferentes padrões e magnitudes de cargas externas perpendiculares ao plano, geometria e condições de apoio.

Para o efeito, foi primeiramente desenvolvido um *dataset* de modelos estruturais de elementos finitos por meio de *software* comercial, seguido do desenvolvimento, treino e teste dos modelos de *deep learning*. Este estudo é realizado utilizando redes neurais convolucionais (CNN), beneficiando-se de sua especialização em processar dados que tem uma topologia em forma de grelha.

Os resultados mostram que os modelos DL desenvolvidos produzem bons resultados na grande maioria das instâncias de teste, com tempos de processamento consideravelmente menores que os modelos de elementos finitos. No entanto, o facto de os modelos não apresentarem bom desempenho frente a alguns exemplos indistinguíveis torna inviável considerar a sua utilização directa na prática.

Keywords: elementos finitos, engenharia de estruturas, redes neuronais convolucionais, deep learning

Acknowledgements

Aos meus orientadores, Prof. Vera Miguéis e Prof. José Luís Borges, por todo o apoio e orientação nesta longa jornada, desde o processo de escolha do tema até às revisões finais.

À Joana, pelo incansável apoio, compreensão e paciência ao longo de todo o Mestrado, o meu sincero agradecimento. Sem a tua ajuda não teria sido possível.

Aos meus pais, ao meu irmão e à minha restante família, o meu agradecimento pelo exemplo de trabalho e humildade que sempre me transmitiram.

Aos meus colegas da primeira fornada do MECD, cuja união e entreeajuda facilitaram muito esta travessia ao longo dos últimos dois anos.

Por fim, aos meus amigos, que de uma forma ou de outra sofreram os danos colaterais consequentes da minha entrega a este trabalho e ao Mestrado, um enorme agradecimento.

Nuno Gaspar

Contents

List of Figures

List of Tables

Abbreviations

AI	Articifial Intelligence
ANFIS	Adaptive Neuro-Fuzzy Inference System
ANN	Artificial Neural Network
BA	Boosting Algorithm
BN	Batch Normalization
CNN	Convolutional Neural Network
SCSE	Concurrent Spacial and Channel Squeeze & Excitation
DL	Deep Learning
DT	Decision Tree
E&C	Engineering and Construction
ELU	Exponential Linear Unit
FEM	Finite Element Method
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IQR	Inter-Quartile Range
GAN	Generative Adversarial Network
GFLOPS	Giga Floating-Point Operations per Second
GPU	Graphics Processing Unit
KDE	Kernel Density Estimate
LTU	Linear Threshold Unit
MAE	Mean Absolute Error
ML	Machine Learning
MSE	Mean Squared Error
NN	Neural Network
PAE	Peak Absolute Error
PPAE	Percentage Peak Absolute Error
PReLU	Parametric Leaky Rectified Linear Unit
RA	Regression Analysis
RBFNN	Radial Basis Function Neural Network
ReLU	Rectified Linear Unit
RMSE	Root Mean Squared Error
RReLU	Randomized Leaky Rectified Linear Unit
RF	Random Forests
SE	Squeeze-and-Excitation
SGD	Stochastic Gradient Descent
SHM	Structural Health Monitoring
SVM	Support Vector Machine
VB6	Visual Basic 6.0
VGG	Visual Geometry Group
WMAPE	Weighted Mean Absolute Percentage Error

Chapter 1

Introduction

1.1 Context and Motivation

The finite element method is a popular tool for the numerical solution of partial differential equations in engineering which origin is not consensual, although most of the engineering community assigns it to Clough [13], back in 1960 [11]. It was originally developed for solving problems in solid-state mechanics (namely, plate bending), but it found application in all areas of computational physics and engineering since then. Simply put, the main idea behind the method is to discretize the continuum into a mesh of simple geometric elements such as triangles or quadrilaterals [53].

Much like an image, the tighter the mesh, the more accurate is the approximation. This undeniable resemblance suggests that the success that convolutional neural networks (CNN) achieved so far in many tasks related to image processing, have much potential to go beyond that domain.

Given this similarity, considering a rectangular panel discretized into finite elements, in comparison to a typical image, the "resolution" of this "image" is the number of finite elements along both orthogonal directions. Instead of color channels, in this approach, each channel will contain fundamental inputs to perform the stress calculations, and of course, the ground truth. These inputs are mainly related to: geometry (whether an element belongs to the structure of it is in a void region), loads (magnitude of the load applied to each element) and supports (if an element is supported or not).

With this work it is intended to promote the approximation between structural engineering and artificial intelligence, broaden horizons with regard to calculation methods, show the transversality of deep learning applications and suggest its adoption in problems of high complexity.

1.2 Objectives

The main goal of this project is to design an effective Convolutional Neural Network, capable of predicting the internal bending moments of linear elastic and non-linear shell finite element models subject to out-of-plane loads.

To achieve such goal, the following objectives are defined:

- Produce a dataset of 18,000 linear elastic finite element models, with different configurations of geometry, loads and supports among each other, calculated with specific software. The same models (and their input features) will also be used to solve the problem with non-linear supports, which leads to a more complex and time consuming calculation method;
- Assess the prediction performance of the resulting finite-element stresses of state-of-the-art architectures, using train, test and validation subsets from the previously described dataset;
- Develop and fine-tune CNN models that best suit the problem;
- Analyze the results, compare the results among different models and evaluate their feasibility for solving real structural engineering problems.

1.3 Document Structure

Besides the present introductory chapter, Chapter 2 presents a brief introduction to Deep Learning (DL) — and more specifically Convolutional Neural Networks (CNNs) — as well as a literature revision of AI research applied to structural engineering. Chapter 3 presents the generated dataset and all its inherent considerations. All information regarding the experimental methodology and results analysis is addressed in Chapters 4 and 5, respectively. Finally, conclusions and future research directions are drawn in Chapter 6.

Chapter 2

State Of The Art & Related Work

2.1 Deep Learning

Deep learning (DL) allows computational models of multiple processing layers to learn and represent data with multiple levels of abstraction, mimicking how the brain perceives and understands multimodal information [65]. DL is part of a family of machine learning (ML) methods based on artificial neural networks (ANN) with the particularity that it consists of several layers in between input and output layer, allowing for many stages of non-linear information processing units that are exploited for feature learning and pattern classification [3]. Figure 2.1 shows a taxonomy of Artificial Intelligence (AI), in which neural networks and deep learning are included.

One of the reasons why DL is so popular is because of its needlessness of "manual" feature extraction. ML algorithms (such as: Decision Trees, SVM, Naïve Bayes Classifier, etc.) presuppose prior feature extraction on raw data, which is usually complex and requires detailed domain knowledge. DL techniques replace this engineering step by learning the best features, in a process called "representation learning". This difference is illustrated in Figure 2.2.

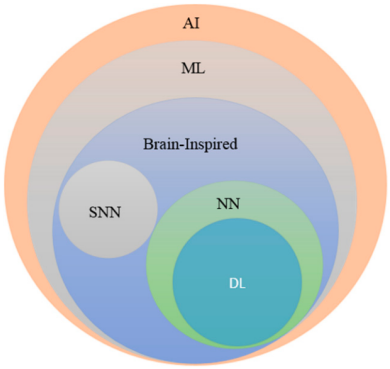


Figure 2.1: Taxonomy of AI. AI: Artificial Intelligence; ML: Machine Learning; NN: Neural Networks; DL: Deep Learning; SNN: Spiking Neural Networks . Extracted from [3].

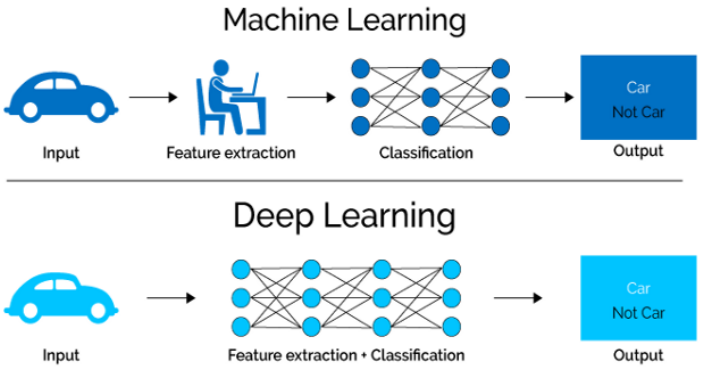


Figure 2.2: Feature Extractions: Machine Learning *vs.* Deep Learning. Extracted from [49].

2.1.1 Historical Background

Deep learning dates back to the 1940s. It only appears to be new because it was relatively unpopular during several years and because it has gone through many different names along time, only recently being called "deep learning" [24].

Goodfellow *et al.* [24] describe a "three wave" development of deep learning. The first wave, between 1940s and 1960s, when deep learning was called *cybernetics*, consisted of early neural networks. In 1943, McCulloch and Pitts [45] tried to understand how the brain could produce highly complex patterns by using interconnected basic cells. The authors presented the first artificial neural network architecture, which consisted of a simplified computational model of how biological neurons might work together in animal brains to perform complex computations using *propositional logic* [22]. In 1957, the *perceptron* was presented by Frank Rosenblatt [54]. The *perceptron* is one of the simplest artificial neural network architectures, based on a slightly different neuron, called a Linear Threshold Unit (LTU), in which the inputs are numbers (instead of on/off values) and each input connection is associated with a weight. An LTU can be used for simple linear binary classification. It processes a combination of inputs, if the result exceeds a threshold it outputs the positive class, otherwise it outputs the negative class [22]. Figure 2.3 illustrates the workflow of a perceptron.

Despite looking very promising then, the first major dip of neural networks started from the publishing, in 1969, of a book called "Perceptrons", by Marvin Minsky and Seymour Papert [46], in which the authors identified a number of serious weaknesses of the linear model (single Perceptron), especially the fact that they are incapable of solving some trivial problems like the XOR (Exclusive or) classification problem [22]. These limitations basically stopped all research in neural networks for 15 years and this period is referred to as an “AI Winter” [62].

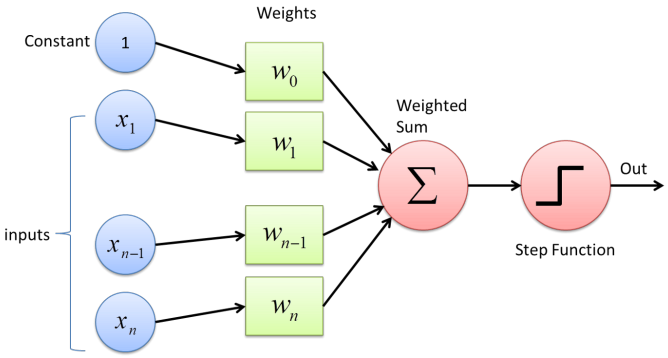


Figure 2.3: Workflow of a perceptron. Extracted from [59].

The second wave started with the *connectionist* approach, from 1980s to 1990s, with a concept called *back-propagation* proposed by Rumelhart *et al.* [56]. *Back-propagation* brought a key element missing from the Rosenblatt era, that was the ability to train a multi-layer neural network, thus to solve non-linear problems such as the XOR problem mentioned before. At this point, despite being able to solve many problems, deep networks were generally believed to be very difficult to train, specially because they were too computationally expensive to allow much experimentation with the hardware available at the time. This difficulty led to a second "AI Winter", during which little research was carried out [24].

The third wave, termed *Deep Learning*, that started in 2006 and extends until nowadays, began with a research, from Hinton *et al.* [28], which concluded that the greedy layer-wise learning procedure could be used to find a good initialization for a joint learning procedure over all the layers, and that this approach could be used to successfully train even fully connected architectures. [24]. This discovery has put Deep Learning back onto the table, and led to the hype that we are still experiencing nowadays. Figure 2.4 shows a timeline with the main events of Deep Learning research.

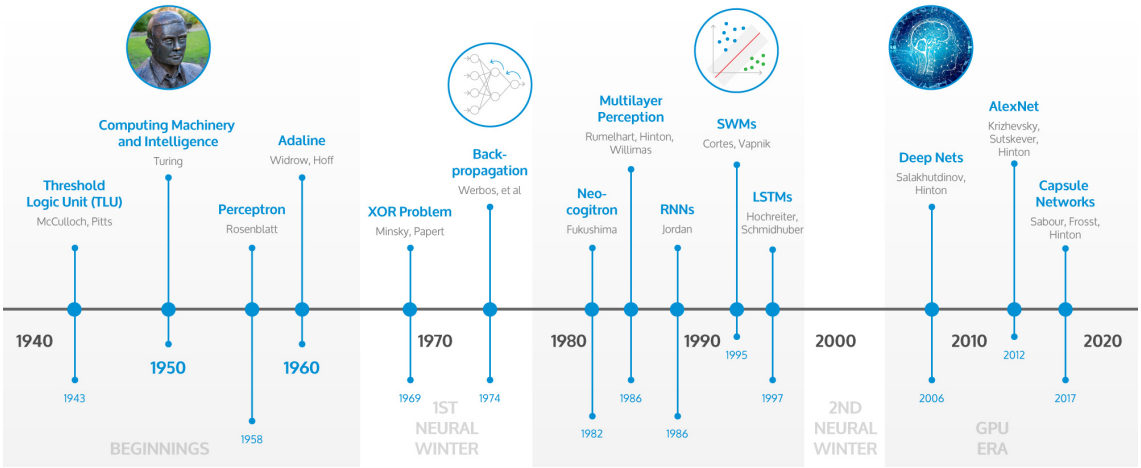


Figure 2.4: Deep Learning Timeline. Adapted from [19].

2.1.2 Background Knowledge

Neural networks are composed of an input layer, hidden layers, and an output layer. The adoption of two or more hidden layers makes it a *deep network* or a *multilayer perceptron*. The input layer transmits the data to the output layer, passing through the hidden layers, where the computations are performed. These computations are not visible to the user, hence the name "hidden" [1].

In a single-layer neural network, the training process is straightforward because the loss function can be computed as a direct function of the weights, which allows easy gradient computation. With multi-layer networks, the problem is that the loss is a complex composition function of the weights in earlier layers. The gradient of a composition function is computed using the *back-propagation algorithm* [56]. Simply put, *back-propagation* leverages the chain rule of differential calculus, which computes the error gradients in terms of summations of local gradient products over the various paths from a note to the output. Wrapping up, the forward phase is required to firstly compute the output values and the local derivatives at various nodes, while the backward phase (*back-propagation*) is required to accumulate the products of these local values over all paths from the node to the output [1].

Some additional important concepts inherent to deep neural networks are briefly presented below.

- **Weight Initialization**: Successfully training a CNN using a gradient-based method without a good initialization is nearly impossible. An important principle to keep in mind is that, weights of the neurons among the network must be different. If they all have the same value, neurons in the same layer will have identical gradients, leading to redundant update rules [2].

  Traditionally, weight initialization involves the use of small random numbers. However, over the last decade, more specific heuristics have been designed to take advantage of additional information about each layer, in order to render the training process more effective, such as the activation function to be employed and the number of inputs and outputs of the node [8].

  In 2010, Xavier Glorot and Yoshua Bengio [23], argued that for the signal to flow properly, it is needed that the variance of the outputs of each layer is equal to the variance of its inputs and that the gradients have equal variance before and after flowing through a layer in the reverse direction. Despite not being possible to guarantee both principles unless the layer has an equal number of inputs and outputs, the authors found a good compromise that has proven to work very well in practice [22]. This weight initialization technique was discovered to have problems when used to initialize networks that use the ReLU activation function. To overcome such limitation, Kaiming He *et al.* [26] proposed a method which would turn out to be commonly known as "He" initialization. This method is widely used nowadays, it is even the default method for initializing ReLU layers in some deep learning frameworks, such as PyTorch[1].

[1] https://www.pytorch.org/

- **Loss Function**: The general idea of gradient descent is to tweak parameters iteratively in order to minimize a loss or cost function. In deep learning, typically by its minimization, it is used to measure how close the model is to the ground truth [22]. The choice of a suitable loss function depends heavily on the task to be solved. The most common loss functions for regressions problems are: Mean Absolute Error (MAE), Mean Squared Error (MSE) or Root Mean Squared Error (RMSE), whereas for classification problems the most commonly used are: Binary Crossentropy, Categorical Crossentropy or Sparse Categorical Crossentropy [52]. Besides these typical loss functions, custom ones may also be developed, according to the problem particularities.

- **Activation Function**: An activation function, sometimes called "transfer function", defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of a network. The output range of the activation function may either be limited ("squashing function") or unlimited. Many activation functions are non-linear, but also typically differentiable (the first-order derivative can be calculated for a given input value). This fact is crucial to allow for the backpropagation of the error [7].

  Usually all hidden layers of a network use the same activation function. The output layer, depending on the range of values to predict, may use a different activation function.

  The most commonly used activation function for hidden layers is the Rectified Linear Unit (ReLU) [7]. Its popularity is mostly due to the fact that it is easy to implement and effective at overcoming the limitations of other popular activation functions such as Sigmoid and Tanh (Hyperbolic Tangent), like the vanishing/exploding gradient problem, since it does not saturate for positive values. ReLU, like all other activation functions, also has problems of its own, such as the "dying ReLU" problem. This problem is related to the fact that, during training, some neurons effectively die, meaning they stop outputting anything other than 0. When such happens, the neuron is not likely to be restored, since the gradient of the ReLU function is 0 when its input is negative [22]. This problem led to the development of a variant of ReLU, called LeakyReLU, which instead of being 0 for negative inputs, the output will be a small negative value given by the hyper-parameter α, which is the slope of the function for negative values, hence the term "leaky". Besides LeakyRelu, other variants of ReLU were developed to solve this problem, although sharing the same concept. For example, *Randomized Leaky ReLu* (RReLU), which is similar to LeakyReLU with the difference that α is determined randomly within a given range throughout training, and *Parametric leaky ReLU*, where α is learned during the training process. More recently, a paper presented by Djork-Arné Clevert *et al.* [12] proposed a new function called *Exponential linear unit* (ELU), which is also part of this family, but this time employing the exponential function. This function not only attempts to solve the vanishing gradient problem but it is also smooth along the entire function, which helps speed up gradient descent, since it does not bounce around 0^- and 0^+. Like Leaky ReLU it also has an α hyperparameter which defines the

value for which the ELU function tends when the input is a large negative number. Comparing to the previous ones, it is slower to compute, but it is expected to compensate due to its faster convergence rate [22]. Figure 2.5 shows a plot of ReLU, Leaky ReLU and ELU activation functions.

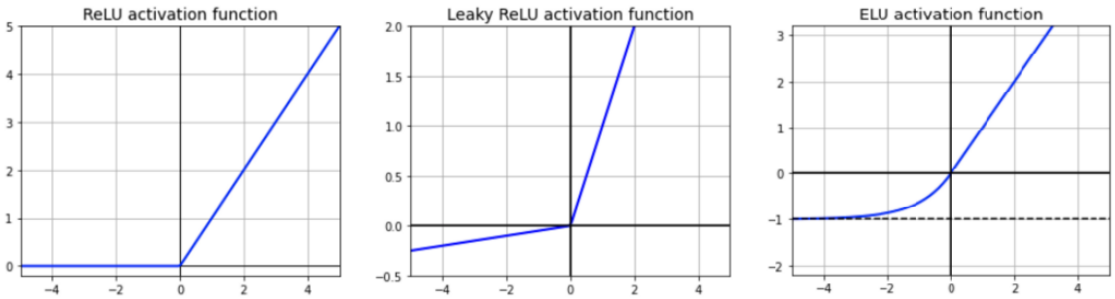


Figure 2.5: ReLU, Leaky ReLu (with α=0.01) and ELU (with α=1.0) activation functions plots. Adapted from [10]

Depending on the problem, the previously introduced activation functions may also be applied to the output layer, however they are not usually suitable for classification problems. In the presence of a binary classification problem, Sigmoid (or logistic function) is a typically used activation function [7], which will output a value between 0 and 1, that can be seen as the probability of belonging to the given class. In multi-class problems, Softmax (which is a generalization of the logistic function to multiple dimensions) is commonly used. Figure 2.6 shows a plot of the Sigmoid function.

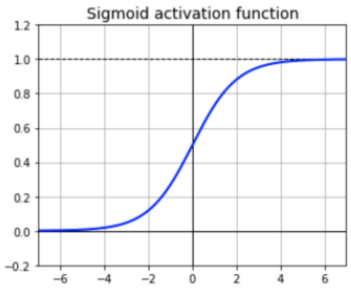


Figure 2.6: Sigmoid activation function plot. Adapted from [10]

- **Learning Rate**: Stochastic gradient descent has an hyper-parameter called *learning rate* (or step size), usually denoted by α, which sets the rate at which a neural network updates its weights in response to a loss. If a learning rate is chosen properly, it is expected to see the loss function decreasing at each iteration at the beginning of the training process. However, with a constant learning rate, at a certain point close to a local minimum, the algorithm may fluctuate near it and never exactly converge. This problem may be related to a high learning rate. Generally it is a good practice to decay the leaning rate over time [2]. There are several

ways of achieving this. The most common decay functions are the step decay schedule (in which the learning decays by half after a given number of epochs) and the exponential decay schedule (as the name suggests, the learning rate decays at an exponential rate). Examples of these are depicted in Figure 2.7.

There are many other learning rate schedulers besides these, not necessarily descending-only, but also cyclic, such as Cosine Annealing [43]. No scheduler can be said to be the absolute best, because it depends on the problem to solve and even on other hyper-parameters.

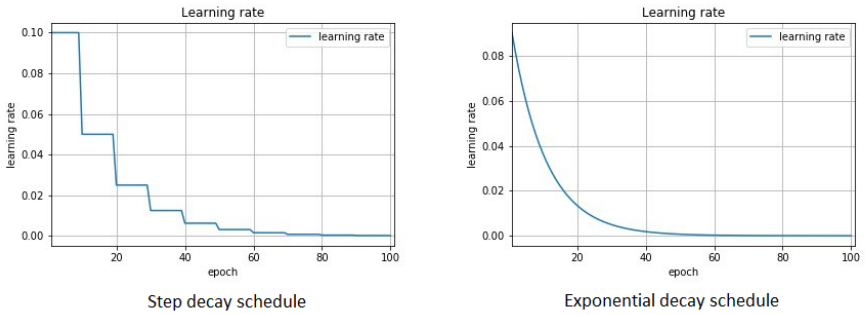


Figure 2.7: Step decay and exponential decay learning rate schedules. Adapted from [39]

- **Optimizer**: The optimizer aims at minimizing the loss function. For that, Gradient Descent is still the most common algorithm, although adaptive methods, such as Adam [37], Adadelta [68], Adagrad [17] and Adabound [44], are progressively gaining traction. Among the community, Adam is considered to be the best suit for general purpose problems. [22]

- **Regularization**: Regularization techniques are usually applied in order to prevent the model from *overfitting*. Technically, a model is said to be *overfitting* if it simply "memorizes" the training set and does not learn to generalize on the test and validation sets.

  The most popular regularization technique for deep neural networks is called *dropout* and it was proposed by Hinton *et al* [29]. With this technique, at every training step, every neuron (including the input neurons but excluding the output neurons) has a probability p of being temporarily ignored during this training step, although it may be active during the next step and so on. It is known that this technique slows down convergence, but it also has proven to work well in practice [22].

2.2 Convolutional Neural Networks (CNN)

Convolutional neural networks are a particular kind of neural networks for processing data that has a grid-like topology. Examples include time-series data, that can be seen as a 1D grid taking samples at regular time intervals, image data, that can be thought of as a 2D grid of pixels, and video

data, that can be seen as a stack of 2D frames. The name "Convolutional Neural Networks" indicate that the network employs a linear mathematical operation called "convolution" [24], which is further described in section 2.2.2.

2.2.1 Historical Background

CNNs initially emerged from the study of the brain's visual cortex. Hubel and Wiesel (1981 Nobel prize laureates in Physiology or Medicine) performed a series of experiments on cats [33, 32]. The authors concluded that many neurons in the visual cortex have a "small local receptive field" that reacts only to visual stimuli located in a limited region of the visual field. These neurons may overlap, and together they cover the whole visual field. It was also showed that some neurons react only to images of horizontal lines, while others react only to lines with different orientations, even if they have the same receptive field. Additionally, the authors concluded that some neurons have larger receptive fields, and that those react to more complex patterns, which are the outputs of neighboring lower-level neurons, as illustrated in Figure 2.8 [22].

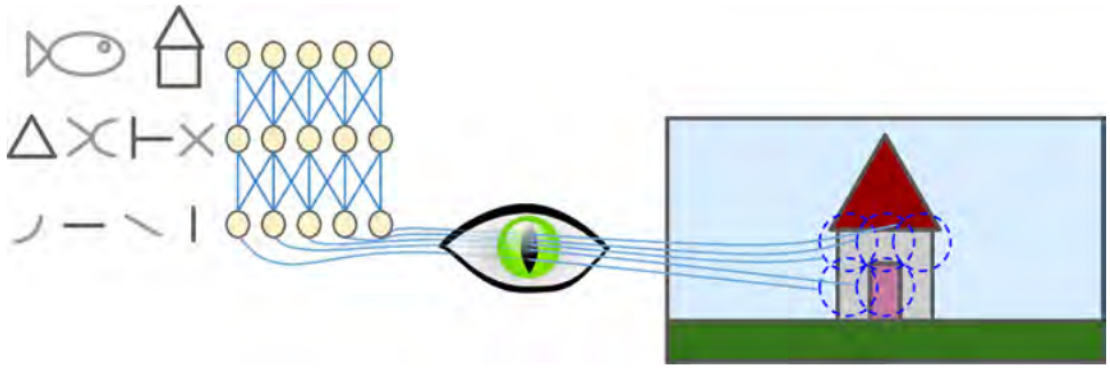


Figure 2.8: Local receptive fields in the visual cortex and incremental complexity of patterns throughout neuron layers. Extracted from [22].

These conclusions were the inspiration to the neocognitron model proposed by Fukushima [20], later, in the 1980s. The neocognitron model includes components named "S-cells" (simple cells) and "C-cells" (complex cells). These are not biological cells, but rather mathematical operations [16]. The "S-cells" sit in the first layer of the model, and are connected to the "C-cells" which sit on the second layer of the model. The main idea, following the conclusions drawn by Hubel and Wiesel, is to capture the "simple-to-complex" concept and turn it into a computational model for visual pattern recognition [16].

On its turn, inspired by the neocognitron, the first work on modern convolutional neural networks occurred in 1998 by Yann LeCun *et al.* [41]. LeNet-5 was a pioneering 7-level CNN model that was able to recognize hand-written numbers digitized in 32x32 pixel gray-scale input images [15]. This architecture and the following modern architectures are further described in 2.2.3.

From the late 1990s up to 2000, various improvements in CNN learning methodology and architecture were performed to make CNN scalable to large, heterogeneous, complex, and multi-class problems [36]. In the early 2000's there was some stagnation in the development of new CNN architectures, especially because its training was not effective in converging to the global minima of the error surface. Thus, CNNs started to be considered as a less effective feature extractor compared to handcrafted features [36].

The revival of CNN's began from 2006 to 2011, as significant efforts have been made to tackle the CNN optimization problem. The use of activation functions other than Sigmoid, such as ReLU, tanh, etc. and the use of graphical processing units (GPU) are among the most significant tweaks that led the re-rise of the CNN research, making it a hot research topic even nowadays.

2.2.2 Background Knowledge

CNNs and DNNs share essentially the same layers, except for two characteristic layers that make CNNs exceptional. These are the "Convolutional layer" and the "Pooling layer", that are briefly described in this subsection.

- **Convolutional Layer**: As stated at the beginning of this chapter, CNNs are a type of neural network especially designed to process data which has a grid-like topology. The convolutional layer takes advantage of each pixel's neighbourhood by computing the dot product between each filter (or kernel) and the input at every position. Each filter, which dimensions are an hyper-parameter, is learned by the network and will be suited to extract the features that help solving the network's task. Figure 2.9 shows a visual representation of the convolutional operation.

  Stride and padding are two hyperparameters that need to be defined when designing a convolutional layer. The result of the convolution operation is a new matrix (feature map), whose dimensions depend on these hyperparameters. When applying convolution to an input as is (*i.e.* with no padding), with a kernel which dimension is greater than 1, the operation will result in a slightly shrinked image – depending on the dimension of the kernel – because the number of operations is smaller than the number of rows and columns. Therefore, padding is generally used in order to prevent this phenomenon from happening, enabling the output to have the same dimensions as the input. Stride, in its turn, specifies the distance (in pixels) between two receptive fields. When stride is greater than 1, the output will be smaller than the input. Figure 2.10 evidences the effect of different striding values on an input with no padding, by highlighting the distance among the coloured boxes. Figure 2.11 shows the difference between a non-padded input and a padded input subject to the same convolution operation, being the latter able to preserve the same dimensions in its input and output.

  Convolutional layers allow for each neuron to be connected not to every single pixel in the input image – as a fully connected layer would – but only to its receptive field. In turn, each neuron in the following convolutional layer is connected only to neurons within a restrained field of the previous layer and so on, like what happens in the brain's visual cortex. [22]

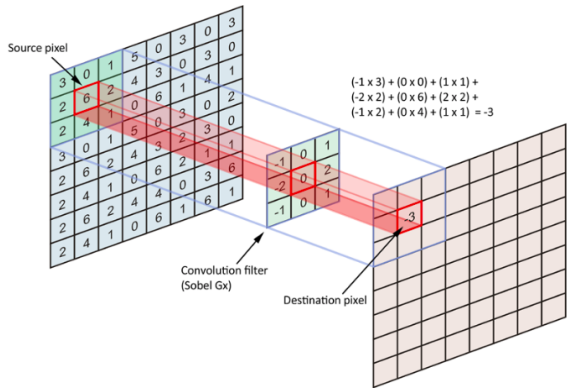


Figure 2.9: Visual representation of the convolutional operation. Extracted from [9].

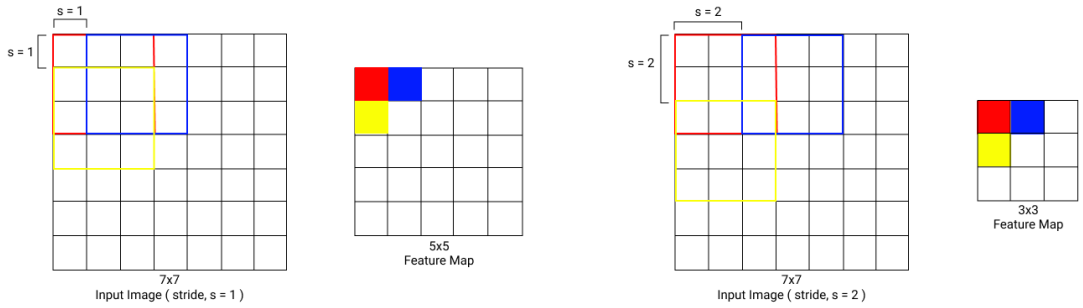


Figure 2.10: Visual representation of different striding values. Extracted from [57].

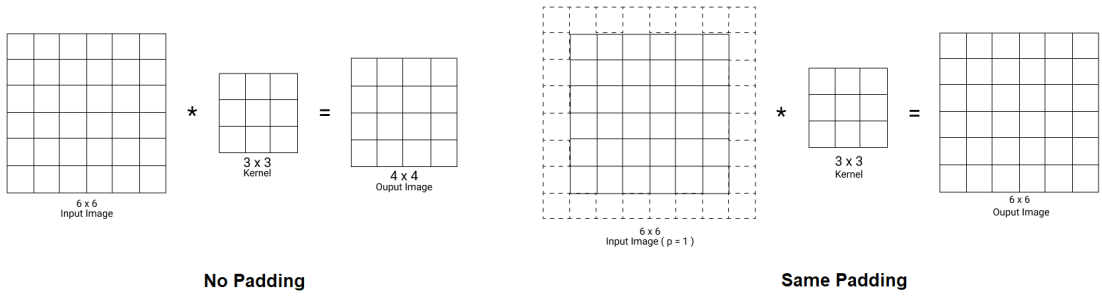


Figure 2.11: Example of the padding effect on convolution operations. Adapted from [57].

- **Pooling Layer**: The main goal of a pooling layer is to subsample in order to reduce dimensionality. Contrary to convolution layers, what pooling layers do is just aggregating inputs, thus having no parameters. The most common pooling types are max pooling and average pooling, obviously named after each aggregation function – maximum and mean, respectively. Like convolutional layers, pooling layers are defined by a window size, stride and

padding. Figure 2.12 shows an example of max pooling and average pooling applied to the same input. [22]

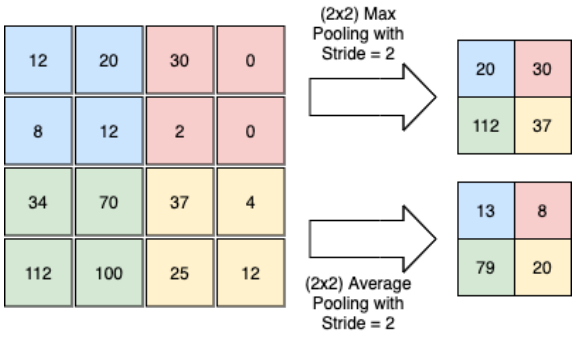


Figure 2.12: Example of a max and average pooling with a window size of 2x2 and stride 2. Extracted from [18].

2.2.3 Architectures

Fuelled by previous successful architectures and the increasing computational power, CNN architectures continued to evolve throughout the years. The evolutionary history of the most famous CNN architectures until 2018 is shown in Figure 2.13. A brief description of the most famous CNNs is also presented below.

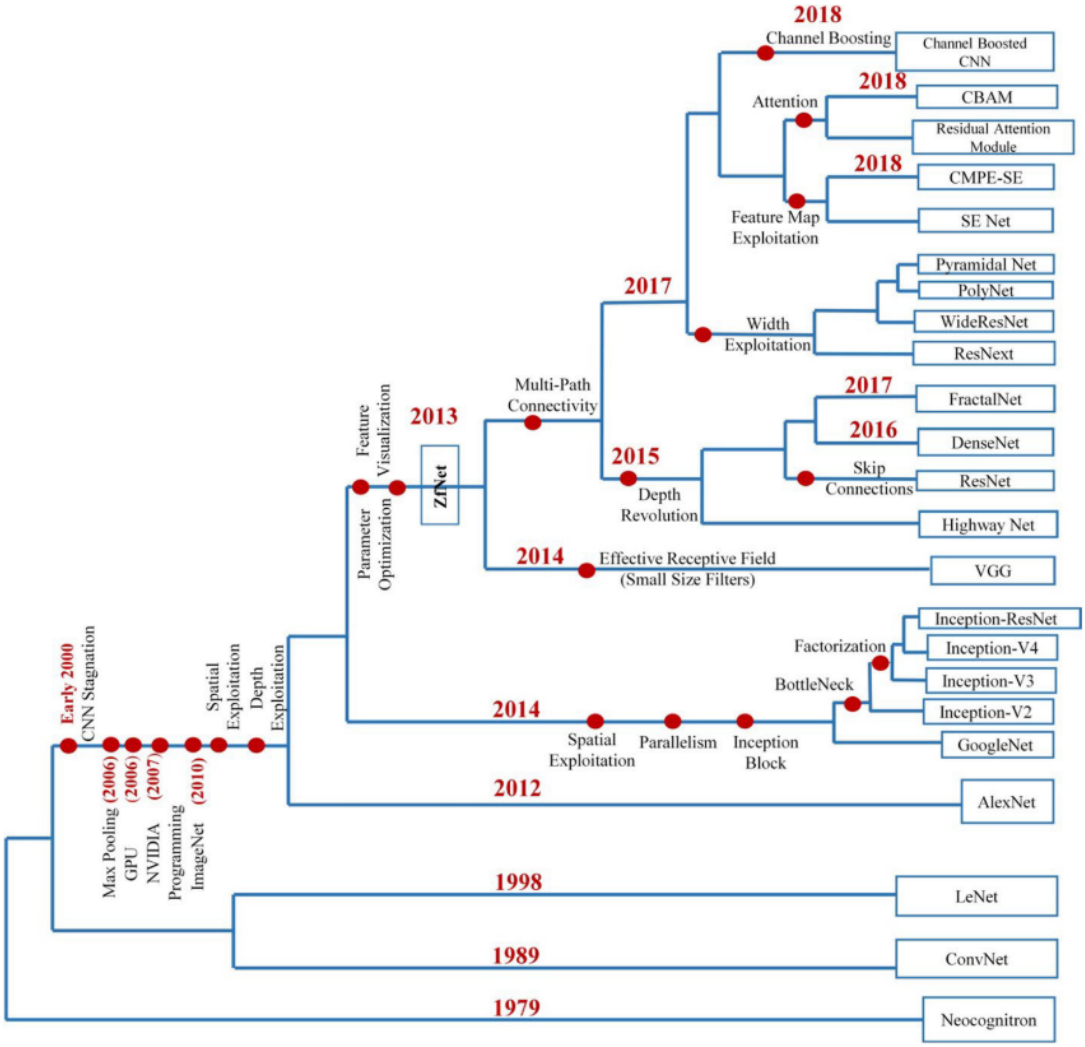


Figure 2.13: Evolutionary history of deep CNNs showing architectural innovations. Extracted from [36].

- **LeNet-5** [41]: As previously stated, developed by Yann LeCun, LeNet-5 is one of the most famous CNN architectures. LeNet is a feed-forward network, consisting of five interleaved convolutional and pooling layers, followed by two fully connected layers. Its input has a size of 32 x 32 and it uses the hyperbolic tangent as the activation function.

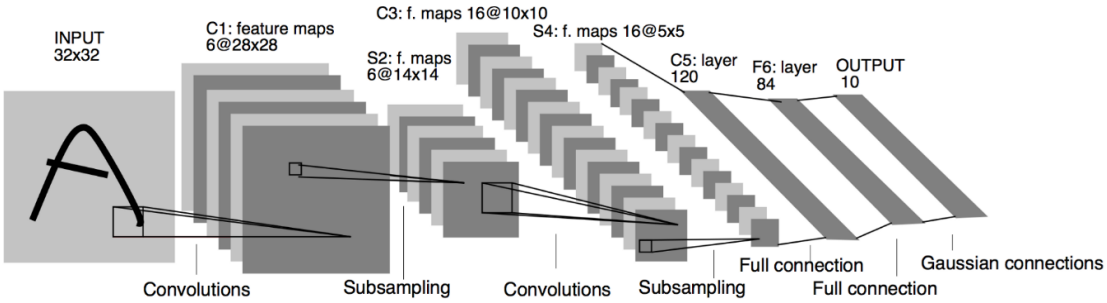


Figure 2.14: LeNet-5 architecture. From C1 to C5 the interleaved convolutional/pooling layers, F6 and OUTPUT are two fully connected layers. Extracted from [41].

- **AlexNet** [38]: Developed by Alex Krizhevsky, the AlexNet architecture won the 2012 ImageNet Large Scale Visual Recognition Challenge[2] (ILSVRC) by a large margin: 17% top-5 error rate, while the second best achieved only 26% [22]. This architecture is similar to LeNet-5, but larger and deeper. The main differences are its input size of 224 x 224, that it stacks convolutional layers directly on top of each other, the use of a larger kernel in the first layer (11 x 11), the use of ReLU activation function for the hidden layers – improving the convergence rate by alleviating the problem of vanishing gradient – and the use of Softmax for the output layer.

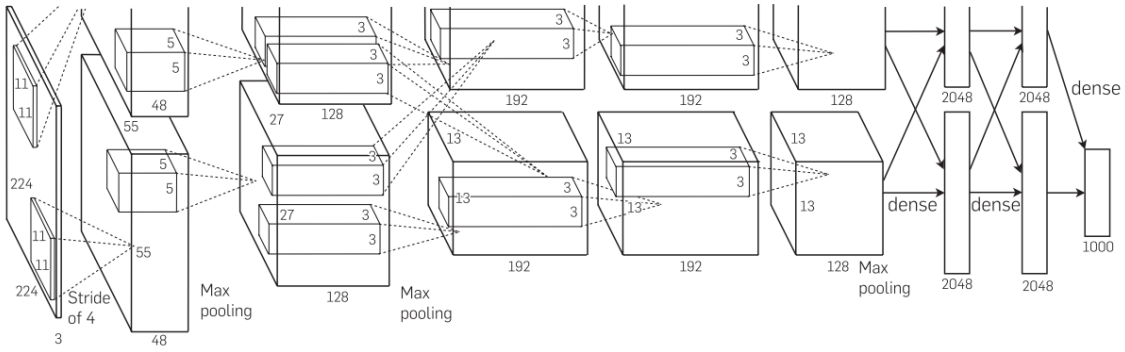


Figure 2.15: AlexNet architecture. The delineation of responsibilities between two GPUs is explicitly shown. The GPUs communicate only at certain layers. Extracted from [38].

- **VGG-16** [60]: Developed by Karen Simonyan and Andrew Zisserman, it was one of the top performing architectures of ILSVRC 2014 together with GoogLeNet. Like AlexNet it uses an input size of 224 x 224, but instead of larger kernels followed by pooling, it uses multiple 3x3 kernel filters (which is the smallest size to capture the notion of left/right, up/down, centre) one after another. By doing such, the authors not only increase the effective receptive field throughout convolutional layers, but also make the decision function far more

[2] https://www.image-net.org/

discriminative, by using multiple non-linear rectification layers. This depth is evidenced by Figure 2.16 and Figure 2.17. Besides VGG-16 with 16 weight layers, there are other variants of VGG, such as VGG-11 and VGG-19 with 11 and 19 weight layers respectively.

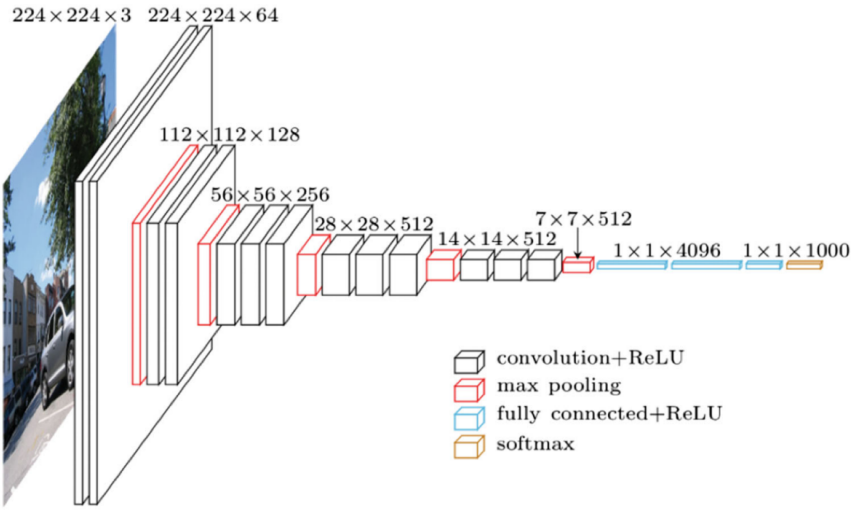


Figure 2.16: VGG-16 architecture. Extracted from [47].

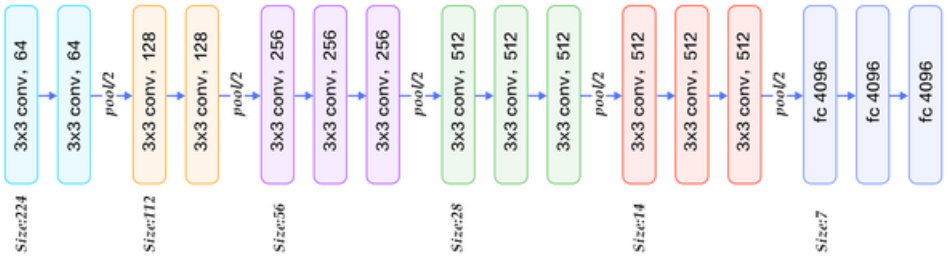


Figure 2.17: VGG-16 layers description. Extracted from [67].

- **GoogLeNet** [61]: Introduced by Christian Szegedy *et al.* from Google Research, it won the ILSVRC 2014 challenge by lowering the top-5 error rate below 7% [22]. This architecture introduced a new concept that the authors called the "inception module", which incorporates multi-scale convolutional transformations using split, transform and merge, allowing GoogLeNet to use parameters more efficiently. This module encloses filters of different sizes (1x1, 3x3, and 5x5) to capture special information at different scales. 1x1 filters work as bottleneck layers in order to perform dimensionality reduction. The architecture of this module is depicted in Figure 2.18. The whole architecture includes 9 inception modules.

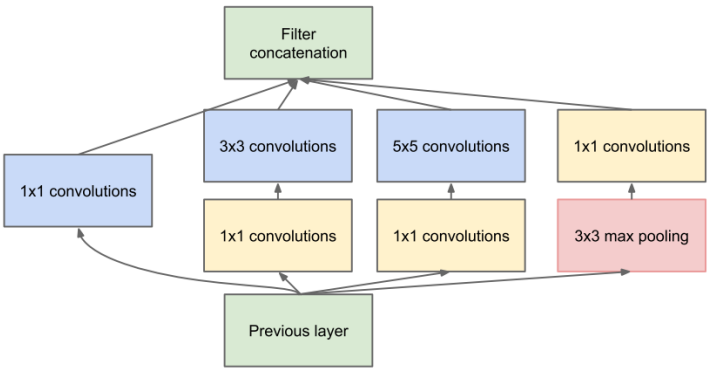


Figure 2.18: GoogLeNet inception module. Extracted from [61].

- **ResNet** [27]: The winner of ILSVRC 2015 was a network called ResNet, developed by Kaiming He *et al.* [27]. The main novelty with this network was the introduction of "skip connections" (also called "shortcut connections"). These are connections that link a certain layer with another one some levels ahead in the chain. The goal is, when modelling a target function $h(x)$, by adding the input x to the output of the network, then the network will be forced to model $f(x) = h(x) - x$ instead of $h(x)$. Figure 2.19 shows a diagram of a residual block. This manoeuvre speeds up training considerably every time the target function is close to the identity function – which is often the case [22].

  Besides ResNet-50, the most typically used, several other architectures, with different number of layers, were also presented, namely 18, 34, 101 and 152 layers. The constitution of the residual layers also varies among these architectures as depicted in Figure 2.20. ResNet-18 and ResNet-34 encapsulates residual blocks composed of two 3x3 convolutional layers with batch normalization and ReLU activation function, whereas the 50-layer architectures and above are composed of residual blocks with a first 1x1 convolutional layer (which acts as a bottleneck, like in GoogLeNet), the 3x3 convolutional layer and finally another 1x1 convolutional layer.

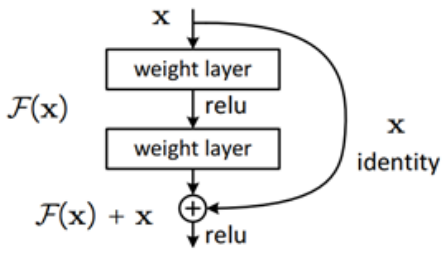


Figure 2.19: Residual learning block. Extracted from [27].

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2.20: Resnet architectures. Two different configurations for the residual blocks are adopted: one for ResNet with 18 and 34 layers, another from Resnet-50 onwards. Extracted from [27].

- **ResNeXt** [66]: Also known as "Aggregated Residual Transform Network", ResNeXt, one of the top performers of ILSVRC 2016 [66] is an enhanced version of the Inception network. It exploits the concept of split, transform and merge in a powerful but simple way by introducing an additional dimension, called "Cardinality", that refers to the size of the set of transformations. Picking up from the improvements that Inception made to other previous conventional CNNs, ResNext improves on the fact that each layer needs to be customized separately due to the use of diverse spatial embeddings (such as the use of 3x3, 5x5, and 1x1 filter) in the transformation branch, by using residual learning to improve the convergence of deep and wide networks [66]. Figure 2.21 depicts the ResNeXt building blocks.

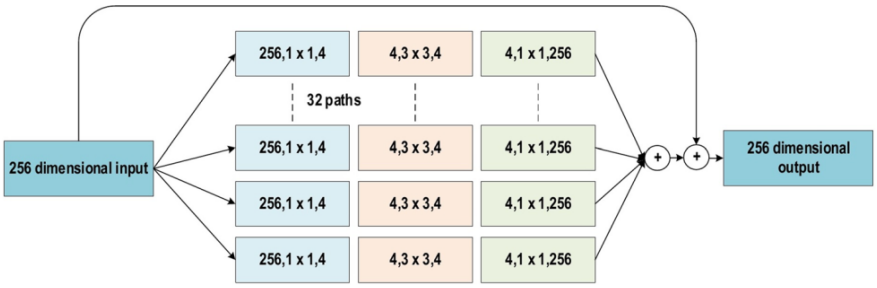


Figure 2.21: Basic block diagram for the ResNeXt building blocks. Extracted from [4].

- **SENet** [30]: Squeeze and Excitation Network (SE-Network), winner of the ILSVRC 2017 classification challenge with a 25% improvement on the top-5 error relatively to the previous year, was reported by Hu *et al.* [30]. The authors proposed a new block, named SE-block, for the selection of feature-maps (*a.k.a.* channels). Regular CNN architectures presented so far weight each of its channels equally when creating the output feature maps.

SENets change this principle by adding a content aware mechanism to weight each channel adaptively. Simply put, this could mean adding a single parameter to each channel and giving it a linear scalar value, stating how relevant each one is [51]. Figure 2.22 depicts the architecture of a SE-block.

Squeeze-and-Excitation is a concept that can be adopted to any other existing CNN architecture. Hu *et al.* [30] demonstrated, as an example, that by adding SE-blocks to ResNet-50, the model not only outperforms the original ResNet-50, but delivers almost the same performance of ResNet-101 with an increase of only 0.26% computational cost (in terms of GFLOPs) over the original ResNet-50.

However, when a SE-block is applied in ResNet, the identity mapping does not take into account the input of the channel-wise attention of the residual flow. This reduces the impact of SE-block and makes ResNet information redundant. In order to alleviate this redundancy Hu *et al.* [31] designed a new network called Competitive Inner-Imaging Squeeze and Excitation for Residual Network, also known as CMPE-SE, which models the competitive relation from both the residual and identity mapping based feature-maps. The results of CMPE-SE-ResNet-50 were "slightly superior" of those of the SE-ResNet-50, improving the top-1 error rate by 0.5% and the top-5 error rate by 0.27% [31].

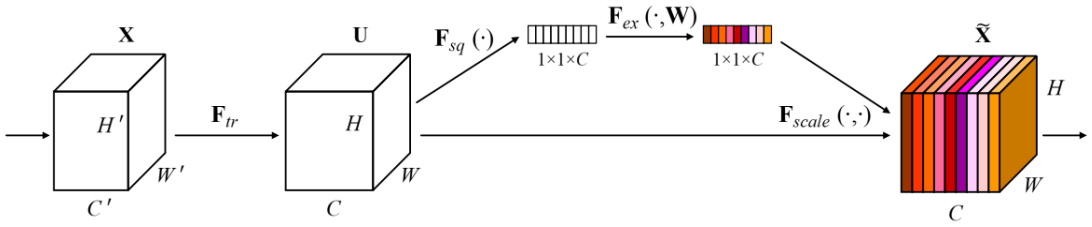


Figure 2.22: SE-block. Extracted from [30].

2.2.4 Current Research Status and Future Directions

CNNs have been extensively applied to different ML tasks, such as: computer vision, speech processing, natural language processing, object detection and segmentation, image classification, video processing, among others. The main advances in CNNs can be categorized in different ways, including activation functions, loss functions, optimization, regularization and innovations in architecture.

Currently, one of the paradigms of research in CNN architectures is the development of new and effective block architectures that can work as an auxiliary learner [4, 36]. Apart from the architectures presented in the previous section, specially SENet, one example of recent research is the Res2Net block. According to Gao *et al.* [21], the Res2Net block exposes a new dimension, namely "scale", that is an essential and more effective factor in addition to existing dimensions of depth, width, and cardinality. This block can be "plugged" into state-of-the-art methods with no

effort, generally achieving superior results over the baseline methods. Another successful example is GhostNet. Introduced by Han *et al.* [25], the Ghost module is a "plug-and-play" component that applies a series of linear transformations with cheap computational cost in order to generate ghost feature maps that can reveal information underlying intrinsic features.

One of main challenges that CNN research is still facing is that deep CNNs are generally like a black box, in a sense that the results may lack interpretability and explainability.

The large number of hyper-parameters also still constitutes a challenge. Hyper-parameter tuning is a difficult and intuition driven task which cannot be defined by explicit formulation [36]. In this respect, Genetic Algorithms have been showing promising results. Johnson *et al.* [35] proposed a novel crossover operator that considers the nature of the underlying structures, as well as a way to explore different possible depths in an automated manner. Another challenge is the fact that deep CNNs are essentially supervised learning algorithms, thus the availability of large sets of annotated data are generally required for a successful generalization, unlike humans, that can learn and generalize from a few examples. In fact, regarding this specific challenge, LeCun *et al.* [40] states that it is expected that the future progress in computer vision will come from systems that are trained combining CNNs with RNNs that use reinforcement learning to decide where to look.

Regarding future directions, Alzubaidi *et al.* [4] and Khan *et al.* [36] agree that those are likely to include deeper research in ensemble learning (through the combination of multiple and diverse architectures, improving generalization and robustness on diverse categories), the attention mechanism — of undeniable potential due to its resemblance to the human visual system — in a way that preserves spatial relevance of objects, and improvements in both hardware technology and pipeline parallelism that can amend training times and power consumption that we are experiencing nowadays.

2.3 Related Work

Structural engineering is not one of the most active areas in nowadays' Machine Learning research. In fact, according to McKinsey, the adoption of AI solutions is quite low in engineering and construction (E&C), particularly compared with other industries, as shown in Figure 2.23 [5].

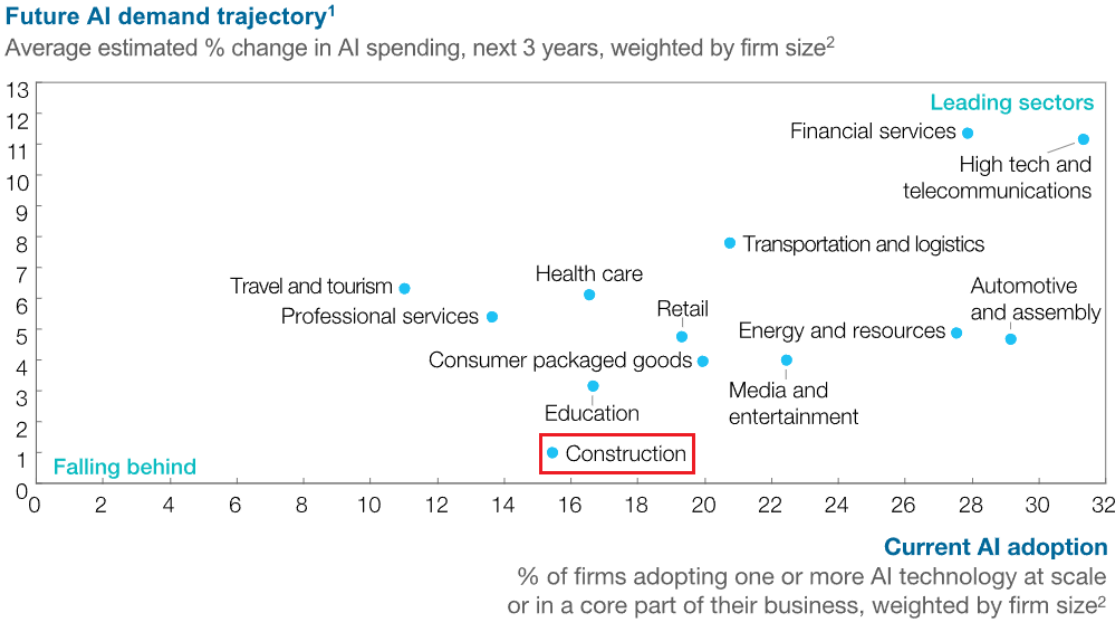


Figure 2.23: Future AI demand trajectories. Adapted from [5].

Despite not being one of the most active areas of research, Structural Engineering research is not totally insensible to the recent advances in machine learning algorithms and computational power. According to Thai [63], a bibliometric survey on Scopus[3] indexed papers collected from well recognised academic databases has identified over 485 relevant publications since 1989, most of which published in the last five years. Figure 2.24 shows the yearly evolution of publications related to machine learning (ML) applications in structural engineering, where an exponential growth in the number of publications from 2018 onward is evident, especially on those using Neural Networks (NN) methods.

[3] https://www.scopus.com/

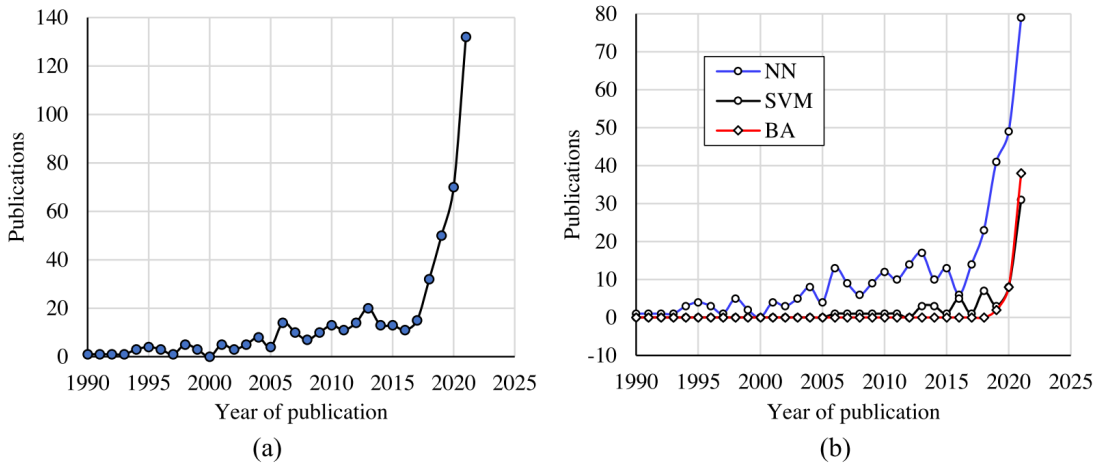


Figure 2.24: Yearly distribution of articles related to ML applications in structural engineering: (a) all algorithms and (b) three mostly used algorithms. Extracted from [63].

Neural Networks is by far the most widely used method for structural engineering. In fact, the ten most cited publications of this sample use Neural Networks. Within Neural Networks (NN) methods, Artificial Neural Networks is dominant (84%), followed by Convolutional Neural Networks (CNN), Adaptive Neuro-Fuzzy Inference Systems (ANFIS) and Radial Basis Function Neural Networks (RBFNN). The breakdown percentage of different ML methods used in structural engineering is depicted in Figure 2.25. Among these are: Boosting Algorithms (BA), Decision Trees (DT), Neural Networks (NN), Regression Analysis (RA), Random Forests (RF), Support Vector Machines (SVM) and other (mostly composed of Naive-Bayes and k-Nearest Neighbours) [63].

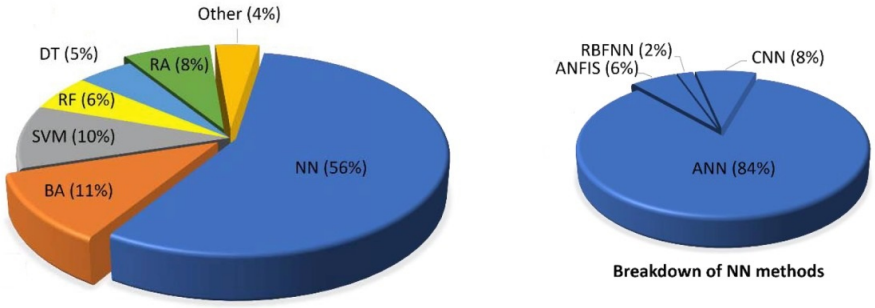


Figure 2.25: Breakdown of ML methods used in structural engineering domain. Edited from [63].

Regarding applications of this research publications, they may be split into five major groups, namely: predictions applied to structural members, prediction of fire resistance of structures, structural health monitoring (SHM) and damage detection, structural analysis and design and prediction of mechanical properties of the material (mostly related to concrete material) [63]. The pie chart in Figure 2.26 shows the distribution of the publications into these groups.

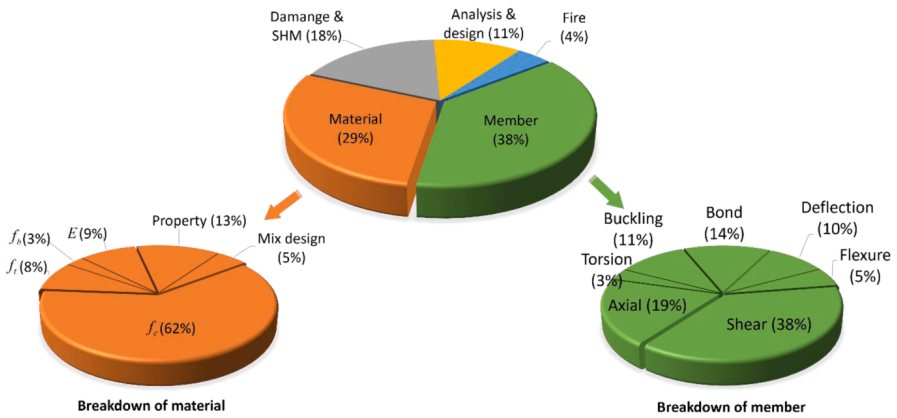


Figure 2.26: Breakdown of ML applications in structural engineering domain. Extracted from [63].

The present work fits in the "Analysis & Design" application. It corresponds not to already built structures, nor to experimental simulations' data, but to the study of "physics-based" or "mechanics-based" models and respective structural responses against a set of input variables, such as: structural properties, loading characteristics, etc. These are usually studied in order to optimize performance by reducing the number of computationally intensive simulations and topology optimization — *i.e.* structural layout optimization within a given design space.

An example of research within this category with considerable resemblance to this work was developed by Nie *et al.* [48]. In this paper, the authors present two architectures for predicting finite-element stress fields in cantilevered structures. Two different CNN architectures are presented, one of which considering all the input variables in one single channel (SCSNet), and the second one (StressNet) considering each variable in a different channel, using SE residual blocks (see "SENet" in 2.2.3). SCSNet and StressNet architectures are depicted in Figures 2.27 and 2.28, respectively.

A different approach to this problem was recently proposed by the same authors [34], using a GAN (Generative Adversarial Network) and comparing it to the previous approach. The authors state that StressGAN generally achieves higher accuracy compared to StressNet, despite StressNet being better in some particular aspects, such as estimating zero stresses in void areas. The architecture of StressGAN is shown in Figure 2.29.

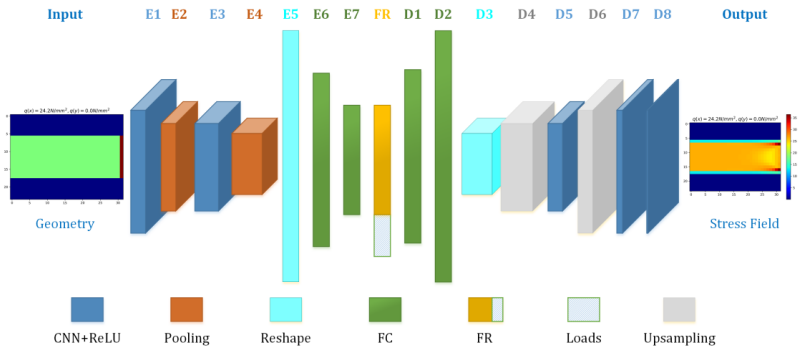


Figure 2.27: SCSNet architecture. Extracted from [48].

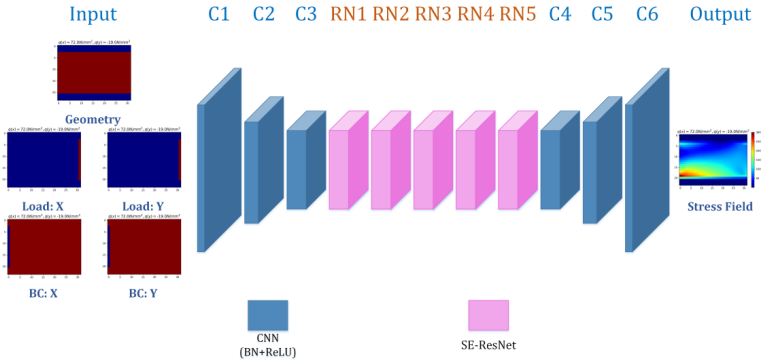


Figure 2.28: StressNet architecture. Extracted from [48].

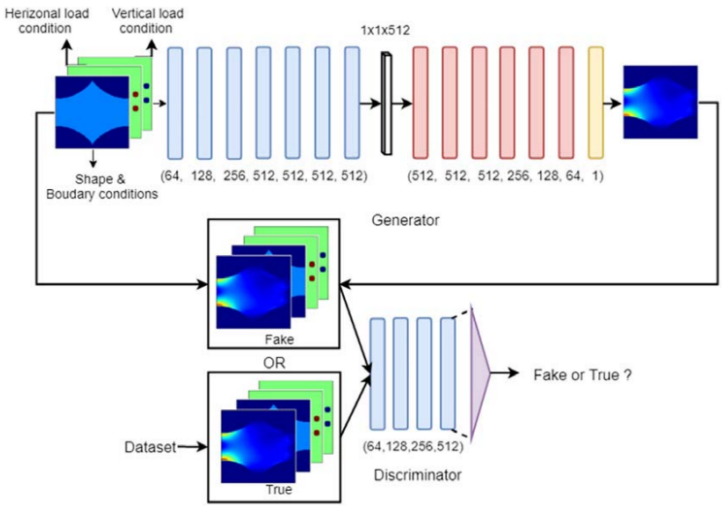


Figure 2.29: StressGAN architecture. The numbers indicate channel dimensions of the output of each block. The purple triangle means a reshape layer followed by a linear layer and a Sigmoid activation. Extracted from [34].

One of the differences of the structural problem portrayed by Nie *et al.* [48] compared to the present work is that it considers in-plane loads, while in this work, out-of-plane loads are considered. In practical terms this means that in this work it is intended to predict bending stresses, which gradient among neighbouring finite elements is expected to be much larger. Other fundamental difference is that cantilevered structures are isostatic (statically determinate) by definition, while in this work it is intended to predict stresses for hyperstatic (statically indeterminate) structures - *i.e.* structures with support redundancy, in which the resultant stress for a given finite element may not only be influenced by the closest supports, but from all the supports of the structure, depending (in simple terms) on how close they are.

Another successful example of finite-element results prediction through a deep learning approach was presented by Liang *et al.* [42], this time applied not to structural engineering *per se*, but to biomechanics. Liang *et al.* [42] developed a DL model to directly estimate stress distributions, given the geometry and material properties, of the human thoracic aorta within 1 second, while the finite-element model takes about 30 minutes on the same computer. The proposed network consists of three modules: shape encoding, non-linear mapping and stress decoding, as depicted in Figure 2.30.

This work presented by Liang *et al.* [42], despite its remarkable accuracy, does not account for the effect of different loading patterns and support conditions, due to inherent specifications of this biomedical problem. In the present work it is intended not only to learn how to generalize the effects of different load magnitudes, but also to different load patterns and support conditions.

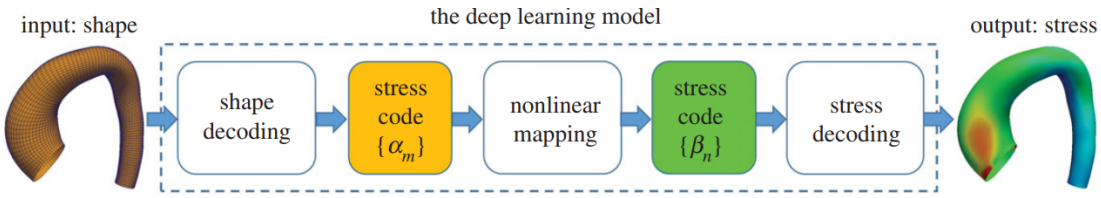


Figure 2.30: Overall data flow of the deep learning model, which takes an aorta shape as the input and outputs the wall stress distribution. Extracted from [42].

Chapter 3

Dataset

This chapter is dedicated to the description of the synthetization process of the dataset. It starts by briefly introducing the Finite Element Method (FEM) theory in Section 3.1, followed by a description of the dataset generation itself through commercial software in Section 3.2, and finally, data augmentation in Section 3.3.

3.1 Finite Element Analysis

As previously mentioned in Chapter 1 the origin of the Finite Element Method (FEM) is not consensual. As a method of piecewise polynomial approximation, to most mathematicians the FEM dates back to 1943, when Courant [14] discussed a piecewise linear approximation of a problem over a network of triangles. On its turn, the approximation of variational problems on a mesh of triangles goes back much further. In 1851, Schellbach [58] proposed a finite-element-like solution to Plateau's problem of determining the surface S of minimum area enclosed by a given curve. On the other hand, to a large segment of the engineering community, the work that represents the beginning of finite elements was presented by Turner *et al.* [64], which presents an attempt at both local approximation and the use of assembly strategies, essential to finite element methodology. Continuous improvements over the next few years and the realisation that these methods could be used to solve difficult engineering problems led to a rise in popularity of this method, which is still widely used nowadays [11].

Simply put, typical linear finite element analysis for stress calculations is expressed by [34]:

$$KQ = F \tag{3.1}$$

Where:

K – global stiffness matrix;

F – vector of the applied load at each node;

Q – nodal displacements.

The elemental stiffness k_e for each element that composes the stiffness matrix K is given as follows [34]:

$$k_e = A_e B^T D B \tag{3.2}$$

Where:
B – strain/displacement matrix;
D – stress/strain matrix;
A_e – area of the element.

B and D depend on material properties and mesh geometry. Nowadays, several direct factorization based or iterative solvers exist for computing the value of Q [34].

Linear finite element calculations (presented in the previous paragraphs) are much simpler than those non-linear. The set of possible non-linearities to be considered in structural finite element analysis is divided into two groups: geometric non-linearities and material non-linearities. A geometrically non-linear model differs from a linear model in two base assumptions, that is the fact that structural deformation has impact on the behaviour of the structure, and that there may be stability failure. On the other hand, a materially non-linear model differs from a linear model in a sense that the strain is not a linear function of the material stress. Regardless of being geometric or material, non-linearities make a finite element model iterative, computationally more demanding, thus more time consuming [69]. Because of being more demanding, non-linear models are mostly used to analyse very specific structures or self-contained phenomena in specific structural parts.

3.2 Dataset Generation

3.2.1 Dataset Design and Considerations

For the purpose of this study, two datasets were developed from the ground up. These datasets contain information regarding several configurations of 2D finite element model calculations, subject to out-of-plane loads, designed and calculated by means of the commercial software *Autodesk Robot Structural Analysis*[1]. The communication, for information extraction, with this commercial software was performed using Visual Basic 6.0 (VB6) [2] scripts specifically developed for this purpose.

The two datasets are essentially equal in terms of input data, differing in a particular calculation assumption that leads to a non-linear calculation method. This calculation assumption is further described in the next section. Once these datasets differ in the calculation method, from this point on they are referred to as "Linear dataset" and "Non-Linear dataset".

These datasets represent a 60x60 frame, in which the information of each 2D panel finite elements model is enclosed. Each instance is composed of three features (geometry, supports and

[1] https://www.autodesk.com/products/robot-structural-analysis/overview
[2] https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-basic-6/visual-basic-6.0-documentation

loads) and the ground truth (bending moments). Those four "information channels" are represented by squary 60x60 matrices, matching the maximum dimensions (in finite elements) of each panel. Both geometry and supports are binary variables. "Geometry" states if a finite element at a specific location exists of not, that is, if a certain "cell" of the panel is solid or void, whereas "supports" indicated whether a certain finite element is supported or not. The "Loads" feature has a different range, unbounded for positive values, where its value specifies the amplitude of the vertical load in kiloNewton, at which obviously 0 means not loaded.

Bending moments, on its turn, have some particularities when compared to the previously mentioned features. A proper representation of bending stresses on a bi-dimensional structure, which is the case, takes not only one, but two maps along two orthogonal directions, usually named as X and Y. The ability of predicting maps along both directions is within the scope of this work. One option to achieve such goal would be to predict both maps independently, using the same network, but having an output of two channels instead of one. Once they share the same theoretical background and some geometrical interdependence, this problem was tackled by developing additional data instances from the original ones, geometrically transformed, so that bending moments along Y direction in the original instances correspond to bending moments along X in the new instances. By sticking to the prediction of a single output channel, this operation is believed to lead to a more effective training of the network, and it is further described in Section 3.3. Unlike the previous features, the range of bending moments' data is unbounded in both positive and negative values, meaning that each map will have its own maximum and minimum value.

Figure 3.1 shows the correspondence between the actual structural model of a sample instance in the calculation software, and the construction of these features. The rightmost column shows a 3D perspective of the model, highlighting the mesh geometric contour, supports (blue triangles) and the loads (pink arrows), followed by the map of bending stresses along X direction. The leftmost column shows the correspondent features extracted from the model.

It is also worth noting that besides the three features (geometry, supports and loads), there are many other variables that can be considered in a finite element model, mainly: structural material, thickness of the panel, finite element size and lateral loads. In this case, for simplification, these were considered constant. The structural material was considered equivalent to C30/37 concrete (EN1992[3]), each panel with a thickness of 30 centimetres, discretized by quadrilateral finite elements with 50 centimetres of border dimension, subject to no lateral loading.

Regarding the composition of the dataset itself, a total of 18 000 instances were crafted. This amount is the result of a combination of 15 geometric configurations with different external contours and inner voids, 40 different load cases and 30 different support configurations assigned to each of the previously mentioned geometries. Figure 3.2 shows a combined plot of each of the base geometric configurations.

Since this dataset is fully synthesized, no additional data preparation processes were conducted prior to the feeding of the DL regression models.

[3]Eurocode 2: Design of concrete structures

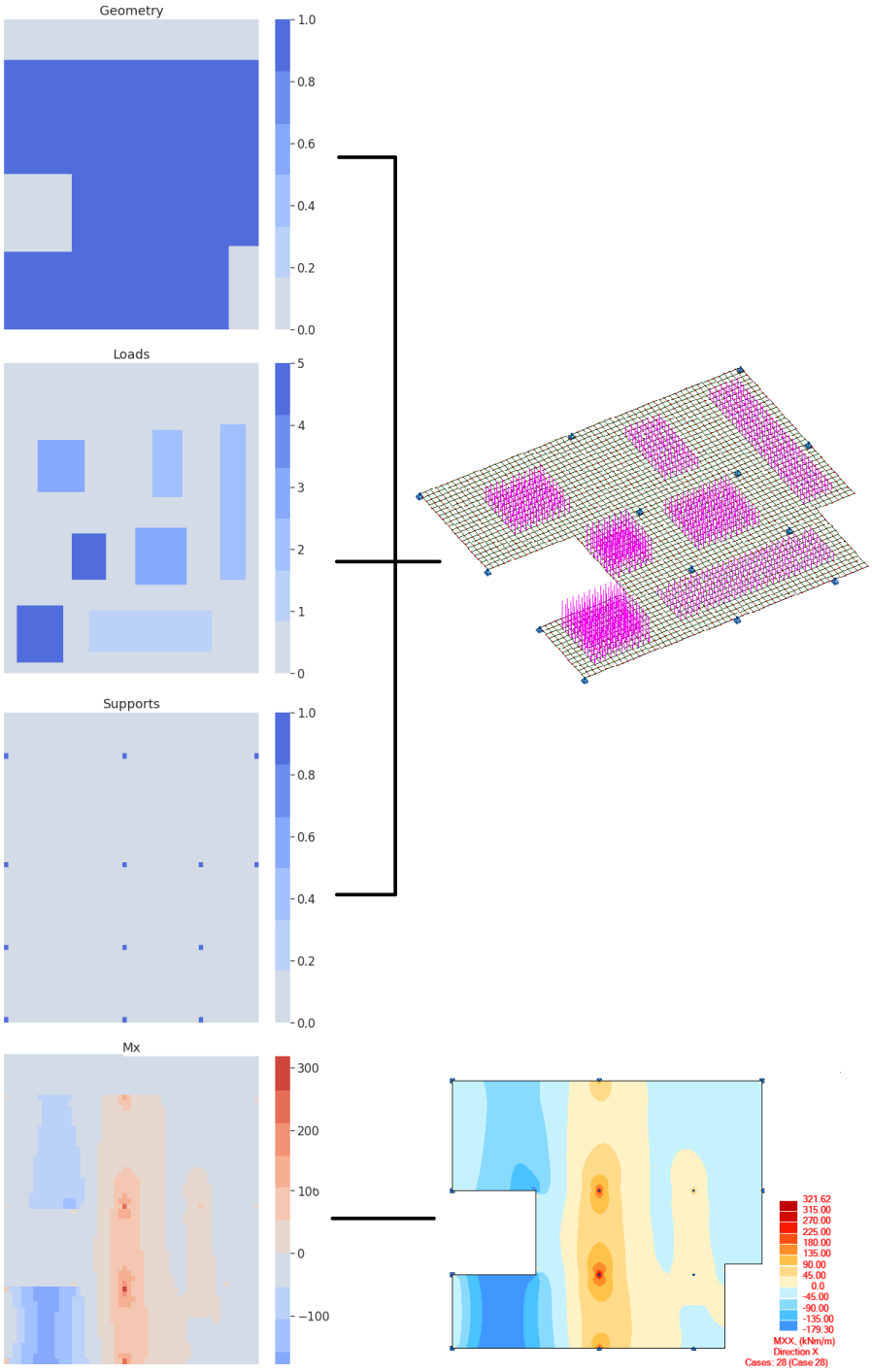


Figure 3.1: Correspondence between a sample calculation model and the feature maps

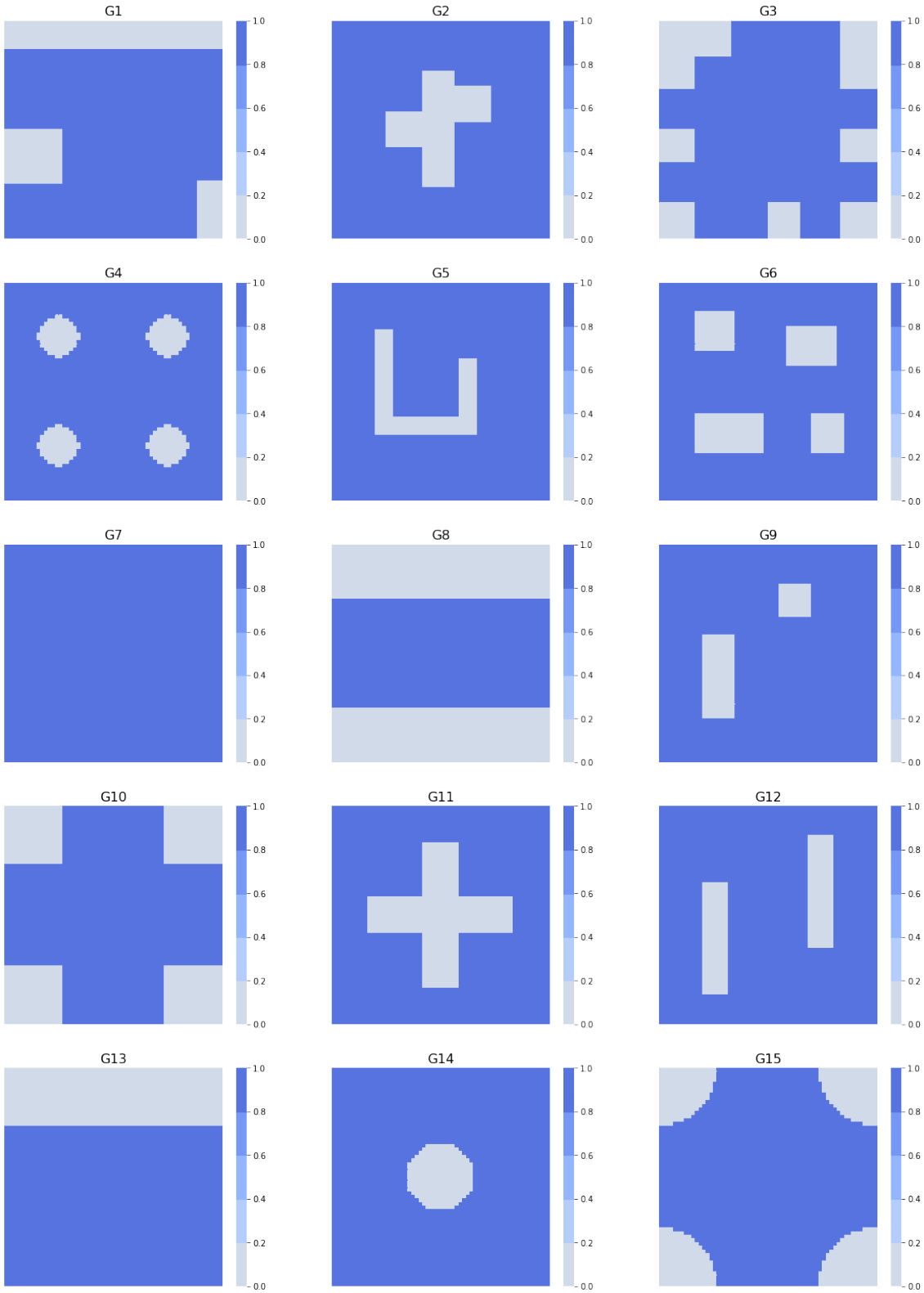


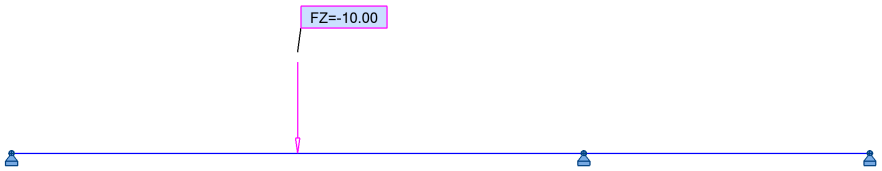
Figure 3.2: Different geometric configurations of the dataset

3.2.2 Linear *vs.* Non-Linear Calculations

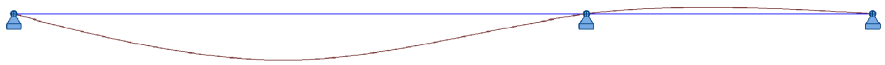
As previously mentioned in Section 3.1, many types of non-linearities can be considered when developing a structural model, even more than one type of non-linerity is adopted in some cases. Each non-linearity increases the computational demand of a model. Most commercial software allow for certain geometric non-linearities, but very few can operate with material non-linearity.

For the purpose of this work, a geometric non-linearity related to the uplifting of the supports was considered. In a general case of a static structure calculated assuming geometric linearity, a support works both ways for a given direction – in the case of the "Z" direction, its reaction can be either upwards or downwards, depending on the load configuration. On the other hand, considering the mentioned non-linearity, a support can either stay in place and have a reaction force upwards, if the applied load demands so, or will uplift if otherwise. In the presence of an hyperstatic structure, these different approaches (linear or non-linear) lead to different stresses and deformations.

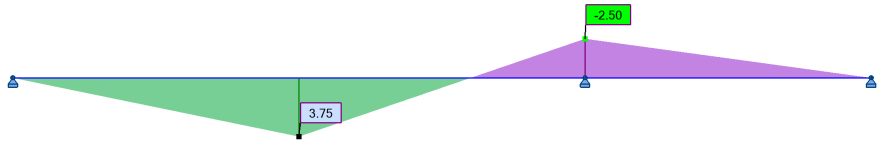
For a clearer understanding of this phenomena, Figures 3.3 and 3.4 show two simple examples of a single bar structure, calculated with linear and non-linear approaches, respectively. By comparing these figures, one can easily spot the differences in the results of the models. In the non-linear case, despite maintaining static equilibrium, the rightmost support uplifts, causing a different behaviour of the structure, and consequently different internal forces.



(a) Model geometry and load

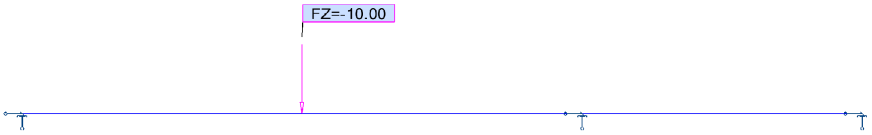


(b) Qualitative deformation

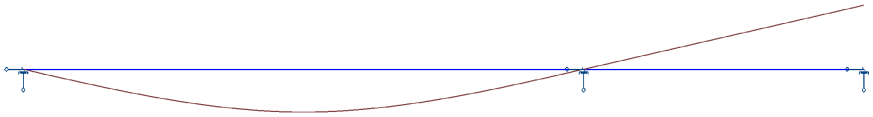


(c) Bending moment [kN.m]

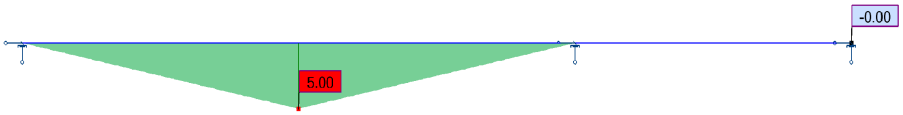
Figure 3.3: Sample linear model and calculation results



(a) Model geometry and load



(b) Qualitative deformation



(c) Bending moment [kN.m]

Figure 3.4: Sample non-linear model and calculation results

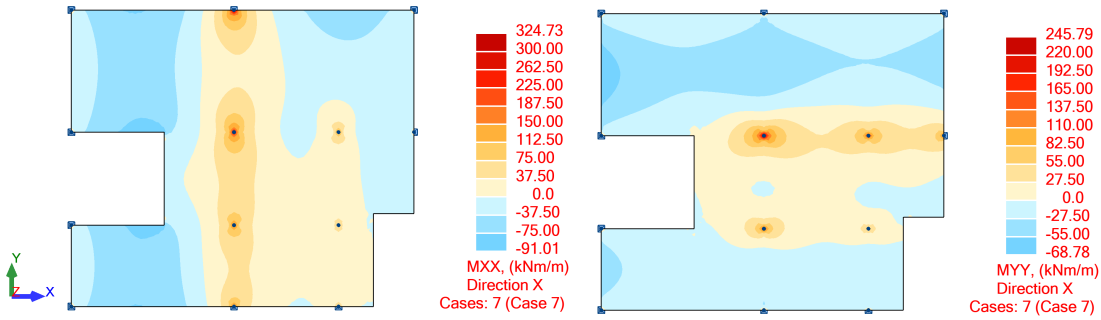
3.3 Data Augmentation

As previously mentioned in section 3.1, the original dataset was augmented as an attempt to train models more effectively.

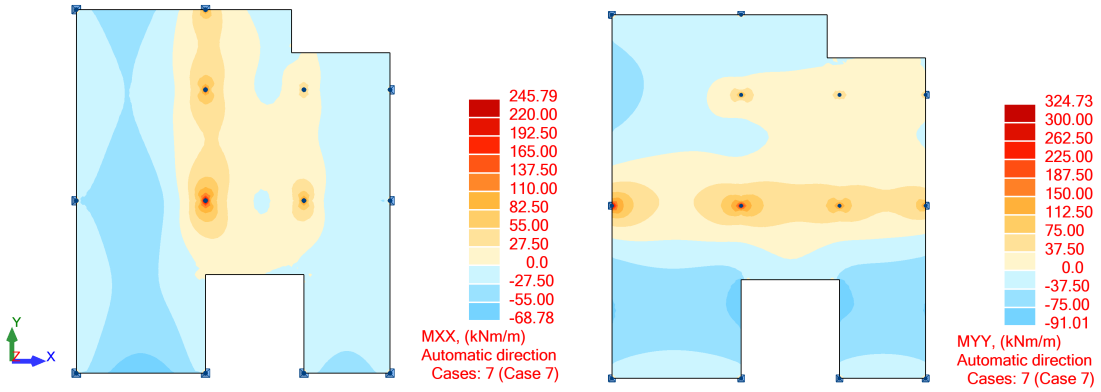
Despite the resemblance of this problem to the generation of general images, in this case data augmentation cannot be performed by means of several traditional computer vision transformations, such as: application of filters, application of patches, isolating parts of the image, combining parts of different images or rotating in angles other than orthogonal. Since the dataset comprehends features of a calculation model and the results of the calculation of those, this data only makes sense as a whole, once the application of any of the before-mentioned transformations would corrupt the mathematical link between features and results.

As also described previously, in section 3.2.1, the representation of bending stresses involves two result maps along two orthogonal directions, generally taken as X and Y. Instead of designing the regression models to predict for these two channels, this problem was tackled by sticking to the prediction of a single output channel (along X direction) and stacking additional transformed data, based on the following principle of interchangeability: the bending moment map along X direction for certain model, corresponds to the bending moment map along Y direction for the same model rotated by 90 degrees. The same principle applies to the bending moments along Y of the original model, which rotated by 90 degrees corresponds to the bending moments along X in the newly generated model. This fact makes the regression model versatile for predicting the

bending moments along X and Y, as long as the input features are oriented in the correct direction. This correspondence is evidenced by Figure 3.5, where the bending moment map along X of the original model (top-left) corresponds to the bending moment map along Y of the rotated model (bottom-right), and likewise, the moment map along Y of the original model corresponds to the map along X of the rotated model.



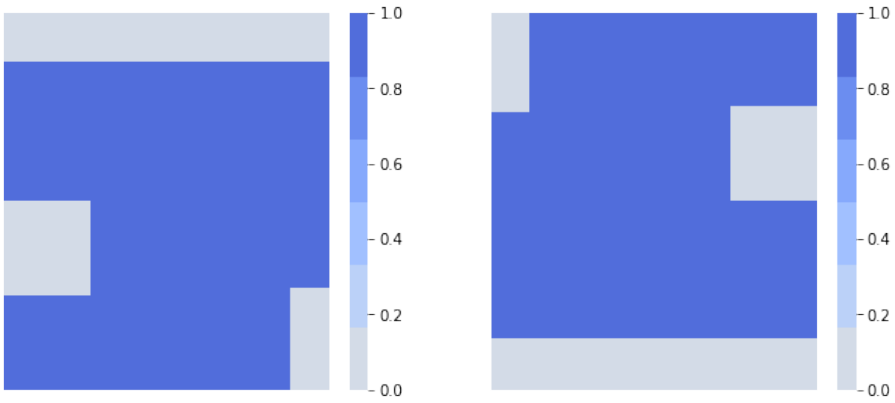
(a) Original model - bending moments along X (LHS) and Y (RHS)



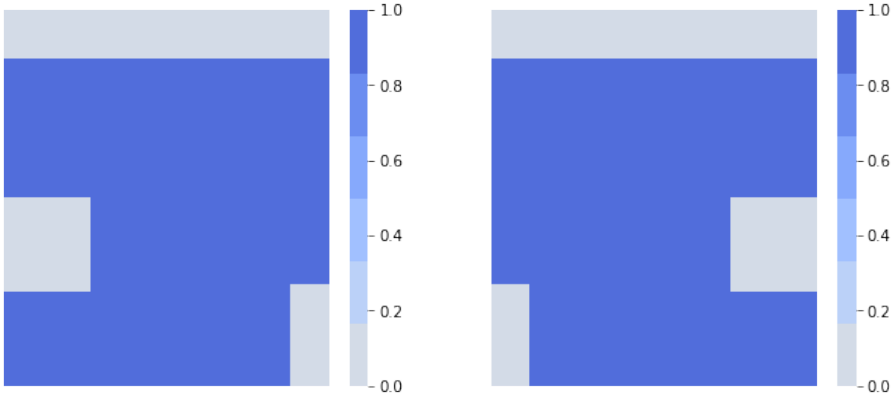
(b) 90-degrees rotated model - bending moments along X (LHS) and Y (RHS)

Figure 3.5: Correspondence of X and Y bending moments between a model and itself rotated by 90 degrees

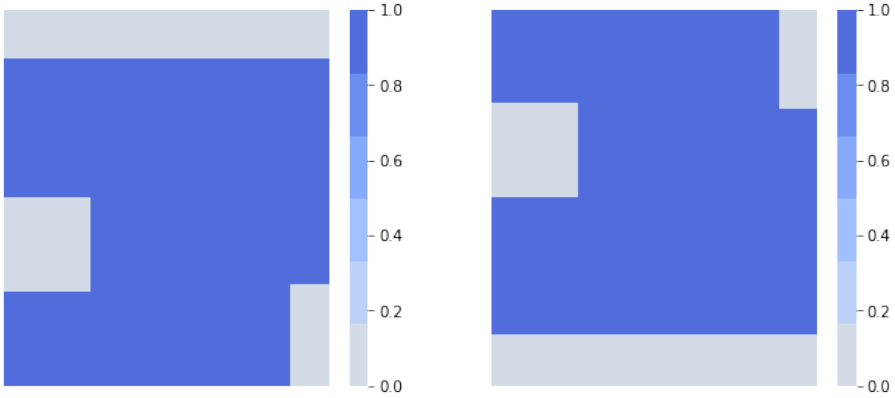
In summary, the application of the previous operation results in a new dataset, with the same size as the initial. The concatenation of these two subsets results in a dataset of 36 000 instances, object of further augmentation. This new dataset was then rotated by 180 degrees and flipped over both vertical and horizontal axes, resulting in a total of **136 886 instances**, excluding repeated cases. Figure 3.6 depicts an example of application of these three operations in the geometry feature of a sample instance, by showing the original instance on the left-hand side, and the transformed instance on the right-hand side.



(a) 180 degree rotation



(b) Flipping along vertical axis



(c) Flipping along horizontal axis

Figure 3.6: Additional data augmentation operations

Chapter 4

Experimental Methodology

The present chapter intends to describe the experimental methodology applied in this work. This methodology comprehends the use of two deep learning CNN models. One of these, Model A, derives from a model proposed by Nie *et al.* [48] for predicting linear FEM stresses of isostatic structures. The second model, Model B, is an improvement attempt over Model A, more complex, hence with more parameters, and some fundamental changes. This exposition starts with a brief explanation of the data split for training, testing and validation, in section 4.1, followed by the presentation of the deep learning models themselves, with particular focus on their architecture and hyperparameters in section 4.2, and finally, the metrics adopted for the evaluation of the models, in section 4.3.

4.1 Train-Test-Validation Data Split

As previously stated, both datasets (calculated using linear and non-linear approaches) are the same size, totalling 136 886 instances after augmentation. In the case of the Linear dataset, all these instances are qualified to participate in the deep learning process through the regression model, but the same does not happen with the case of the Non-Linear dataset. The introduction of the non-linearity that makes the calculation models more complex – by adding the possibility of uplifting supports – made some instances statically inadmissible and therefore non-solvable. This fact leads to a slightly shorter Non-Linear dataset when compared to the Linear case. Since this work also has the objective of comparing the performance of the same model trained with Linear and Non-Linear datasets, once the latter is slightly shorter, it was taken as a baseline for splitting for training, testing and validation at fixed percentages, being the remainder of the Linear dataset added to the testing subset. Tables 4.1 and 4.2 summarise the breakdown setup of the Linear and Non-Linear datasets, respectively.

Table 4.1: Data split setup – Linear Dataset

	Instances	**Percentage**
Train	85 620	62.55%
Test	30 734	22.45%
Validation	20 532	15.00%
Total	**136 886**	**100.00%**

Table 4.2: Data split setup – Non-Linear Dataset

	Instances	**Percentage**
Train	85 620	70.00%
Test	18 348	15.00%
Validation	18 347	15.00%
Total	**122 315**	**100.00%**

4.2 Deep Learning Models

As previously mentioned, two deep learning regression models (Model A and Model B) were developed in this work. These models were built up using the PyTorch framework, and both were trained using a Nvidia Tesla P100 16GB GPU provided by Google Colab[1] platform. Further considerations regarding these models are outlined below.

4.2.1 Model A

Model A is an encode-decoder CNN model, adapted from the *StressNet* approach proposed by Nie *et al.* [48], previously presented in Section 2.3, for the prediction of linear FEM stresses of isostatic structures subject to in-plane loads, where it has shown to produce good results.

The model proposed by Nie *et al.* [48] cannot simply be used "as is" in the present work due to the fact that it needs to be able to output both negative and positive values, not just positive. The output layer of *StressNet* is a convolutional layer that takes 32 channels as input and outputs one channel (result) with a ReLU activation function. As stated in Section 2.2.2, the ReLU activation function outputs a value of 0 for every input lower than or equal to 0. This peculiar characteristic prevents the output from having negative values, which is incompatible with the dataset in analysis. To overcome this limitation while keeping the same number of activations, the previously mentioned layer was kept with the ReLU activation function but with an output size of 32 channels (same as input). After that, as the output layer, an additional convolutional layer was adopted, with a kernel size of 1x1 and no activation function, with the sole objective of performing dimensionality reduction from 32 channels to one single channel, *i.e.* the result.

[1] https://colab.research.google.com/

Figure 4.1 outlines the whole architecture of the model, thoroughly described in Table 4.3. The encoder stage is composed of 3 convolutional layers, with batch normalization (BN) and ReLU activation function, progressively decreasing the size of the input (downsampling) and increasing the number of channels (Layers 1 to 3). The decoder stage does the exact opposite, by increasing the input dimension up to its initial size while decreasing the number of channels (Layers 9 to 11), followed by the additional dimensionality reduction convolution introduced in the last paragraph (Layer 12). Amid these stages sits a stack of 5 Residual Squeeze & Excitation blocks. A residual block, in its essence, is used to mimic identical layers in order to fight the vanishing gradient problem [48]. This block is composed of two consecutive convolutional layers with a kernel size of 3x3, batch normalization, and ReLU activation function, followed by a Squeeze & Excitation block (hence the name) and a shortcut connection. Figure 4.2 depicts the architecture of this block.

The Squeeze & Excitation block, as already presented in Section 2.2.3, is meant to improve the representational capacity of the network by adding a mechanism to weight each channel depending on how relevant each one is. Following the scheme shown in Figure 4.3, the input data $u \in R^{HxWxC}$ is squeezed into $S(u)$ through the global average-pooling layer. The output of this operation is then subject to two consecutive fully connected layers, downsampling (FC+ ReLU) and upsampling (FC+ Sigmoid) back to the dimension of $S(u)$. $E(u)$ is obtained by reshaping the output into a tensor in order to finally multiply with the input u, obtaining v with the same dimensions as the input u.

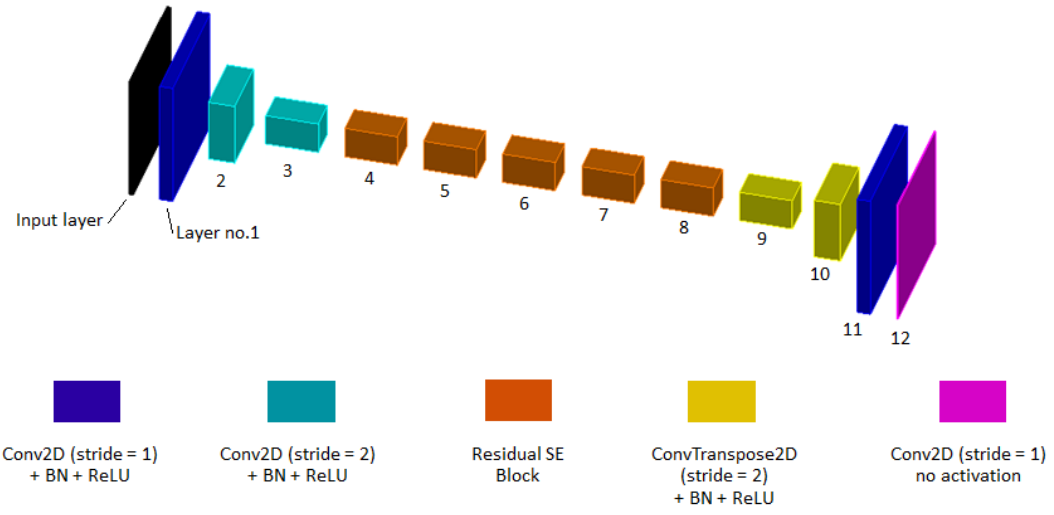


Figure 4.1: Architecture of Model A

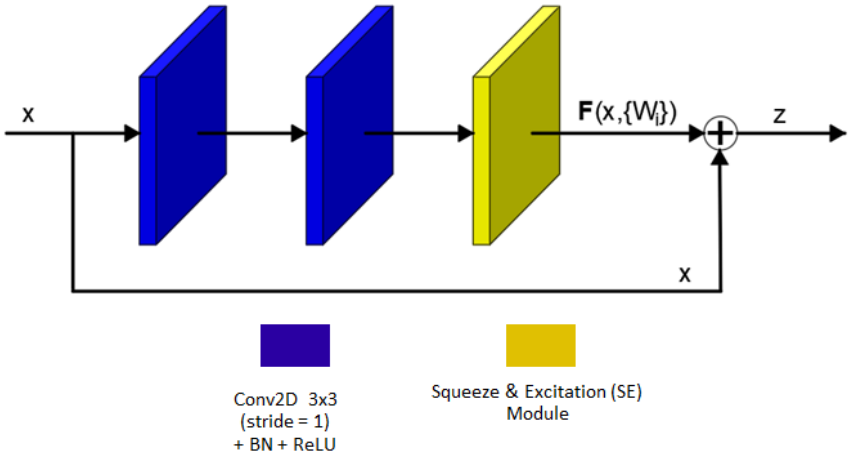


Figure 4.2: Architecture of a Residual SE Block. Adapted from [48].

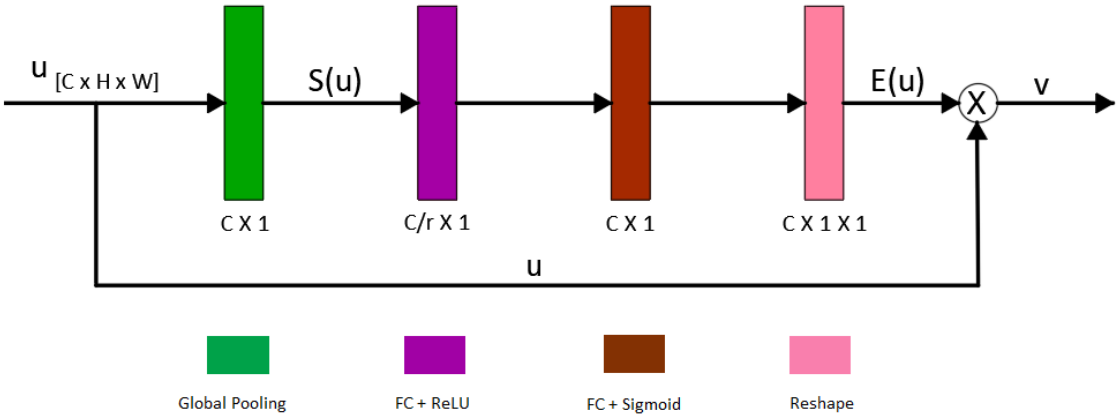


Figure 4.3: Architecture of a Squeeze and Excitation (SE) block Adapted from [48].

Table 4.3: Model A – Layer Setup

Layer no.	**Type**	**Activation function**	**Stride**	**Pad size**	**Filter size**	**Output Shape [Chan., L, C]**	**Parameters**
1	Conv. + BN	ReLu	1	4	9x9	[32, 60, 60]	7 872
2	Conv. + BN	ReLu	2	1	3x3	[64, 30, 30]	18 624
3	Conv. + BN	ReLu	2	1	3x3	[128, 15, 15]	74 112
4	Residual SE Block	n.a.	n.a.	n.a.	n.a.	[128, 15, 15]	299 920
5	Residual SE Block	n.a.	n.a.	n.a.	n.a.	[128, 15, 15]	299 920
6	Residual SE Block	n.a.	n.a.	n.a.	n.a.	[128, 15, 15]	299 920
7	Residual SE Block	n.a.	n.a.	n.a.	n.a.	[128, 15, 15]	299 920
8	Residual SE Block	n.a.	n.a.	n.a.	n.a.	[128, 15, 15]	299 920
9	Transposed Conv. + BN	ReLu	2	1	3x3	[64, 30, 30]	73 920
10	Transposed Conv. + BN	ReLu	2	1	3x3	[32, 60, 60]	18 528
11	Conv. + BN	ReLu	1	4	9x9	[32, 60, 60]	83 040
12	Conv.	None	1	0	1x1	[1, 60, 60]	33
Total number of parameters:							1 775 729

Model A was trained along 3500 epochs, with a batch size of 256 instances, using Adam optimizer, a learning rate of 0.02 with exponential decay, and Mean Squared Error (MSE) as its function.

The adopted stopping criteria for the training process was the lack of significant evolution not only in the loss of the validation set, but also along other parallel performance metrics, that were kept track of, namely: Mean Absolute Error (MAE) and median Weighted Mean Absolute Percentage Error (WMAPE) – detailed in section 4.3. These metrics were monitored at the end of every epoch.

The GPU model used for the training process can handle a batch size up to 1048 instances. The use of such a large batch size results in faster processing times per epoch, but the whole process takes far more epochs to converge than with a smaller batch size. Having this in mind, a batch size of 256 was found to be a good compromise between the processing time taken for each epoch and the number of epochs to achieve the so-called convergence. With this batch size, each epoch takes nearly 68 seconds to compute, resulting in a total training time of nearly 66 hours for each dataset (Linear and Non-Linear).

Mean Squared Error is one of the typical loss functions used for regression problems [52]. After some trials with other absolute and relative loss functions, such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Huber Loss, and median Weighted Mean Absolute Percentage Error (WMAPE), MSE has proven to be the most suitable for this problem by being able to achieve lower validation error among the considered metrics.

4.2.2 Model B

Model B is a tentative upgrade of Model A. Following the same encoder-decoder type of architecture, one of the differences between this model and Model A is the robustness of both encoder and decoder stages. Despite keeping only two downsampling steps in the encoder stage, two additional convolutional layers activated by non-linear functions were placed before each downsampling layer (convolutional layer with stride 2). The same happens in the decoder stage. Apart from that, all the ReLU activation functions in the main layers were replaced by Exponential Linear Units (ELU), all the Squeeze & Excitation (SE) Residual blocks between the encoding and decoding stages were replaced by Residual Concurrent Spatial and Channel Squeeze & Excitation Blocks (SCSE) blocks, and an additional operation was added in the output layer, which is the element wise multiplication of the output of the decoding stage with the "Geometry" channel of the input features, generating the new output layer.

The objective of adding two additional convolutional layers before each downsampling layer is to provide the model with more non-linearity, improving its potential for generalization. By replacing the SE Residual Blocks with SCSE Residual Blocks the idea is to extend the recalibration not only along channels, but also along space providing spatial attention to focus on certain regions of the channels [55]. This block has half of its architecture in common with the previously presented SE Block, apart from the Spatial Squeeze & Excitation (SSE), which occurs parallelly to Channel Squeeze & Excitation (CSE, previously designated simply as SE). Both CSE and SSE outputs are summed at the end, producing the final output. The architecture of this block is depicted in Figure 4.5. The replacement of the ReLU activation functions of the convolutional layers leads to a faster convergence of the model (in terms of number of epochs) with the plus of ditching the "dying ReLU" problem (Section 2.1.1), even though it is slower to compute than ReLU. The additional element-wise operation placed at the output stage of the network intends to take advantage of some prior logic knowledge about FEM models. This knowledge is simply the fact that it is known beforehand that the output results (*i.e.* bending moments) must be null in void regions of the model. In other words, the result must be null at every node where the "Geometry" feature has the value of 0. Once this binary feature only assumes the values 0 and 1, a simple element-wise multiplication with the output will zero eventual residual values in the void regions and keep the result "as is" in every node representing actual material. The whole architecture of Model B as well as the detailed layer setup are shown in Figure 4.4 and Table 4.4, respectively.

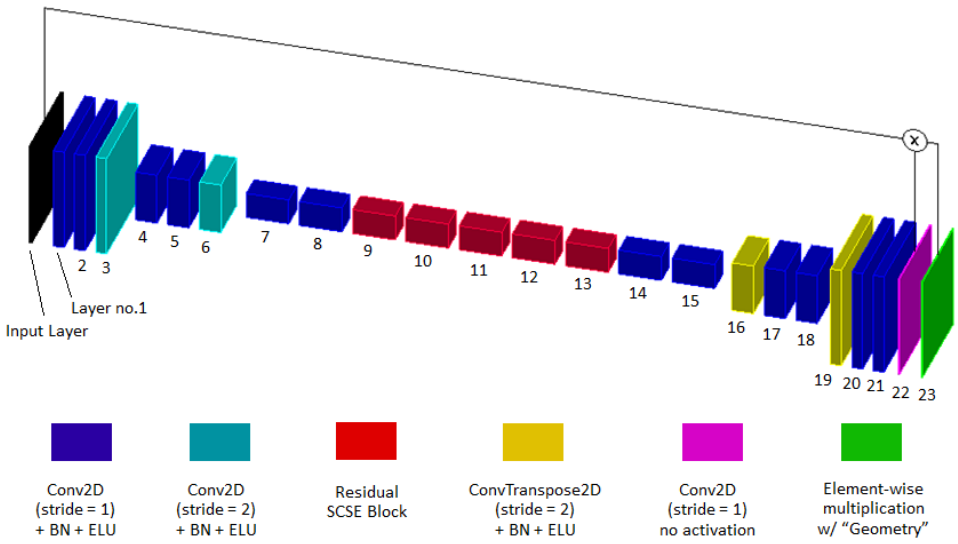


Figure 4.4: Architecture of Model B

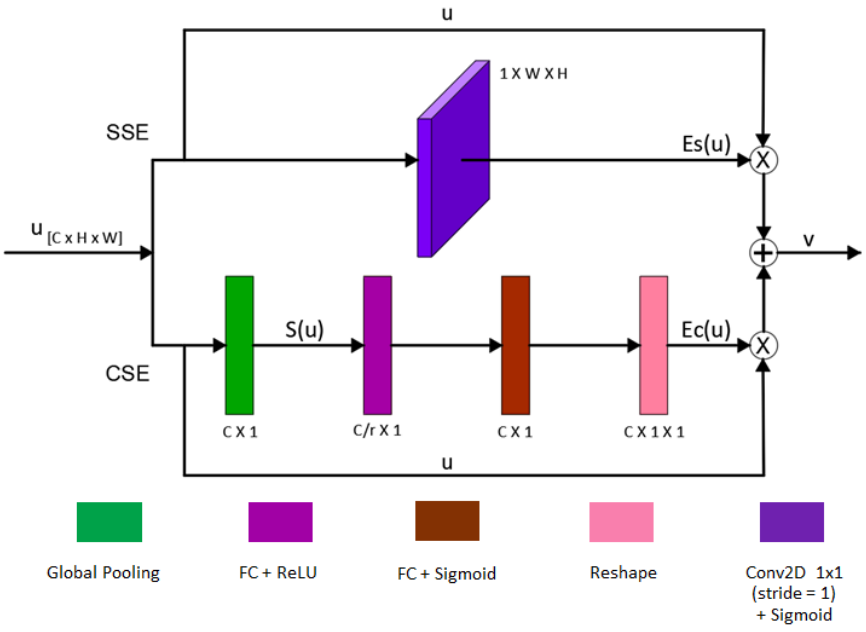


Figure 4.5: Concurrent Spatial and Channel Squeeze & Excitation (SCSE) Block

Table 4.4: Model B – Layer Setup

Layer no.	**Type**	**Activation function**	**Stride**	**Pad size**	**Filter size**	**Output Shape [Chan., L, C]**	**Parameters**
1	Conv. + BN	ELU	1	4	9x9	[32, 60, 60]	7 872
2	Conv. + BN	ELU	1	4	9x9	[32, 60, 60]	83 040
3	Conv. + BN	ELU	2	4	9x9	[64, 30, 30]	83 040
4	Conv. + BN	ELU	1	2	5x5	[64, 30, 30]	51 392
5	Conv. + BN	ELU	1	2	5x5	[64, 30, 30]	102 592
6	Conv. + BN	ELU	2	2	5x5	[128, 15, 15]	102 592
7	Conv. + BN	ELU	1	1	3x3	[128, 15, 15]	74 112
8	Conv. + BN	ELU	1	1	3x3	[128, 15, 15]	147 840
9	Residual SCSE Block	n.a.	n.a.	n.a.	n.a.	[128, 15, 15]	300 049
10	Residual SCSE Block	n.a.	n.a.	n.a.	n.a.	[128, 15, 15]	300 049
11	Residual SCSE Block	n.a.	n.a.	n.a.	n.a.	[128, 15, 15]	300 049
12	Residual SCSE Block	n.a.	n.a.	n.a.	n.a.	[128, 15, 15]	300 049
13	Residual SCSE Block	n.a.	n.a.	n.a.	n.a.	[128, 15, 15]	300 049
14	Conv. + BN	ELU	1	1	3x3	[128, 15, 15]	147 840
15	Conv. + BN	ELU	1	1	3x3	[128, 15, 15]	147 840
16	Transposed Conv. + BN	ELU	2	1	3x3	[64, 30, 30]	73 920
17	Conv. + BN	ELU	1	1	3x3	[64, 30, 30]	37 056
18	Conv. + BN	ELU	1	1	3x3	[64, 30, 30]	37 056
19	Transposed Conv. + BN	ELU	2	1	3x3	[32, 60, 60]	18 528
20	Conv. + BN	ELU	1	1	3x3	[32, 60, 60]	9 312
21	Conv. + BN	ELU	1	1	3x3	[32, 60, 60]	9 312
22	Conv.	None	1	0	1x1	[1, 60, 60]	33
23	Element-wise multiplication	n.a.	n.a.	n.a.	n.a.	[1, 60, 60]	0
Total number of parameters:							2 633 62

Model B was trained along 2500 epochs. Similarly, and for the same reasons as Model A, this model was trained with a batch size of 256 instances, using Adam optimizer, a learning rate of 0.02 with exponential decay and Mean Squared Error (MSE) loss function. Each epoch takes 114

seconds to compute, leading to a total training time of about 79 hours for each dataset.

4.3 Evaluation Metrics

The results of the regression models were evaluated by means of five absolute error metrics (Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE) and Peak Absolute Error (PAE)) and two relative error metrics, *i. e.* Weighted Mean Absolute Percentage Error (WMAPE) and Percentage Peak Absolute Error (PPAE). Their definitions are presented herein.

4.3.1 Mean Absolute Error (MAE)

Mean Absolute Error (MAE) measures the average magnitude of the errors in a regression. Regardless of the direction of the errors, it always ranges from 0 to ∞, where 0 means a perfect approximation. MAE can be estimated using (4.1):

$$\text{MAE} = \frac{1}{N}\sum_{i}^{N}|y_i - \hat{y}_i| \tag{4.1}$$

Where:
N – number of samples;
$\hat{y}$ – prediction value;
y – target value.

4.3.2 Mean Squared Error (MSE)

Mean Squared Error (MSE) represents the average magnitude of the squared errors. Compared to MAE, MSE has the characteristic of heavily weighting outlier values. As a result of using squared values, MSE units are not the same as the actual observed values, hence not directly comparable. Its calculation is given by the following expression (4.2):

$$\text{MSE} = \frac{1}{N}\sum_{i}^{N}(y_i - \hat{y}_i)^2 \tag{4.2}$$

Where:
N – number of samples;
$\hat{y}$ – prediction value;
y – target value.

4.3.3 Root Mean Squared Error (RMSE)

Simply put, Root Mean Squared Error (RMSE) is the root value of MSE. This metric has the advantage of combining both larger penalization of outliers and having the same unit as the observed values, making it directly comparable. Its expression is given as follows (4.3):

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} = \sqrt{MSE} \tag{4.3}$$

Where:

N – number of samples;

$\hat{y}$ – prediction value;

y – target value.

4.3.4 Peak Absolute Error (PAE)

Peak Absolute Error (PAE) estimates the local absolute error at the finite-element where the highest stress is observed. This value is of critical importance in engineering applications. Its location is firstly determined on the ground truth side, then compared to the prediction value at the same position, in order to account for its magnitude but also for its position in the array. Its calculation is given by the following expression (4.4):

$$\text{PAE} = |\max y_i - \hat{y}_i[\arg\max y_i]| \tag{4.4}$$

Where:

$\hat{y}$ – prediction value;

y – target value.

This metric was used discriminately for positive and negative values of the results. Therefore, this metric will be referred to as PAE+ or PAE- depending on whether positive or negative values are at stake.

4.3.5 Weighted Mean Absolute Percentage Error (WMAPE)

Weighted Mean Absolute Percentage Error (WMAPE) is a relative metric that gives more importance to errors of observations with a high realized value. Furthermore, WMAPE ranges from 0 to ∞ and equally penalizes over-estimating and under-estimating [50]. Its calculation is given by dividing the sum of errors by the sum of target values (4.5).

The name assigned to this metric is not consensual among the literature. Some references refer to this metric as Weighted Absolute Percentage Error (WAPE).

$$\text{WMAPE} = \frac{\sum_{i=1}^{N}|y_i - \hat{y}_i|}{\sum_{i=1}^{N}|y_i|} \tag{4.5}$$

Where:

N – number of samples;

$\hat{y}$ – prediction value;

y – target value.

4.3.6 Percentage Peak Absolute Error (PPAE)

Percentage Peak Absolute Error (PPAE) is conceptually similar to Peak Absolute Error (PAE) in the sense that it measures the local error at the highest stress point of each instance. The difference, compared to PAE, is that instead of this estimating the absolute error, it is expressed in relative terms. Its calculation is given by the following expression (4.6):

$$\text{PPAE} = \frac{|\max y_i - \hat{y}_i[\arg\max y_i]|}{|\max y_i|} = \frac{\text{PAE}}{|\max y_i|} \tag{4.6}$$

Where:
$\hat{y}$ – prediction value;
y – target value.

Similarly to PAE, this metric was unfolded into PPAE+ and PPAE- depending on whether positive or negative values, respectively, are object of evaluation.

4.4 Success Criteria

From the metrics presented in the previous section, in order to assess the performance of the models, a success criteria was established based on structural engineering practice. It is obvious that, like with every other kind of practise, the lower error, the better, but in this case it was settled that for conceptual/early stages of structural design, each predicted instance is considered a successful approximation if at least one of the two criteria are met:

- $PPAE- \leq 1\%$, $PPAE+ \leq 1\%$ and $WMAPE \leq 1\%$;
- $PAE+ \leq 5.0$, $PAE- \leq 5.0$ and $MAE \leq 1.0$.

For being of harder interpretation and inherently correlated with each other, MSE and RMSE were disregarded in this criteria. These metrics differ from MAE in the sense that they are more penalized by outliers. Since WMAPE is itself weighted by the magnitude of each value and that both PAE+ and PAE- assess the errors at potential outliers (maximum and minimum value), the variability of errors among each instance is assumed to be covered.

Chapter 5

Results and Discussion

The present chapter is dedicated to the presentation and discussion of the results obtained using the models presented in Chapter 4. Firstly, a brief overview of the training process of both models is outlined in section 5.1, followed by the results and error metrics obtained by the application of the trained models against the test datasets (section 5.2). Finally, section 5.3 intends to present some interpretation and comments to these results.

5.1 Models Training

This section is organized in two sub-sections corresponding to the two models object of study (Model A and Model B). Each section comprises the training process of the corresponding model under both datasets.

5.1.1 Model A

Figures 5.1, 5.2 and 5.3 plot the evolution, at logarithmic scale, of MSE, MAE and median WMAPE, respectively, throughout the training process of Model A for both datasets. It is noteworthy to mention that, despite the equal sizes of both datasets, training the model on the Non-Linear dataset leads to higher errors along all metrics, compared to training on the Linear dataset. This fact complies with the increased mathematical complexity of the Non-Linear calculation models when compared to the linear calculation models. It is also worth noting that from epoch 250 onwards, the MSE of the validation set starts to drift apart from the training set. Despite the increase of this gap along further epochs, the error on the validation set continues to improve along all metrics, however constantly slowing down the rate. Since an actual increase of the error on the validation set is never observed, it cannot be said that the model experiences overfitting.

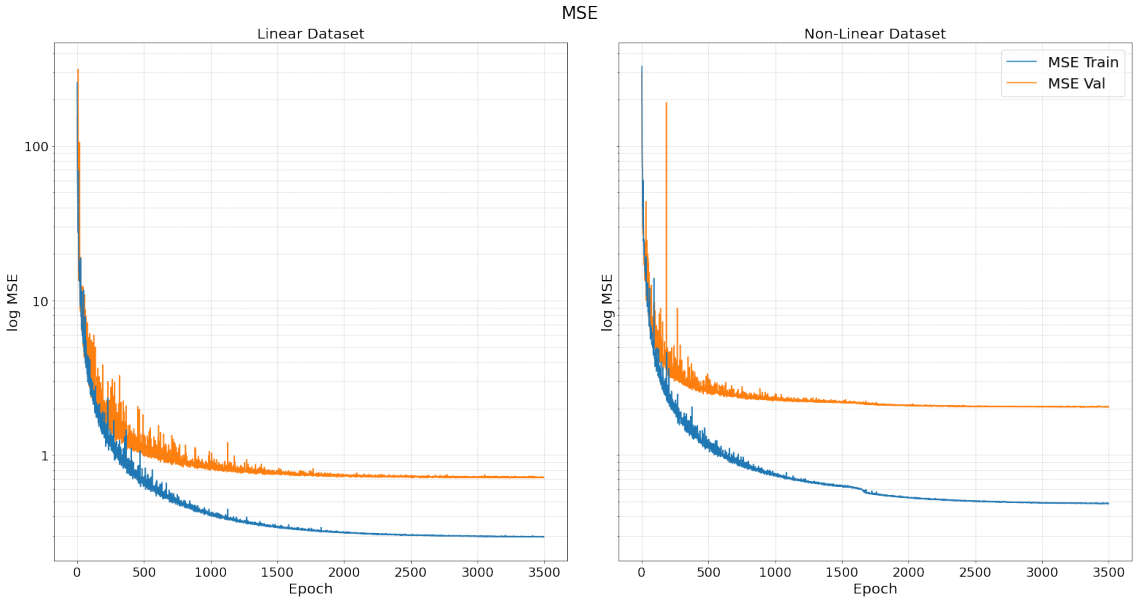


Figure 5.1: Loss function (MSE) evolution along the training process of Model A

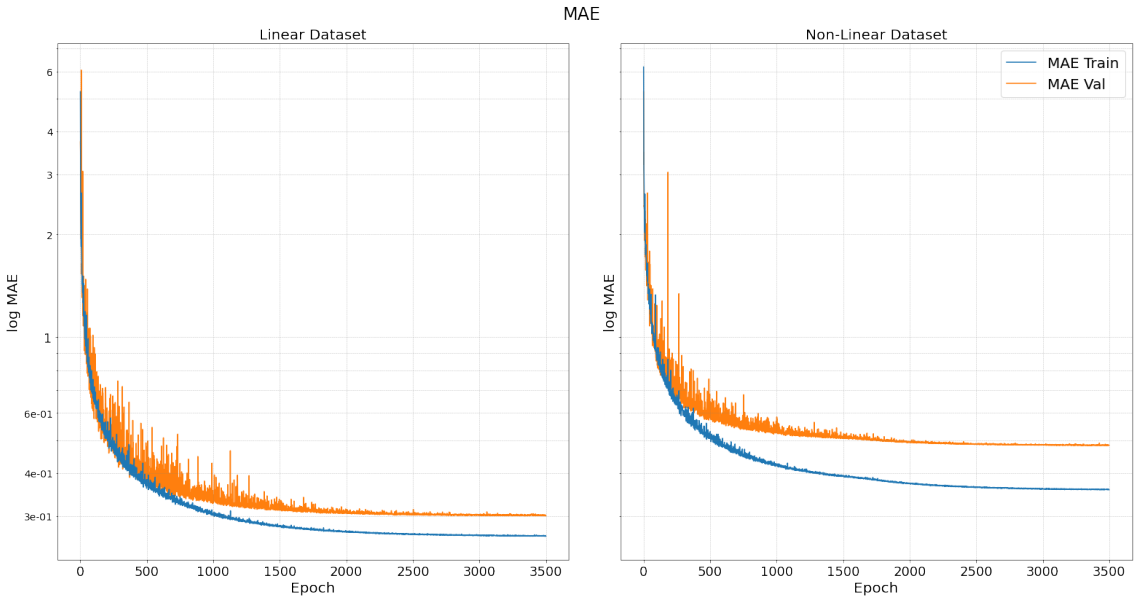


Figure 5.2: MAE evolution along the training process of Model A

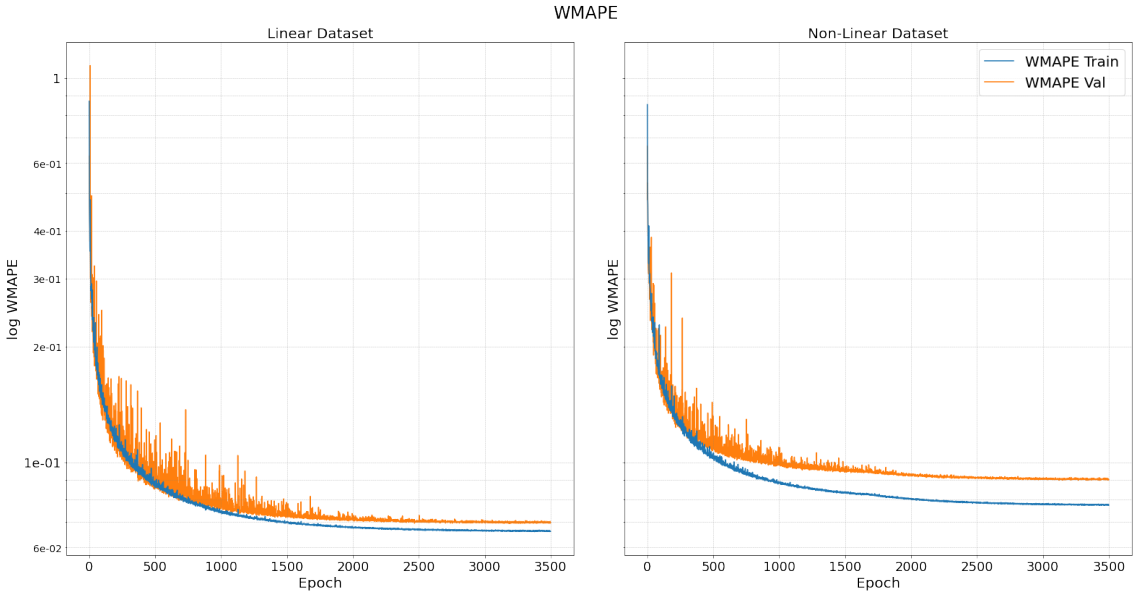


Figure 5.3: Median WMAPE evolution along the training process of Model A

5.1.2 Model B

Figures 5.4, 5.5 and 5.6 show the evolution of MSE, MAE and median WMAPE, respectively, along the training process of Model B. Compared to Model A, lower errors along either metrics are observable, as well as a narrower gap between the train and validation results for all metrics, which is desirable. Similarly to the training process of Model A, when training on the Non-Linear dataset, the validation errors are higher among all metrics.

Figure 5.7 shows the prediction results of a sample instance of the test set using the weights of intermediate checkpoint models (Epoch no. 10 and Epoch no. 100) and the final weights of the model (Epoch no. 2500) *versus* the ground truth, *i.e.* the bending moments obtained by the mathematical finite-element model. The differences are visually noticeable, especially between the first two checkpoints (Epoch no.10 and Epoch no.100), where considerable changes have taken place within a small range of epochs. From Epoch no.100 to Epoch no.2500, differences are still noticeable, although at a finer level.

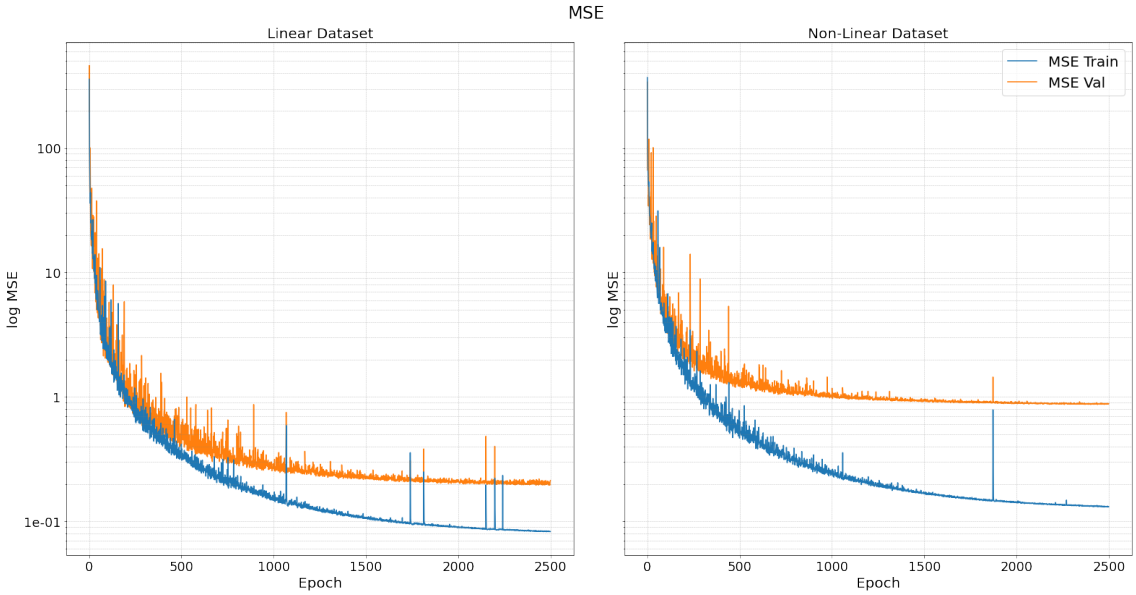


Figure 5.4: Evolution of the prediction of a testing sample along the training process of Model B

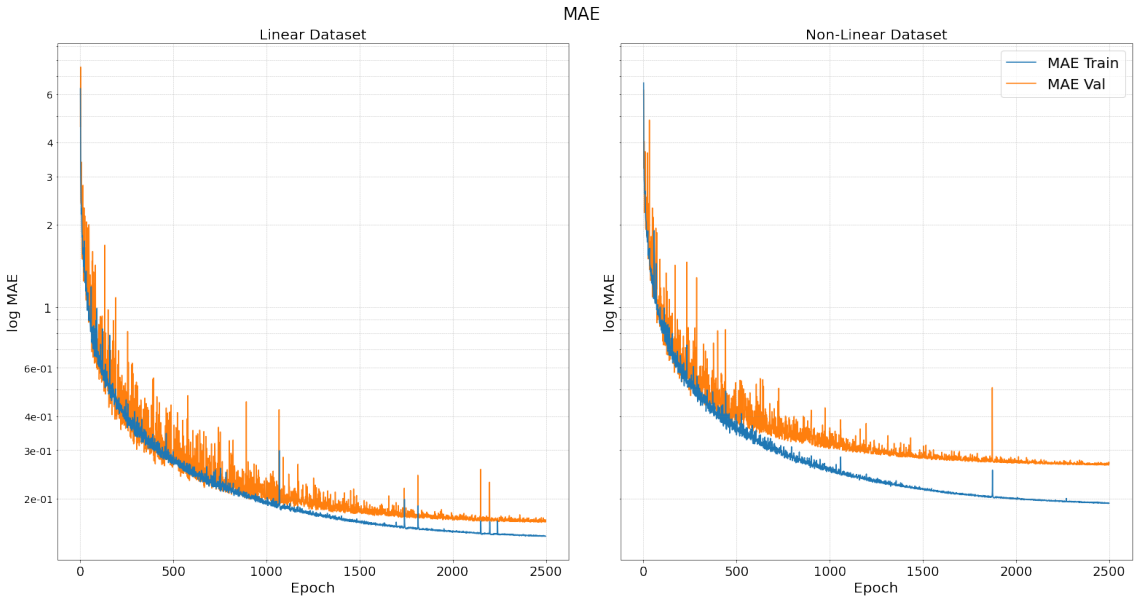


Figure 5.5: Evolution of the prediction of a testing sample along the training process of Model B

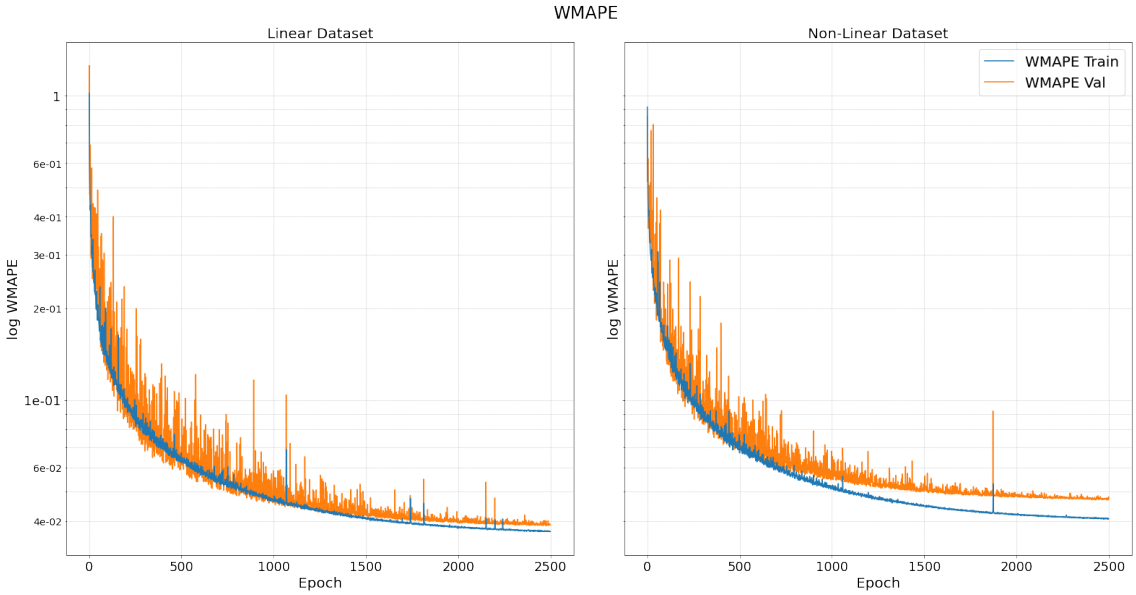


Figure 5.6: Evolution of the prediction of a testing sample along the training process of Model B

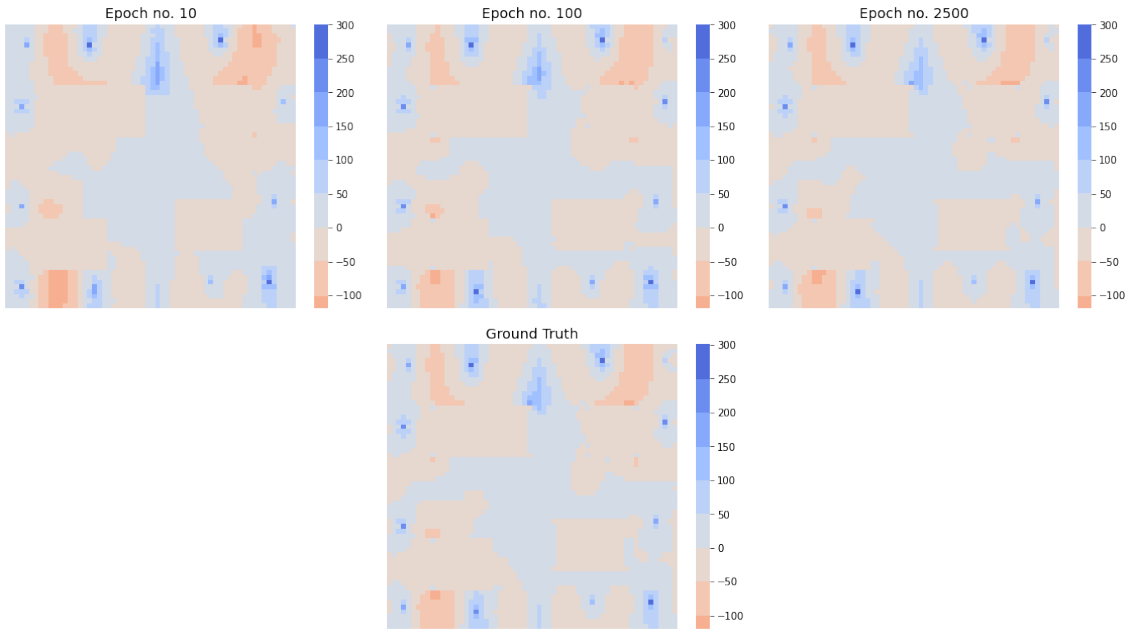


Figure 5.7: Evolution of the prediction of a testing sample along the training process

5.2 Models Testing

Identically to the previous section, this section is organized in two sub-sections, corresponding to Models A and B, respectively. Histograms of all considered error metrics as well as summary statistics and some regression plots of test samples are presented herein.

5.2.1 Model A

5.2.1.1 Error Metrics

Regarding the results obtained by Model A, Figures 5.8 to 5.15 plot the histograms of all the considered error metric distributions, namely: MAE, MSE, RMSE, PAE+, PAE-, WMAPE, PPAE+ and PPAE-. Due to the fact that all distributions present a very wide range of values, each metric (X axis) is shown at a logarithmic scale.

All metrics show positively skewed distributions, being the mean of each distribution significantly higher than its median. This fact shows that despite the majority of the instances being clustered within lower error values, there are also some observations whose error values are notably higher. Along with the distribution histograms, Figures 5.8 to 5.15 also show box-plots highlighting the outliers calculated according to the 1.5 IQR (Inter-Quartile Range) rule.

The performance of Model A on the Non-Linear dataset is worse than on the Linear dataset among all metrics, as suggested by the validation error of the training process presented in the last section, with exception for the mean PAE- which is slightly higher, in spite of the median being also lower. Despite the median WMAPE being higher on the Non-Linear dataset, the mean value is very close to that of the Linear dataset, suggesting a less skewed distribution. It is also worth noting that, despite PAE+ being higher than PAE- in both datasets, in relative terms, PPAE+ is lower and similar to PPAE- on the Linear and Non-Linear dataset, respectively. This is due to the fact that, in absolute terms, the maximum positive value of each instance is generally higher than the maximum negative. Descriptive statistics are shown in Tables 5.1 and 5.2 for the case of the Linear Dataset, and Tables 5.3 and 5.4 for the Non-Linear Dataset case.

Moreover, both maximum values (positive and negative) tend to form peaks with different shapes. While the maximum positive value tends to form a peak over a supported element, the maximum negative value occurs most of the times in between supported elements, forming a plateau-like peak, with a smoother variation among the neighboring elements. This tendency, particularly visible in Figures 5.16, 5.17, 5.18 and 5.21, does not seem to cause particular trouble to the DL model, since PPAE+ is not particularly higher than PPAE- in either datasets.

Table 5.1: Model A – Linear Dataset results – Absolute metrics

	MAE (kN.m)	**MSE ((kN.m)^2)**	**RMSE (kN.m)**	**PAE+ (kN.m)**	**PAE- (kN.m)**
mean	0.307	0.991	0.545	2.716	1.664
std. dev.	0.479	23.878	0.833	6.252	4.874
median	0.231	0.169	0.411	1.489	0.621
max	2.705×10^{1}	2.816×10^{3}	5.307×10^{1}	3.651×10^{2}	3.047×10^{2}
min	5.019×10^{-4}	2.519×10^{-7}	5.019×10^{-4}	6.104×10^{-5}	1.144×10^{-6}

Table 5.2: Model A – Linear Dataset results – Relative metrics

	WMAPE (%)	**PPAE+ (%)**	**PPAE- (%)**
mean	17.646	8.437	10.348
std. dev.	55.775	23.204	23.639
median	6.512	1.541	1.941
max	2.620×10^{3}	3.616×10^{2}	3.458×10^{2}
min	6.190×10^{-1}	8.652×10^{-5}	1.144×10^{-4}

Table 5.3: Model A – Non-Linear Dataset results – Absolute metrics

	MAE (kN.m)	**MSE ((kN.m)^2)**	**RMSE (kN.m)**	**PAE+ (kN.m)**	**PAE- (kN.m)**
mean	0.475	2.225	0.786	3.543	1.549
std. dev.	0.745	53.595	1.267	6.914	4.572
median	0.348	0.334	0.577	1.847	0.683
max	4.482×10^{1}	5.875×10^{3}	7.665×10^{1}	2.479×10^{2}	3.558×10^{2}
min	2.980×10^{-3}	8.881×10^{-6}	2.980×10^{-3}	2.441×10^{-4}	1.116×10^{-4}

Table 5.4: Model A – Non-Linear Dataset results – Relative metrics

	WMAPE (%)	**PPAE+ (%)**	**PPAE- (%)**
mean	17.756	9.447	9.444
std. dev.	27.672	23.624	23.020
median	8.984	2.254	2.228
max	6.870×10^{2}	3.038×10^{2}	3.199×10^{2}
min	6.962×10^{-1}	9.799×10^{-5}	3.415×10^{-4}

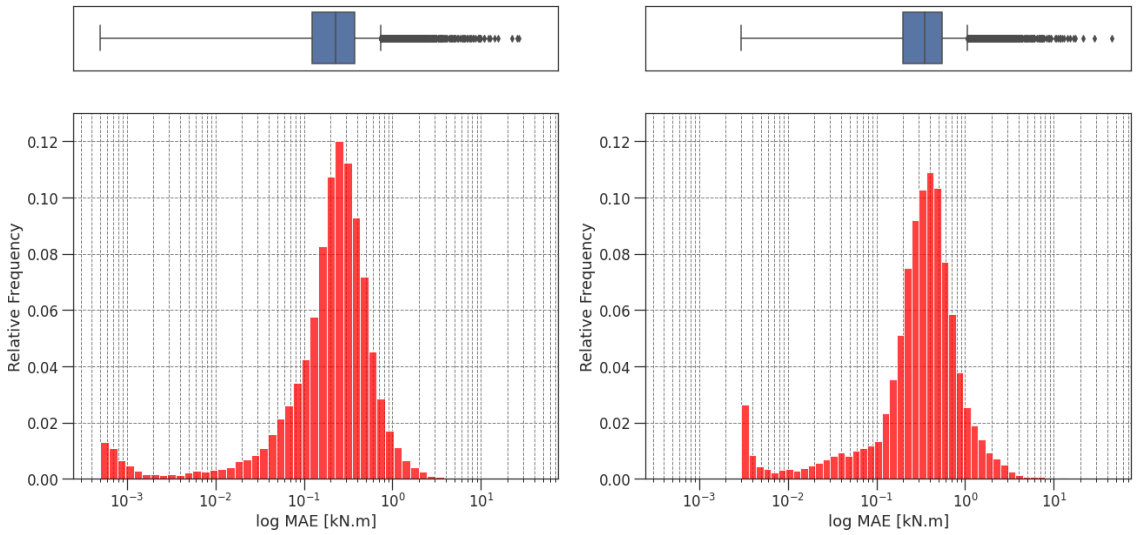


Figure 5.8: Mean Absolute Error (MAE) histograms of the results obtained by Model A for the Linear Dataset (LHS) and Non-Linear Dataset (RHS)

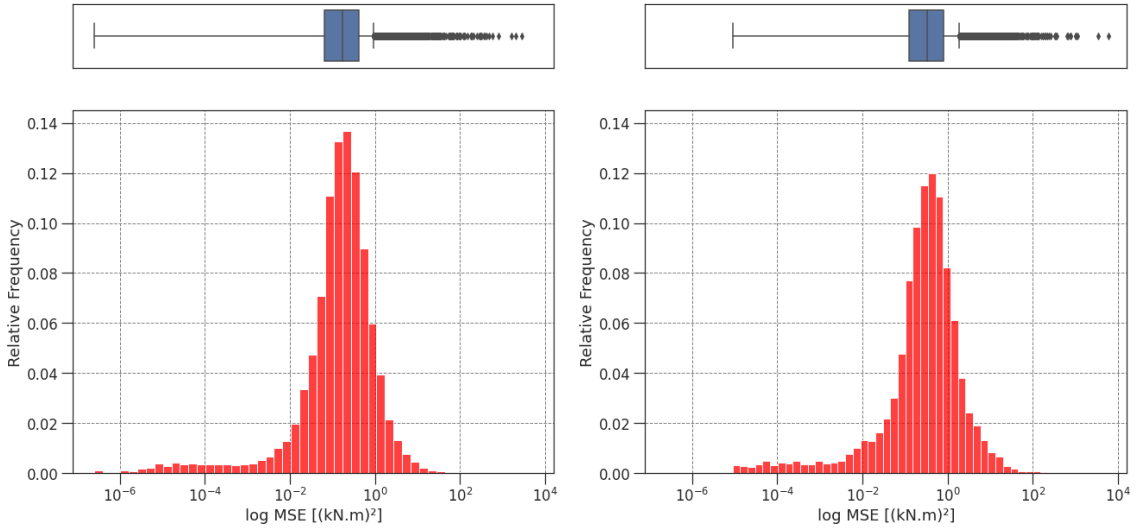


Figure 5.9: Mean Squared Error (MSE) histograms of the results obtained by Model A for the Linear Dataset (LHS) and Non-Linear Dataset (RHS)

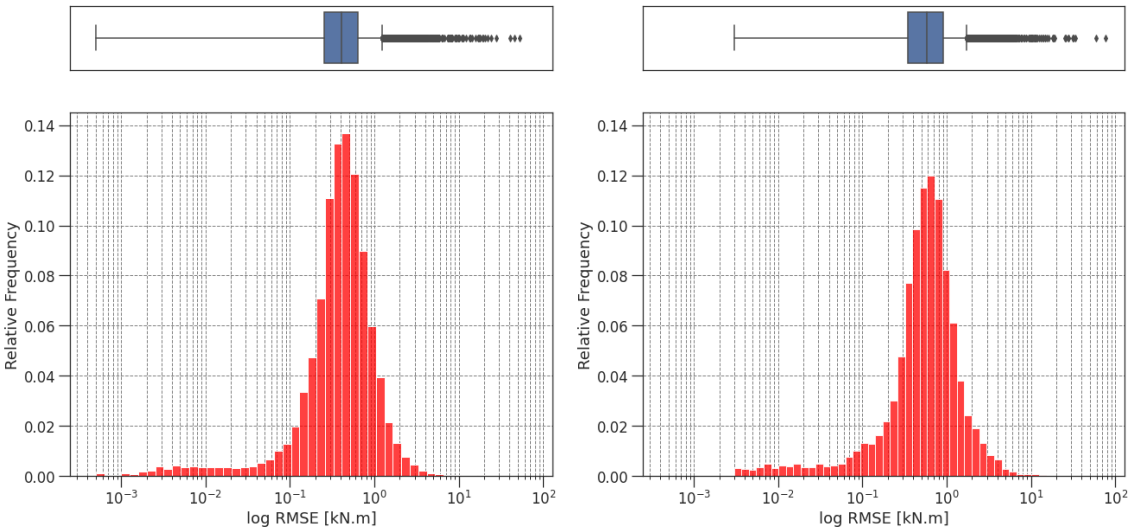


Figure 5.10: Root Mean Squared Error (RMSE) histograms of the results obtained by Model A for the Linear Dataset (LHS) and Non-Linear Dataset (RHS)

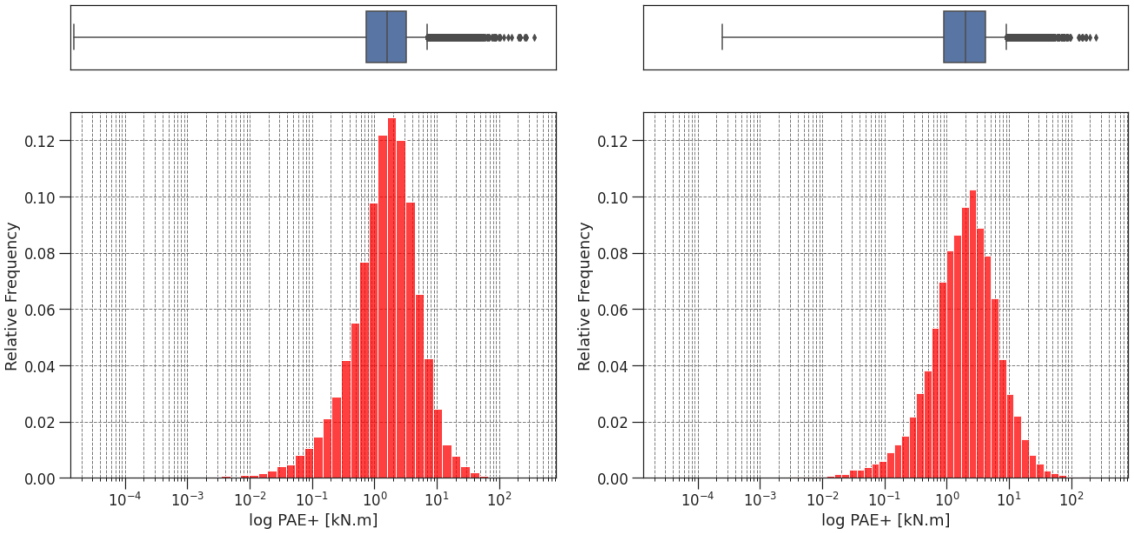


Figure 5.11: Peak Absolute Error applied to positive values (PAE+) histograms of the results obtained by Model A for the Linear Dataset (LHS) and Non-Linear Dataset (RHS)

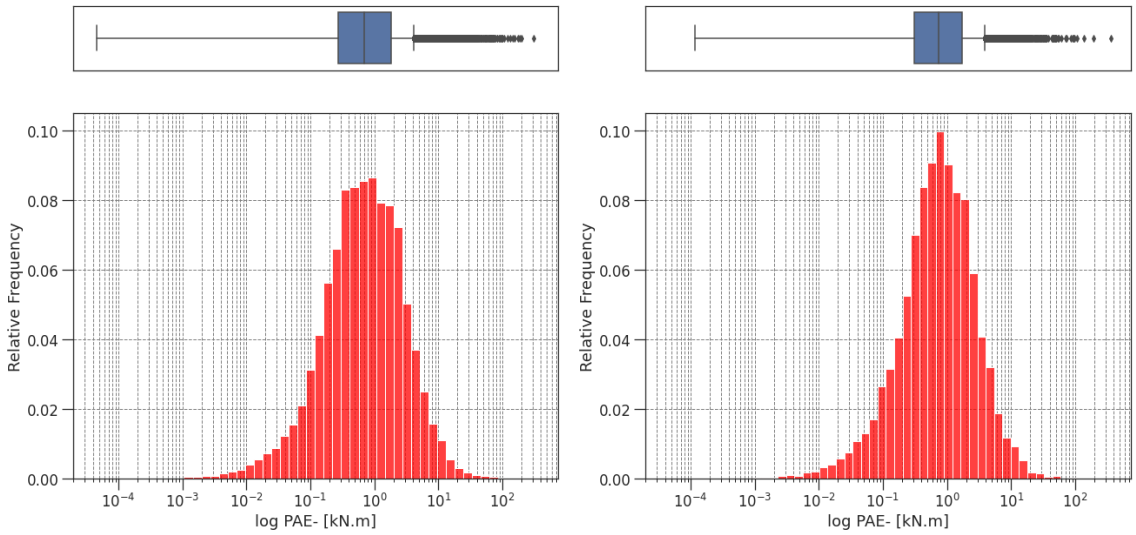


Figure 5.12: Peak Absolute Error applied to negative values (PAE-) histograms of the results obtained by Model A for the Linear Dataset (LHS) and Non-Linear Dataset (RHS)

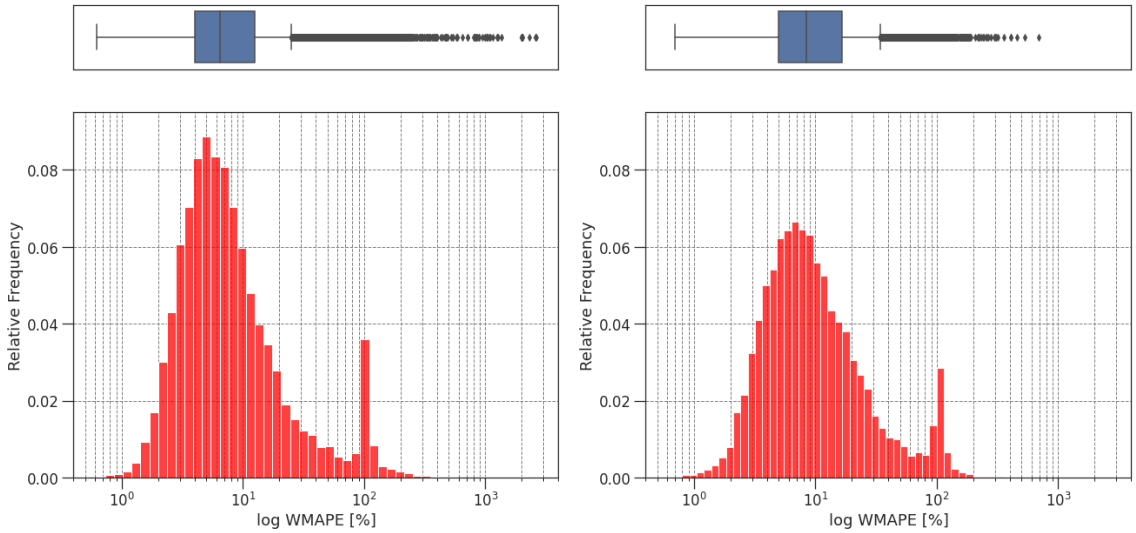


Figure 5.13: Weighted Mean Absolute Percentage Error (WMAPE) histograms of the results obtained by Model A for the Linear Dataset (LHS) and Non-Linear Dataset (RHS)

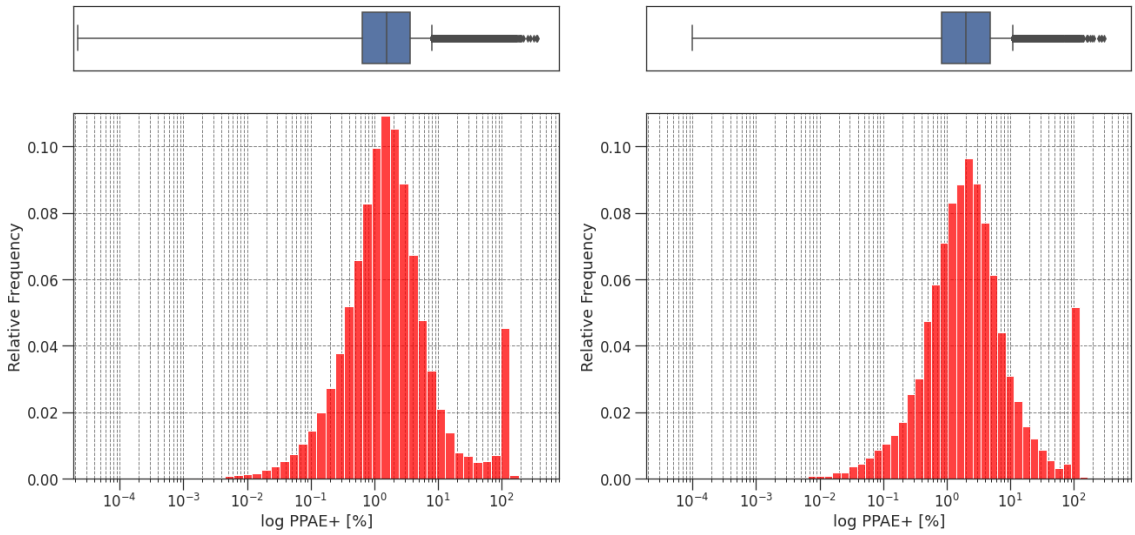


Figure 5.14: Percentage Peak Absolute Error applied to positive values (PPAE+) histograms of the results obtained by Model A for the Linear Dataset (LHS) and Non-Linear Dataset (RHS)

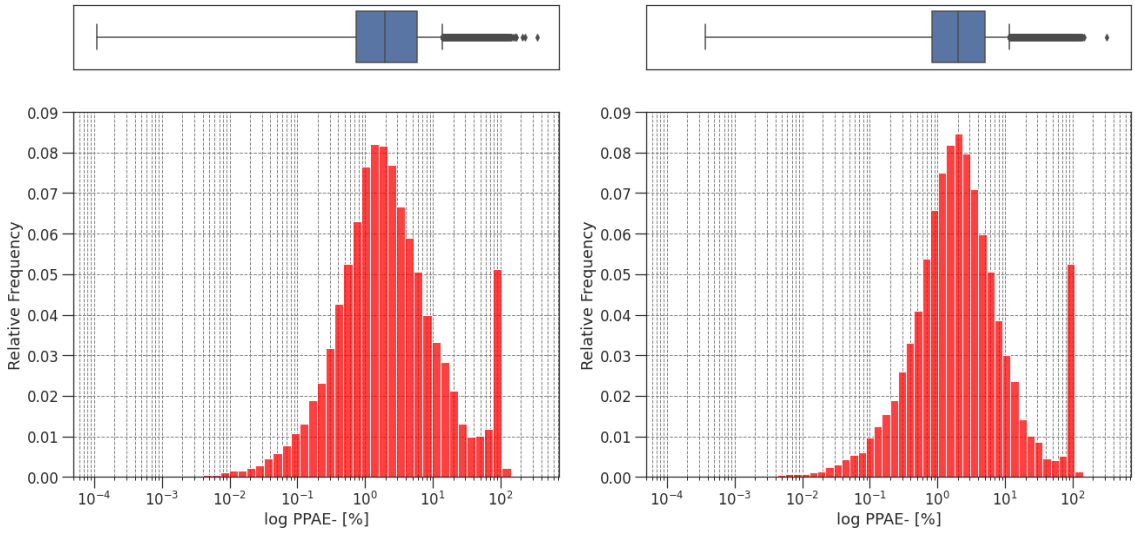


Figure 5.15: Percentage Peak Absolute Error applied to negative values (PPAE-) histograms of the results obtained by Model A for the Linear Dataset (LHS) and Non-Linear Dataset (RHS)

5.2.1.2 Result Samples

In order to illustrate the results presented in the past section, some prediction plots of sample instances are presented herein. The chosen instances are those where the median and maximum MSE and WMAPE were observed, *i.e.* the absolute error metric chosen as loss function for the training process and a relative error metric.

The choice for the median observation over the mean is due to the fact that the error metric distributions are positively skewed. Thus, by not being so affected by extreme outliers, the median is believed to be a better measure of central tendency in this case. By showcasing the instances

where the maximum errors were observed, it is intended to outline the inferior limit to the quality of the obtained results. Each of these plots outline the input features at the top line, followed by the regression result (Prediction), results obtained by the FEM model (Ground Truth) and the absolute error (|GT - Pred|) in the bottom line.

Figures 5.16 and 5.17 plot the results for the instances where the median MSE and maximum MSE were observed, respectively. The former presents a prediction visibly close to the ground truth, as it can be confirmed by the map of absolute errors (|GT - Pred|) where most of the elements unveil values very close to 0. The same does not apply to the latter, where the differences between the prediction and the ground truth are evident. The error is especially high in the surroundings of the element located at row 9 and column 34, even though there are no loads applied within that area.

Figures 5.18 and 5.19 outline the results for the samples where the median WMAPE and maximum WMAPE were observed, respectively. Like the case of MSE, the results of the median WMAPE instance are also visibly close to the ground truth. Regarding the maximum WMAPE instance, it is an instance with a single loaded element (as outlined in the "Loads" feature plot). This fact implies that the bending moments are generally low across the ground truth map. Once WMAPE is a relative measure, even low absolute prediction errors lead to a high WMAPE value.

From Figure 5.20 onward the result plots concern the Non-Linear dataset. This plot outlines the results for the instance with median MSE. Even put up to the Non-Linear dataset, known to be more mathematically complex in terms of FEM calculations, Model A is able to achieve results very close to the ground truth, as evidenced by the absolute error plot and by visually comparing the Prediction plot against the Ground Truth one. Curiously, the instance of maximum MSE of the Non-Linear dataset, depicted in Figure 5.21, is similar, in terms of the "Geometry" and "Supports" input features, to that of the Linear case. This time, most of the errors are located in the area under loading, but also around the inner corners of the geometry void.

Finally, the instances of median and maximum WMAPE for the Non-Linear dataset are very similar, in terms of analysis, to those of the Linear dataset. The results of the instance of median WMAPE reveals a near perfect approximation, while the case of the instance of maximum WMAPE reveals a generally low absolute error, but high relative error for being subject to low bending moments across its elements. These results are shown in Figures 5.22 and 5.23, respectively.

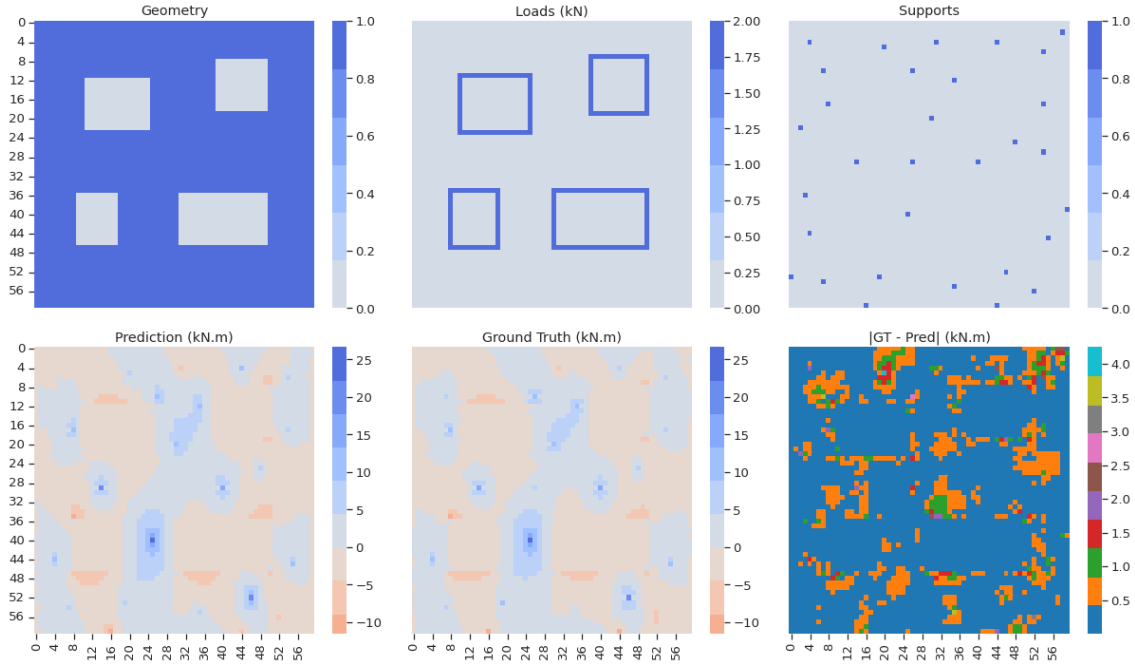


Figure 5.16: Model A – Linear Dataset – Regression results plot of the instance with median MSE

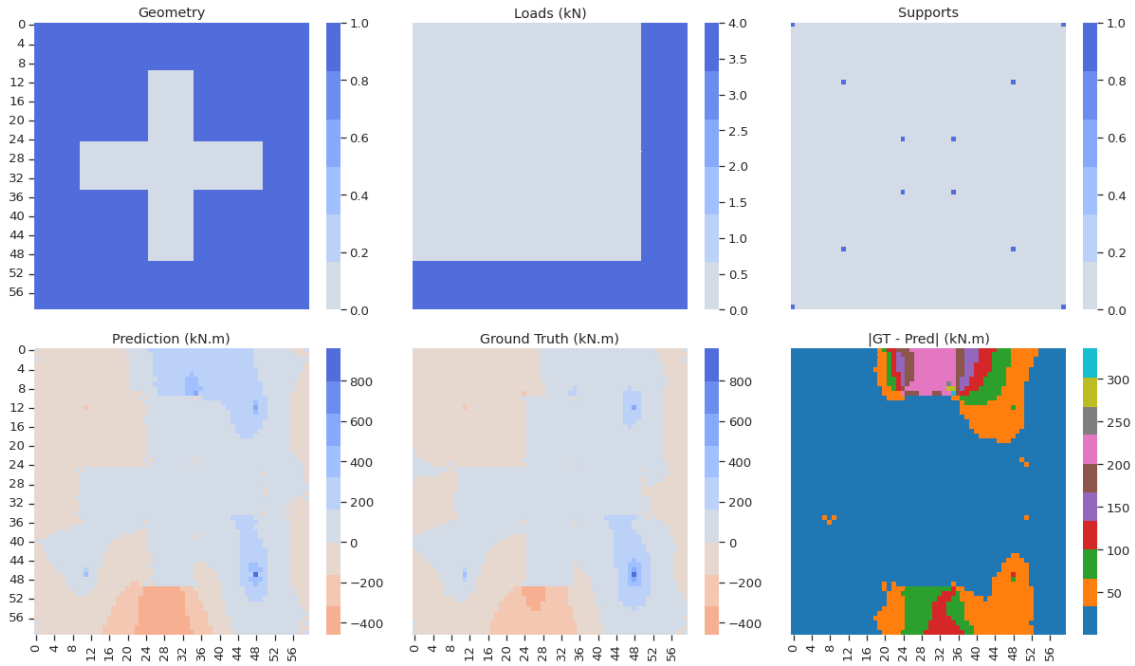


Figure 5.17: Model A – Linear Dataset – Regression results plot of the instance with maximum MSE

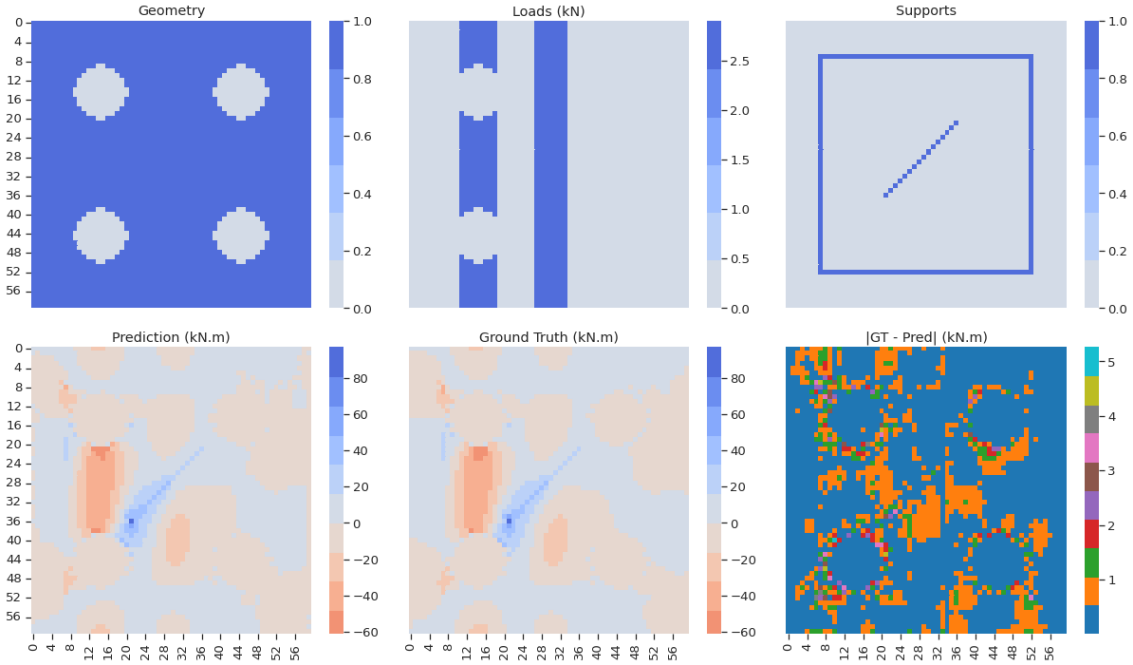


Figure 5.18: Model A – Linear Dataset – Regression results plot of the instance with median WMAPE

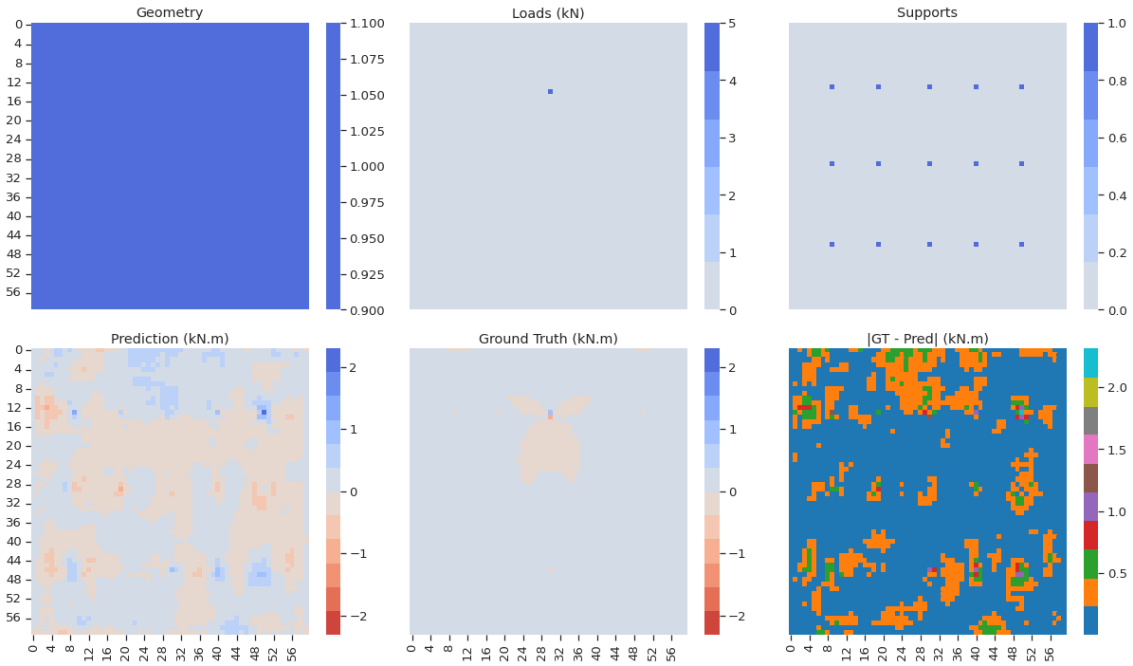


Figure 5.19: Model A – Linear Dataset – Regression results plot of the instance with maximum WMAPE

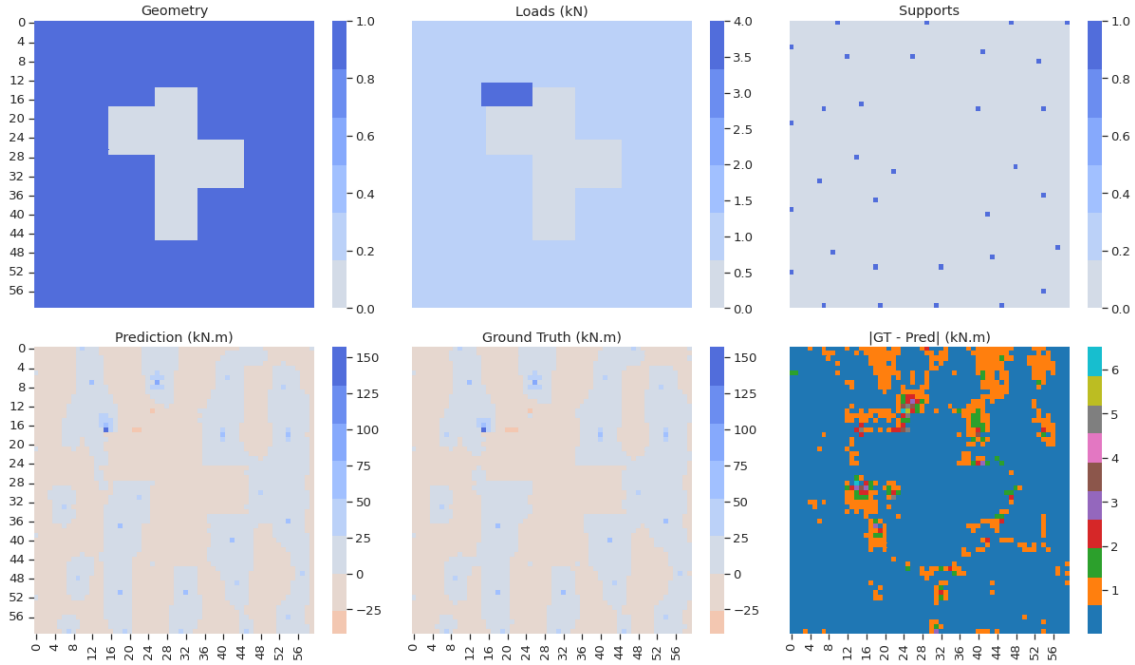


Figure 5.20: Model A – Non-Linear Dataset – Regression results plot of the instance with median MSE

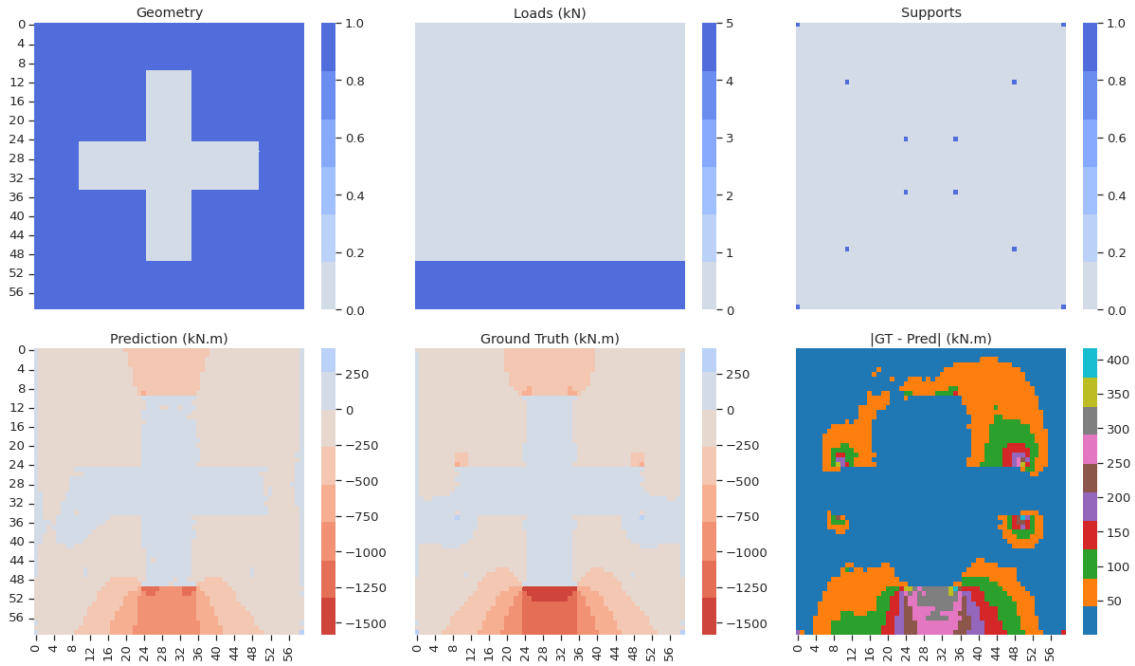


Figure 5.21: Model A – Non-Linear Dataset – Regression results plot of the instance with maximum MSE

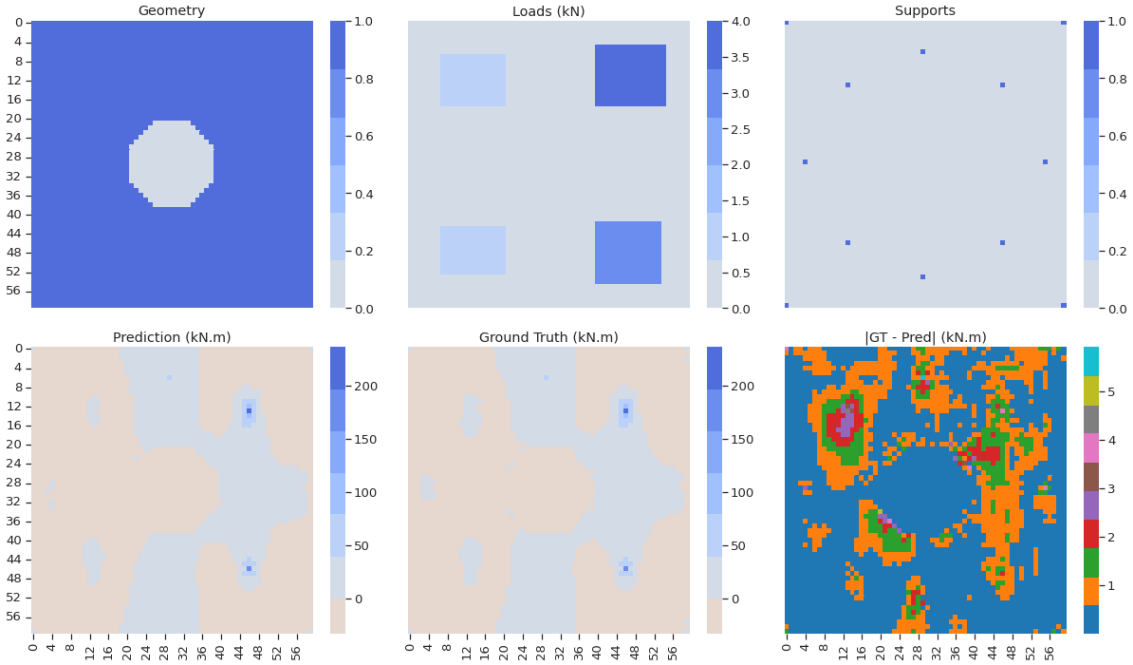


Figure 5.22: Model A – Non-Linear Dataset – Regression results plot of the instance with median WMAPE

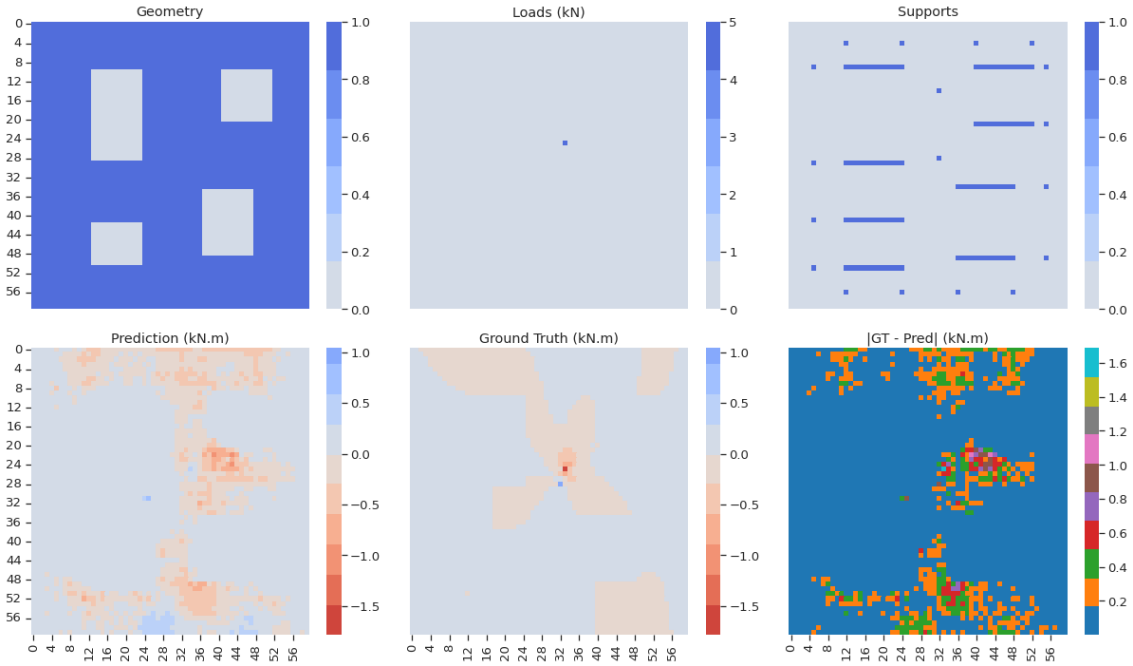


Figure 5.23: Model A – Non-Linear Dataset – Regression results plot of the instance with maximum WMAPE

5.2.1.3 Run-time

As previously mentioned, both deep learning models were trained on a NVIDIA Tesla P100 provided by Google Colab platform. Since the commercial FEM software used to generate the dataset does not take advantage of a GPU and that it was ran on a local computer, in order to compare their running times it is imperative that both are ran on the same setup. To this end, the test set was run on Model A, on the local CPU, loaded with the weights obtained from the training process. This so-called local setup is composed of an AMD Ryzen 5600H CPU @ 3.300 GHz, with 16GB DDR4 RAM @ 3200 MHz.

The run-time comparison between Model A and the commercial FEM software for both Linear and Non-Linear calculation approaches, per 100 calculated instances, is presented in Tables 5.5 and 5.6, respectively. As expected, the run-times of Model A on both datasets is virtually the same, since it is the same model. The FEM Model on the Linear calculation approach presents a slight variability due to the number of finite elements being variable among instances, contrary to the Non-Linear approach, where a large variability may be observed, not only because of the different number of finite elements varying among instances, but also because it is an iterative calculation method whose solution may be closer or farther depending on the complexity of the problem. It is also important to note that each run of the FEM model computes not only the bending moments along X direction (as Model A and Model B do), but all sorts of stresses and displacements, despite bending stresses are the most preponderant for the type of loads considered in this dataset. Considering this fact, to get a full characterization of the bending stresses of an instance (along directions X and Y), as presented in section 3.3, the deep learning model would have to be ran twice, rotating the input data by 90 degrees in one of the runs, for obtaining the bending moments along Y direction.

Table 5.5: Model A – Linear Dataset results – Processing time per 100 instances

	FEM Model (s)	**Model A (s)**
mean	14.627	1.010
std. dev.	1.869	0.031
median	15.034	1.004

Table 5.6: Model A – Non-Linear Dataset results – Processing time per 100 instances

	FEM Model (s)	**Model A (s)**
mean	1333.104	0.987
std. dev.	764.089	0.012
median	1067.077	0.987

5.2.2 Model B

5.2.2.1 Error Metrics

Following the exposed for Model A, Figures 5.24 to 5.31 plot the histograms and box-plots for all the considered error metric distributions at a logarithmic scale.

Identically to the error metrics obtained by Model A, those of Model B also present positive skewness, *i.e.* most of the observations concentrate at lower error values, but there are undoubtedly some instances presenting higher error values.

Also similar to Model A is the fact that Model B performs worse on the Non-Linear dataset compared to the Linear dataset, except for the mean values of WMAPE and PPAE-, contrary to their median. It also becomes evident that despite the similarities, Model B outperforms Model A for either Linear and Non-Linear datasets. Regarding errors at peak values, like in Model A, PPAE+ shows slightly lower errors than PPAE- in the Linear dataset, and similar results in the Non-Linear dataset. Descriptive statistics are presented in Tables 5.7 and 5.8 for the Linear case, and Tables 5.9 and 5.10 for the Non-Linear case.

Table 5.7: Model B – Linear Dataset results – Absolute metrics

	MAE (kN.m)	**MSE ((kN.m)^2)**	**RMSE (kN.m)**	**PAE+ (kN.m)**	**PAE- (kN.m)**
mean	0.167	0.272	0.293	1.420	0.916
std. dev.	0.251	6.692	0.431	3.310	3.162
median	0.125	0.049	0.222	0.784	0.341
max	2.070×10^{1}	9.781×10^{2}	3.127×10^{1}	2.325×10^{2}	2.160×10^{2}
min	8.125×10^{-3}	2.873×10^{-4}	1.695×10^{-2}	0.000	0.000

Table 5.8: Model B – Linear Dataset results – Relative metrics

	WMAPE (%)	**PPAE+ (%)**	**PPAE- (%)**
mean	16.122	5.900	7.505
std. dev.	56.750	18.911	20.186
median	3.577	0.794	1.045
max	1.231×10^{3}	1.846×10^{2}	1.896×10^{2}
min	3.033×10^{-1}	4.356×10^{-5}	1.413×10^{-4}

Table 5.9: Model B – Non-Linear Dataset results – Absolute metrics

	MAE (kN.m)	**MSE ((kN.m)^2)**	**RMSE (kN.m)**	**PAE+ (kN.m)**	**PAE- (kN.m)**
mean	0.261	0.830	0.435	1.955	0.921
std. dev.	0.457	19.100	0.800	4.212	2.895
median	0.185	0.096	0.310	1.026	0.406
max	1.846×10^{1}	1.723×10^{3}	4.151×10^{1}	1.444×10^{2}	1.55×10^{2}
min	7.468×10^{-3}	2.253×10^{-4}	1.501×10^{-2}	0.000	0.000

Table 5.10: Model B – Non-Linear Dataset results – Relative metrics

	WMAPE (%)	**PPAE+ (%)**	**PPAE- (%)**
mean	14.848	7.173	6.800
std. dev.	50.855	22.490	19.360
median	4.411	1.071	1.173
max	3.288×10^{3}	4.081×10^{2}	2.665×10^{2}
min	4.371×10^{-1}	1.401×10^{-4}	1.428×10^{-5}

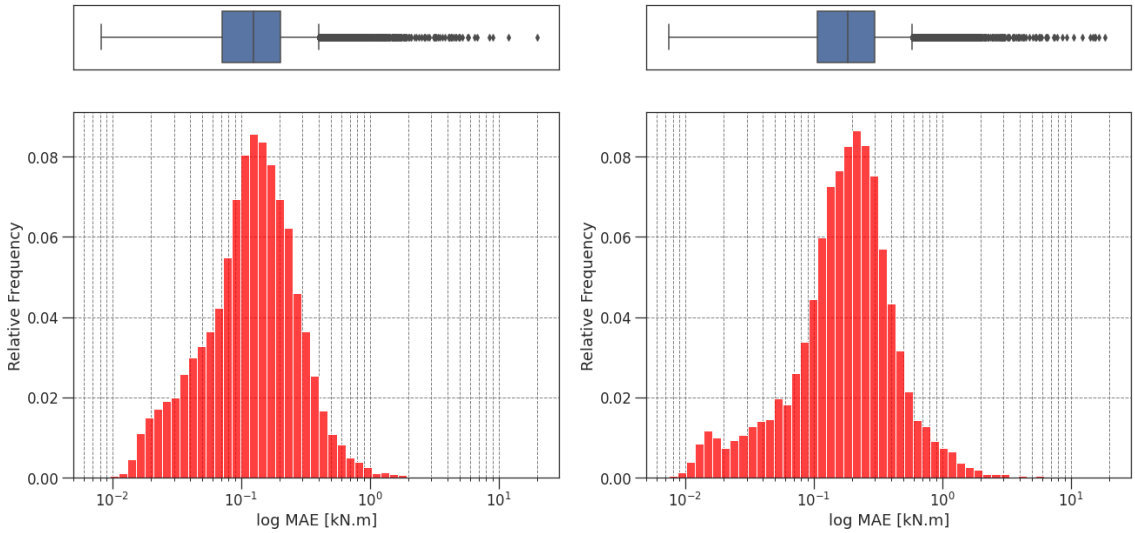


Figure 5.24: Mean Absolute Error (MAE) histograms of the results obtained by Model B for the Linear Dataset (LHS) and Non-Linear Dataset (RHS)

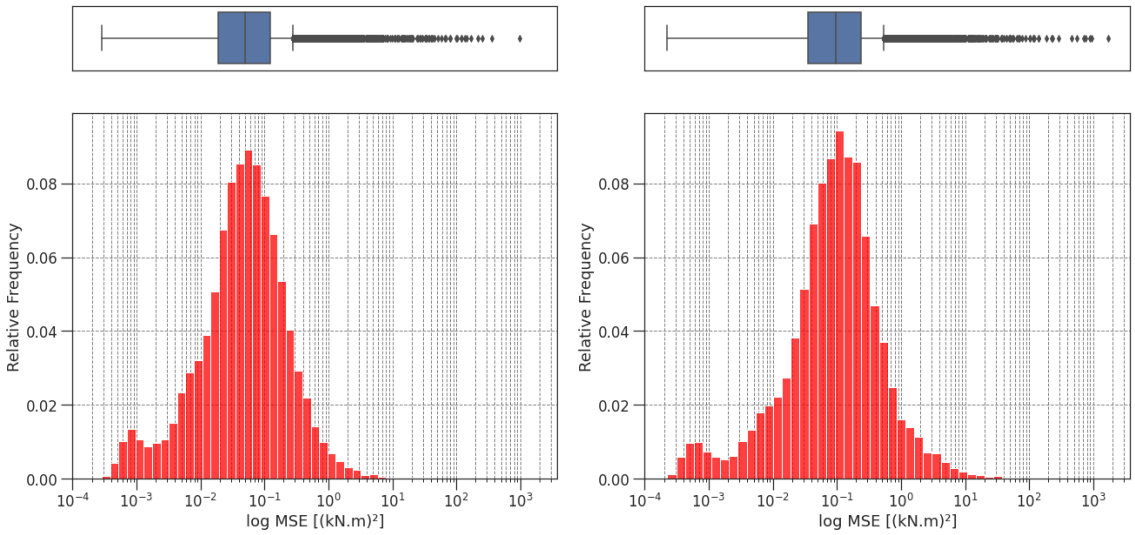


Figure 5.25: Mean Squared Error (MSE) histograms of the results obtained by Model B for the Linear Dataset (LHS) and Non-Linear Dataset (RHS)

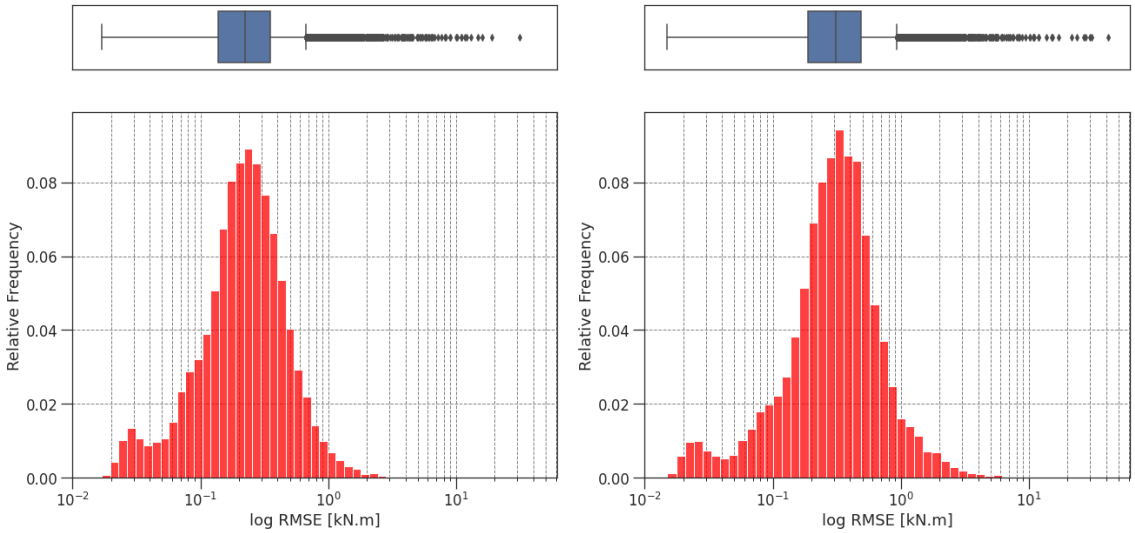


Figure 5.26: Root Mean Squared Error (RMSE) histograms of the results obtained by Model B for the Linear Dataset (LHS) and Non-Linear Dataset (RHS)

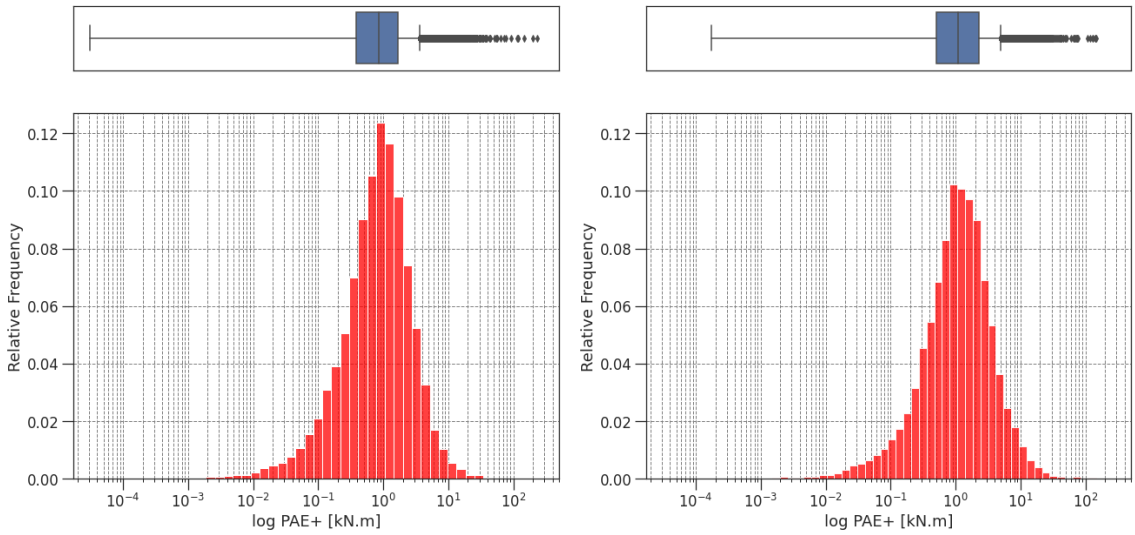


Figure 5.27: Peak Absolute Error applied to positive values (PAE+) histograms of the results obtained by Model B for the Linear Dataset (LHS) and Non-Linear Dataset (RHS)

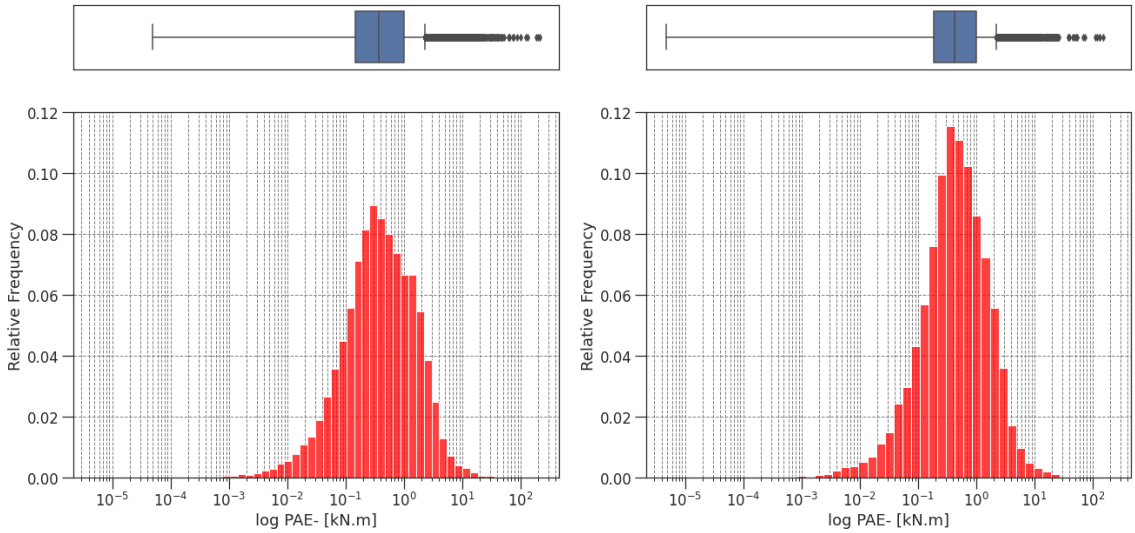


Figure 5.28: Peak Absolute Error applied to negative values (PAE-) histograms of the results obtained by Model B for the Linear Dataset (LHS) and Non-Linear Dataset (RHS)

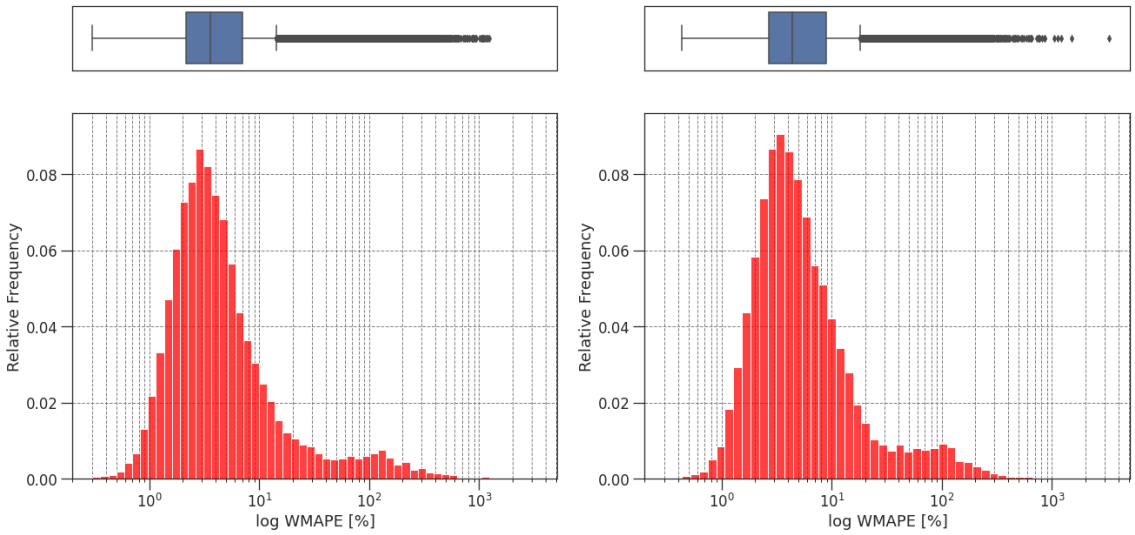


Figure 5.29: Weighted Mean Absolute Percentage Error (WMAPE) histograms of the results obtained by Model B for the Linear Dataset (LHS) and Non-Linear Dataset (RHS)

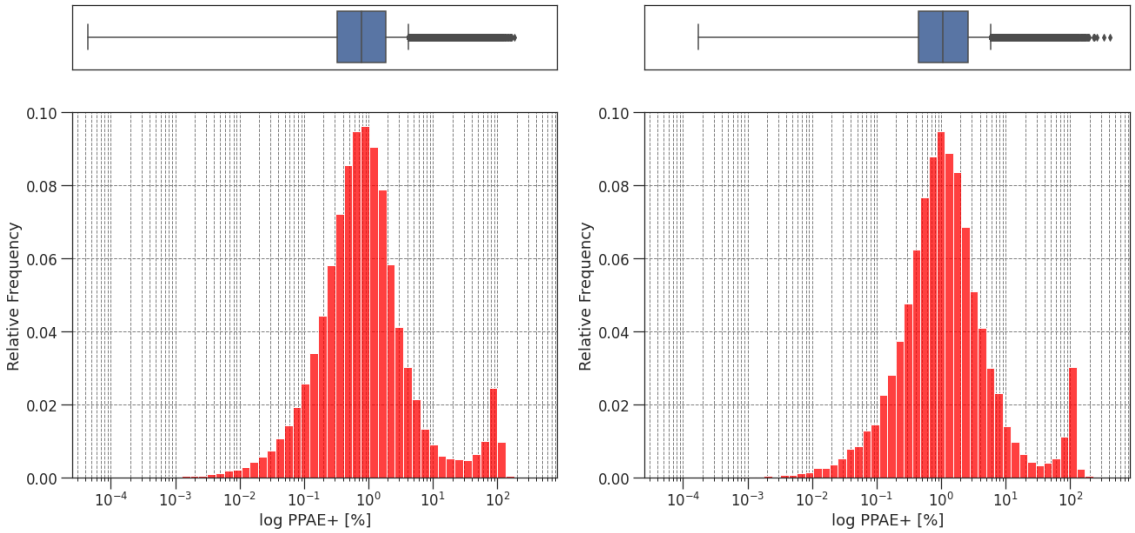


Figure 5.30: Percentage Peak Absolute Error applied to positive values (PPAE+) histograms of the results obtained by Model B for the Linear Dataset (LHS) and Non-Linear Dataset (RHS)

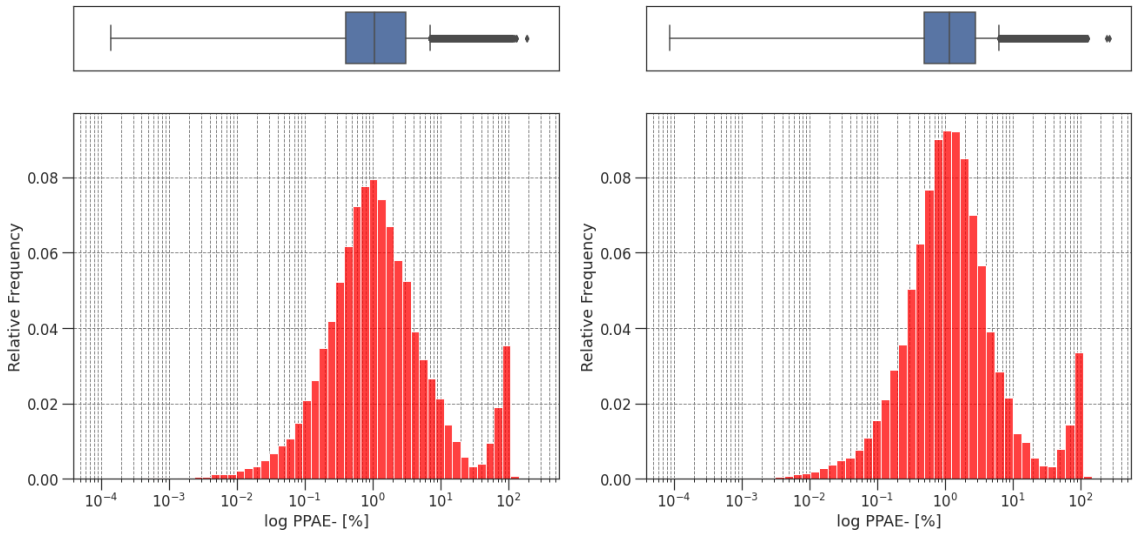


Figure 5.31: Percentage Peak Absolute Error applied to negative values (PPAE-) histograms of the results obtained by Model B for the Linear Dataset (LHS) and Non-Linear Dataset (RHS)

5.2.2.2 Result Samples

In the same way as with Model A, prediction plots of the instances corresponding to the median and maximum MSE and WMAPE are presented within this section, so as to illustrate the results debated in the previous section. In general, the results are qualitatively equivalent to those of Model A, *i.e.*, the prediction results obtained for the instances of median MSE and WMAPE are very close to the ground truth, while the errors of the instances of maximum MSE are unmistakenly high and the instances of higher WMAPE are single loaded instances, with high relative error, but low absolute error. The results for the instances of median and maximum MSE and WMAPE of the Linear dataset respectively are shown from Figure 5.32 to Figure 5.35. As for the Non-Linear dataset, by the same order, the results are presented from Figure 5.36 to 5.39.

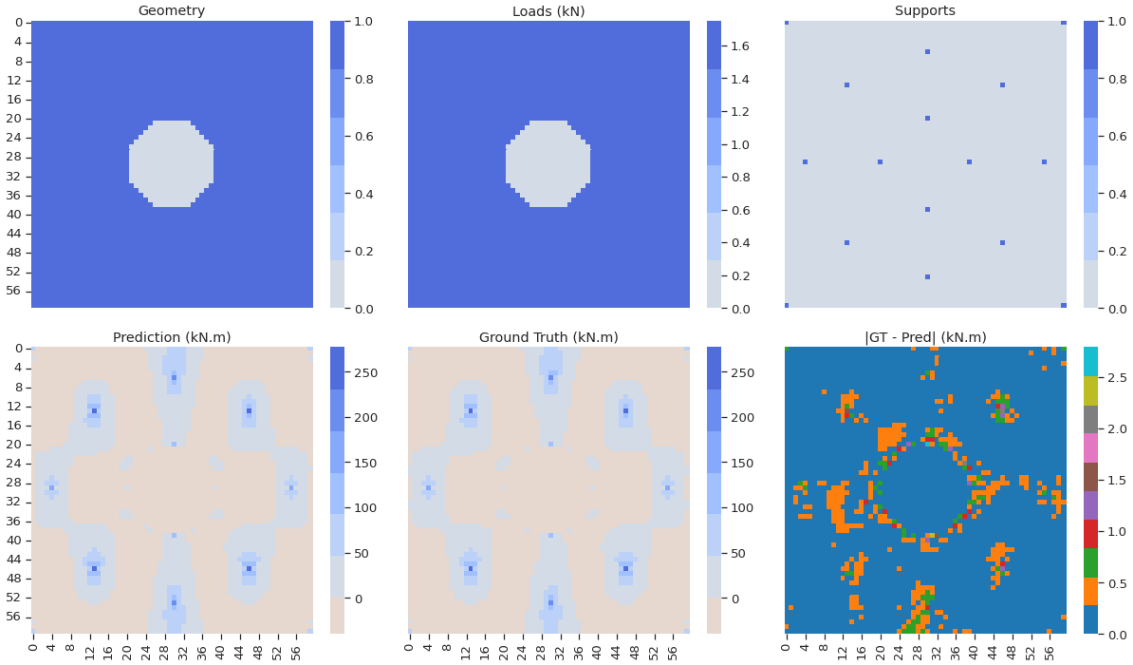


Figure 5.32: Model B – Linear Dataset – Regression results plot of the instance with median MSE

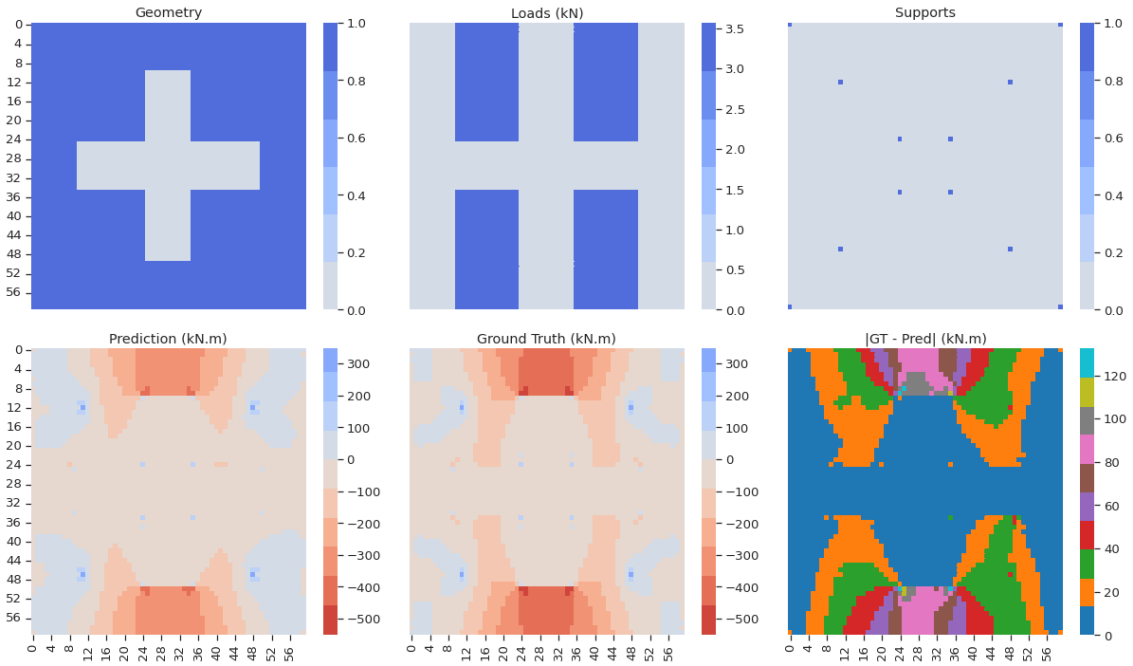


Figure 5.33: Model B – Linear Dataset – Regression results plot of the instance with maximum MSE

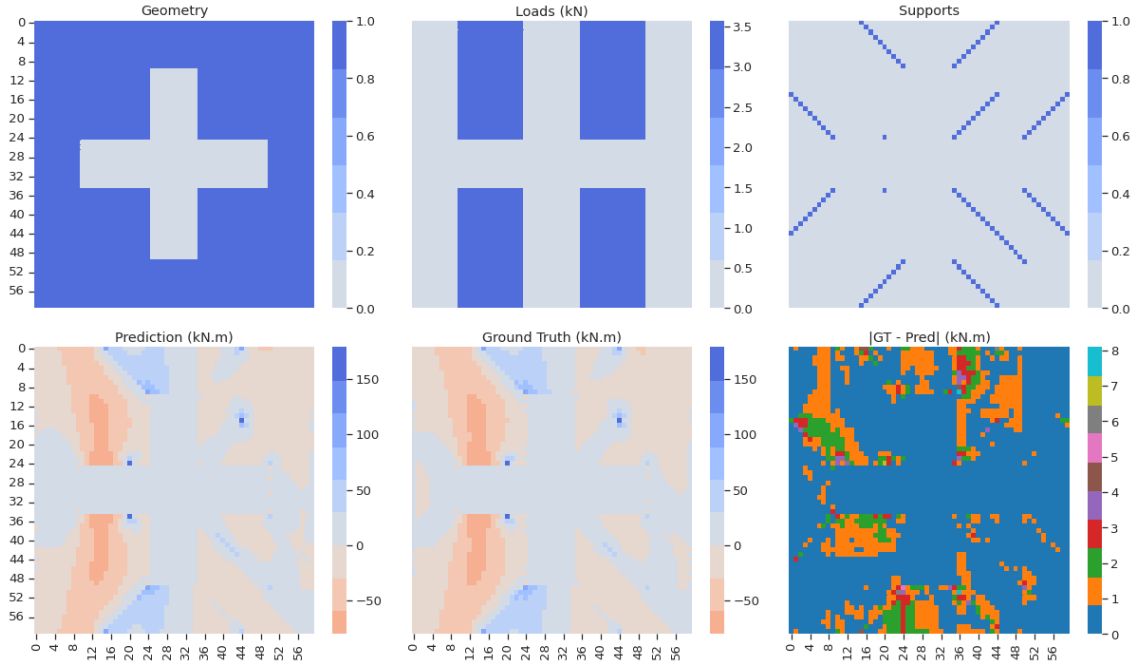


Figure 5.34: Model B – Linear Dataset – Regression results plot of the instance with median WMAPE

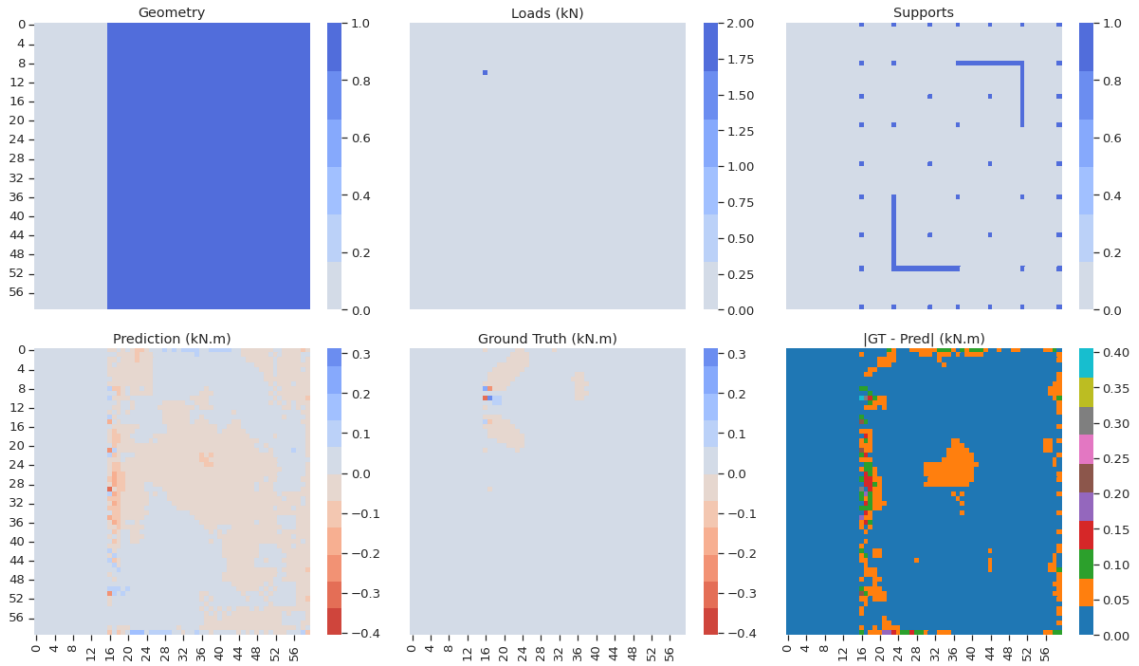


Figure 5.35: Model B – Linear Dataset – Regression results plot of the instance with maximum WMAPE

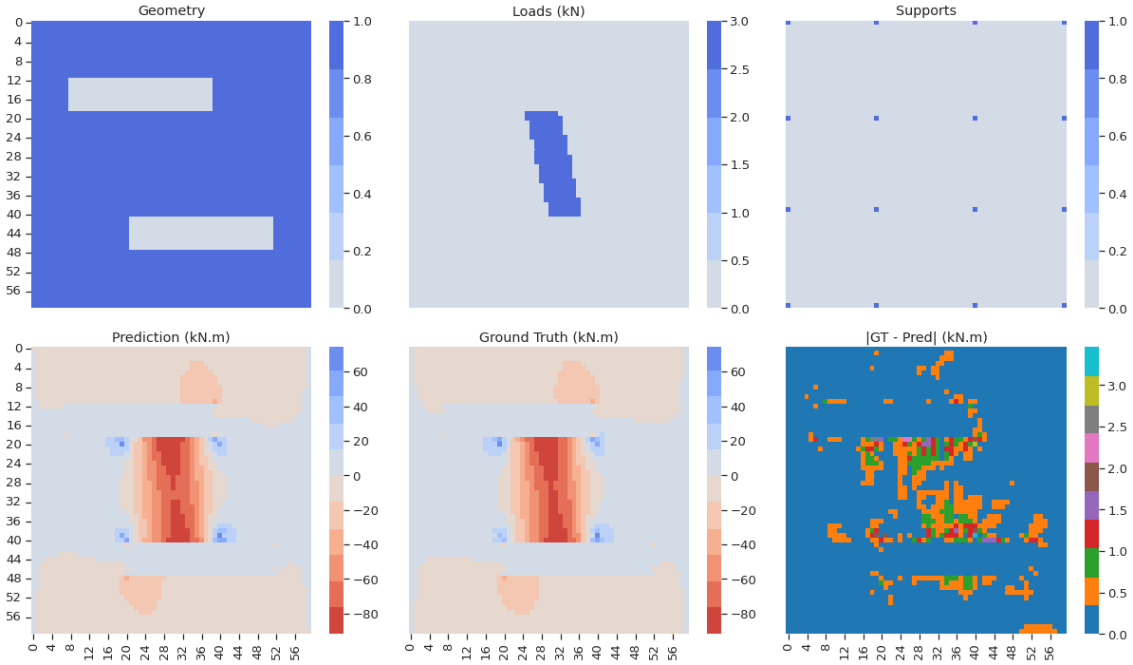


Figure 5.36: Model B – Non-Linear Dataset – Regression results plot of the instance with median MSE

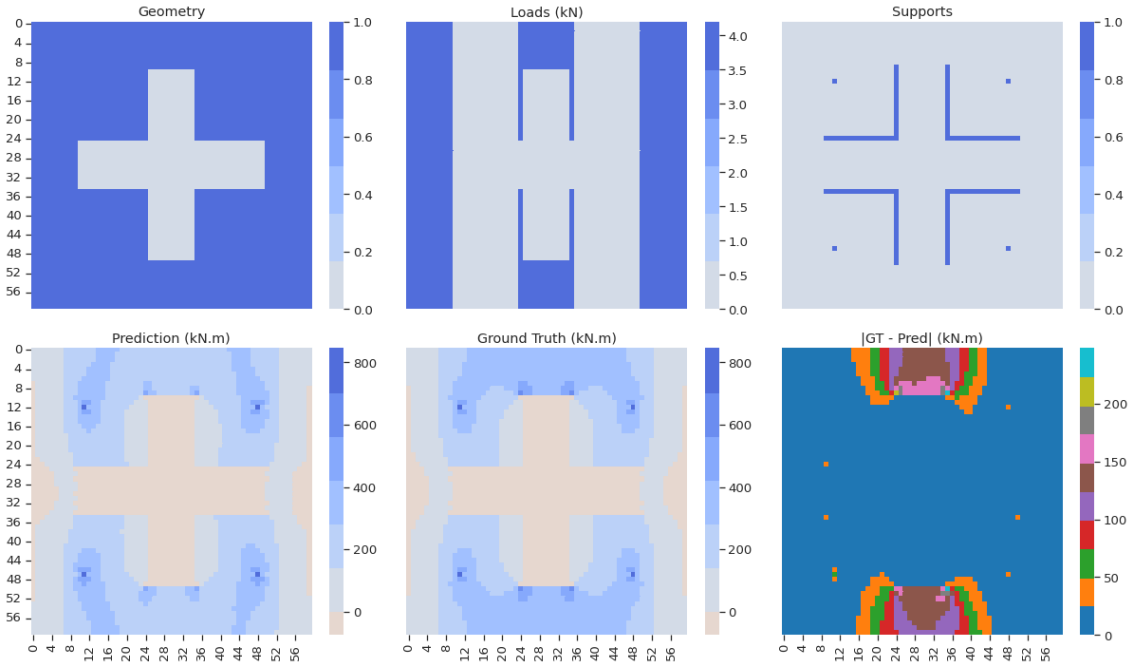


Figure 5.37: Model B – Non-Linear Dataset – Regression results plot of the instance with maximum MSE

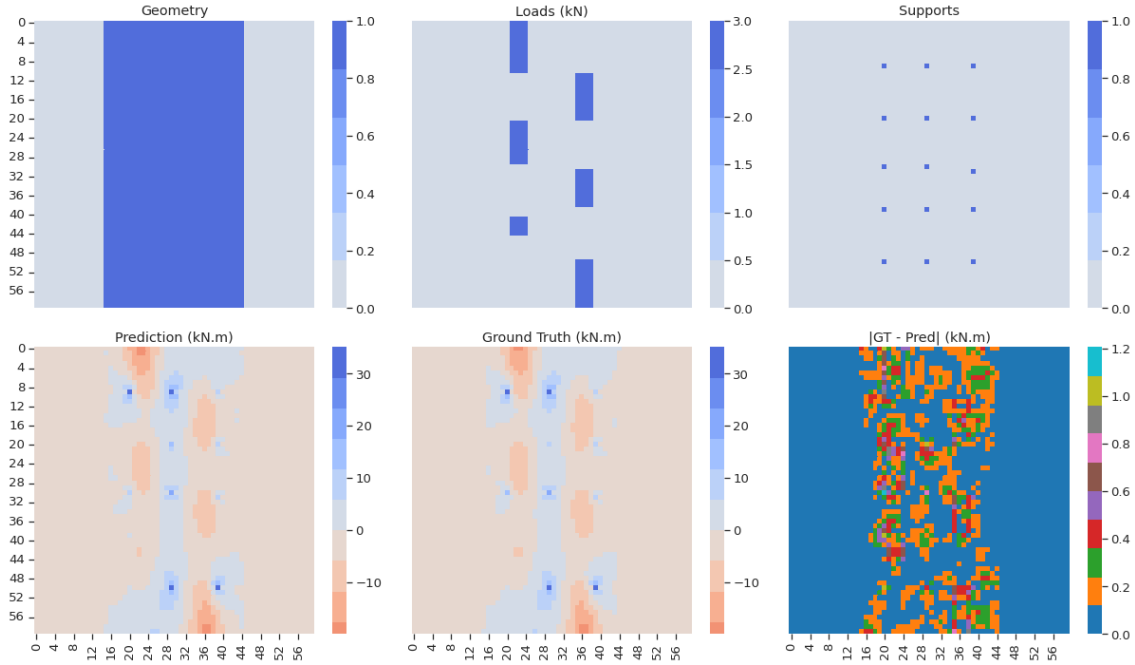


Figure 5.38: Model B – Non-Linear Dataset – Regression results plot of the instance with median WMAPE

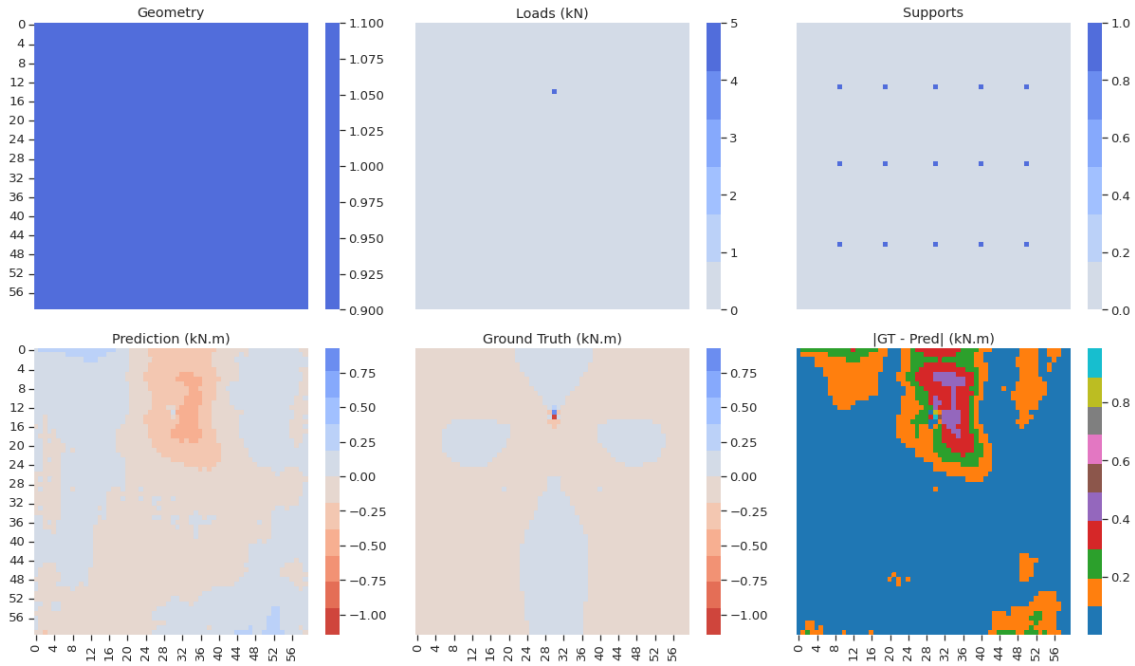


Figure 5.39: Model B – Non-Linear Dataset – Regression results plot of the instance with maximum WMAPE

5.2.2.3 Run-time

Like with Model A, Model B was also ran in the local setup loaded with the weights obtained from the training process. The run-time results in comparison with the FEM model are presented in Tables 5.11 and 5.12 for the Linear and Non-Linear approaches, respectively. Like with Model A, Model B takes virtually the same time to run regardless of the dataset in analysis. The higher complexity of Model B when compared to Model A comes at a cost of an increase of roughly 65% on the run-time. Nevertheless, in absolute terms, it can still be considered a low computation time when compared to the FEM model.

Table 5.11: Model B – Linear Dataset results – Processing time per 100 instances

	FEM Model (s)	**Model B (s)**
mean	14.627	1.664
std. dev.	1.869	0.059
median	15.034	1.657

Table 5.12: Model B – Non-Linear Dataset results – Processing time per 100 instances

	FEM Model (s)	**Model B (s)**
mean	1333.104	1.645
std. dev.	764.089	0.028
median	1067.077	1.638

5.3 Discussion

The main takeaways from the previous sections are that both models output results with very different levels of quality, *i.e.* with discrepant error metrics among each other, that Model B is able to achieve better results than Model A independently of the dataset in analysis, and that both models are more erroneous against the Non-Linear dataset than against the Linear dataset. The present section focuses in understanding what the magnitude of these error metrics represents in practise, as well as trying to circumscribe the worst-performing instances by means of unraveling singular characteristics of their input features. Taking into account the success criteria presented in section 4.4, Table 5.13 summarizes the success instance counts and rates among both models and datasets.

Table 5.13: Success instance counts and rates among models and datasets

Instances	**Linear Dataset**		**Non-Linear Dataset**	
	Model A	**Model B**	**Model A**	**Model B**
Successful	23 405	27 727	13 251	16 142
Total	30 734	30 734	18 348	18 348
Successful (%)	**76.153**	**90.216**	**72.220**	**87.977**

Focusing on the unsuccessful instances, efforts were made in order to find patterns within the input features that could lead to such high error metrics. To that end, the following set of features was engineered from the three input features (Geometry, Loads and Supports):

- **geom_group (categorical)**: Original geometry group (Figure 3.2) to which the geometry of an instance corresponds, despite being potentially rotated and/or flipped;
- **load_count (numerical)**: Count of loaded elements of each instance regardless of its magnitude;
- **load_sum (numerical)**: Sum of applied loads among elements of each instance;
- **sup_count (numerical)**: Count of supported elements of each instance;
- **loads_on_supp (numerical)**: Count of elements both loaded and supported;
- **Psupp_edges (numerical)**: Percentage of supports located within the geometrical edges of the panel or in the frontier with geometric voids;
- **Psingle_loads (numerical)**: Percentage of single loaded nodes, *i.e.* percentage of loads surrounded by unloaded nodes, with relation to the total loads count;
- **Plinear_loads (numerical)**: Percentage of loads arranged in a linear (vertical, horizontal or diagonal) pattern.

The genesis of the last four features are due to technical reasons. With the count of elements both loaded and supported ("loads_on_supp") it is intended to assess whether the model was able to learn the principle that loads directly applied in a supported element do not cause bending moments. Supports located at the edge of the panel or near void contours tend to be critical points in the sense that the bending moments pattern generated within their neighbours is potentially different than those located internally. This success in learning this difference in behaviour is what "Psupp_edges" tries to assess. "Psingle_loads" and "Plinear_loads" are designed to appraise whether the models have problems in dealing with particular load patterns, *i.e.* isolated loads or load patterns with a "thickness" of one finite element. Examples of linear loading (Plinear_loads = 1.0) and single loading patterns (Psingle_loads = 1.0) are shown in Figure 5.40.

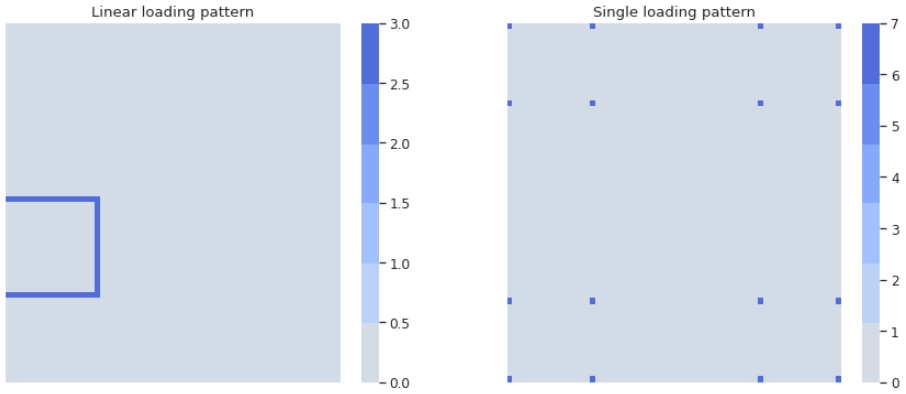


Figure 5.40: Examples of linear loading pattern and single loading pattern

Through the analysis of the "geom_group" feature, it can be seen that the so-called unsuccessful instances are spread over all of the geometry groups. Despite G06 and G09 standing out for the Linear dataset case, and G09 standing out for the the Non Linear dataset case, it cannot be stated that the errors are specific to these groups. Figure 5.41 shows a stacked bar plot with the count of successful and unsuccessful instances across both models and datasets. In relative terms, Figure 5.42 summarizes the success rates for each model and dataset throughout each geometry group.

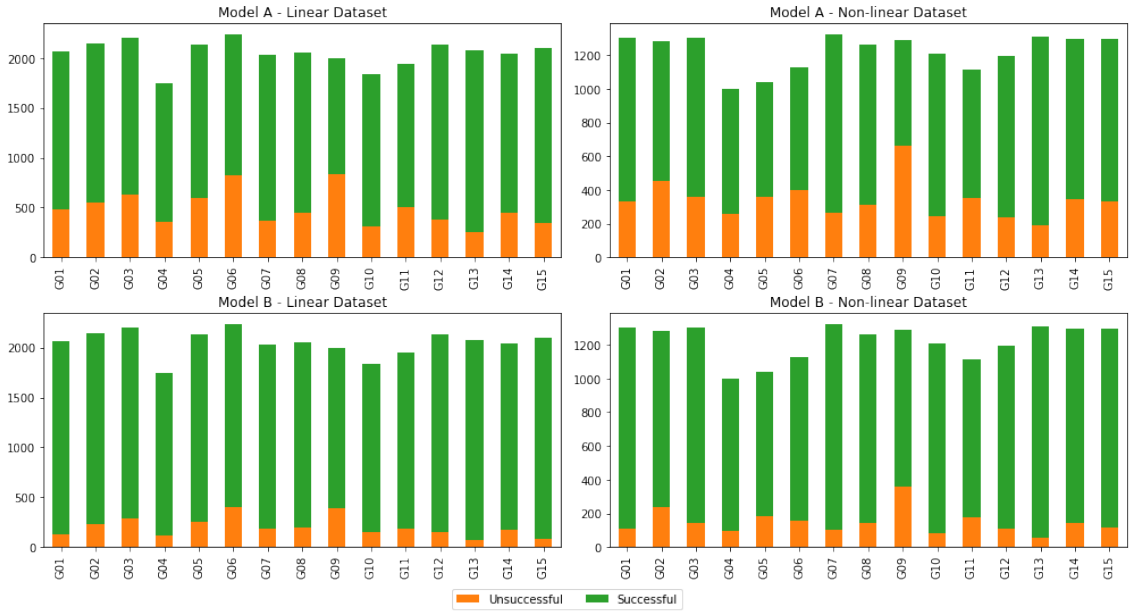


Figure 5.41: Count of successful and unsuccessful instances across the geometry groups

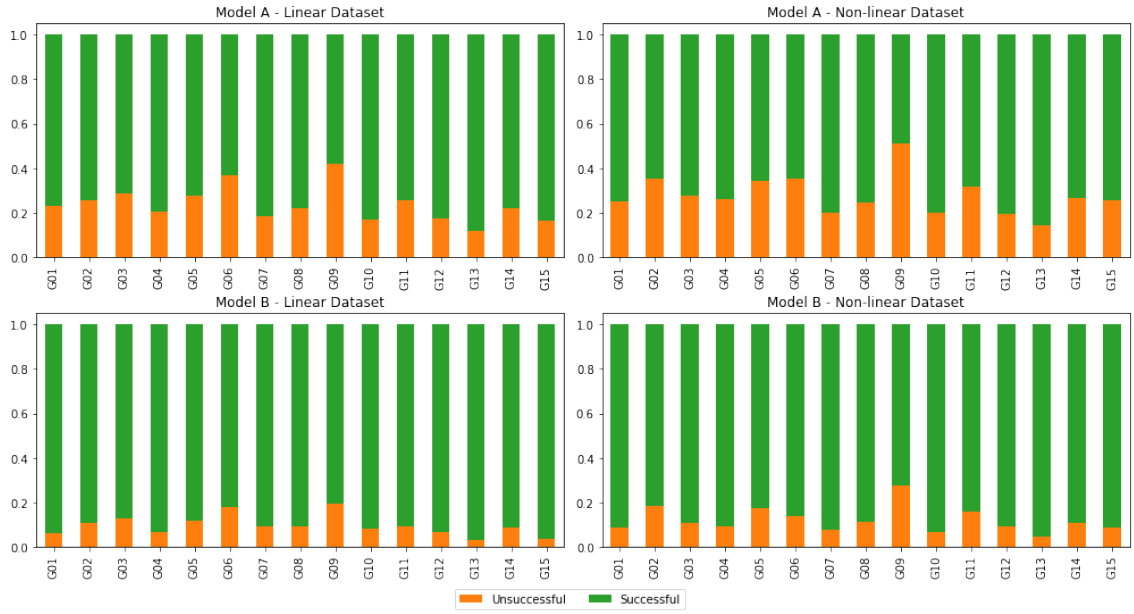


Figure 5.42: Rates of successful and unsuccessful instances across the geometry groups

Continuing the univariate analysis of these features, histograms of the remaining (numerical) features were analyzed. From the analysis of these, similarly to "geom_group" it can be concluded that the unsuccessful instances are spread over the range of each feature. In other words, they do not concentrate at any specific zone within the range of any of the features so that any conclusion can be drawn. Since the results of the various combinations of models and datasets are qualitatively equivalent, by virtue of simplicity, instead of presenting histograms for all the combinations of models and datasets, these are presented for one combination only. Figure 5.43 plots the histograms of both successful and unsuccessful instances for all numeric features for the combination of Model B and the Non Linear dataset.

Following the principle of simplicity from the last paragraph, Figure 5.44 plots a pair-wise distribution of the features in analysis for the combination of Model B and the Non Linear dataset. Subplots below the diagonal represent scatter plots between each of the feature while the subplots above the diagonal show KDE (kernel density estimate) plots. From this bivariate analysis no additional conclusion can be drawn, since once again, none of the pairs evidentiate a clear separation between successful and unsuccessful instances.

From the analysis of these results it is concluded that no simple rules can be drawn in order to control or limit the range of application of the models in analysis as a way of preventing erroneous results.

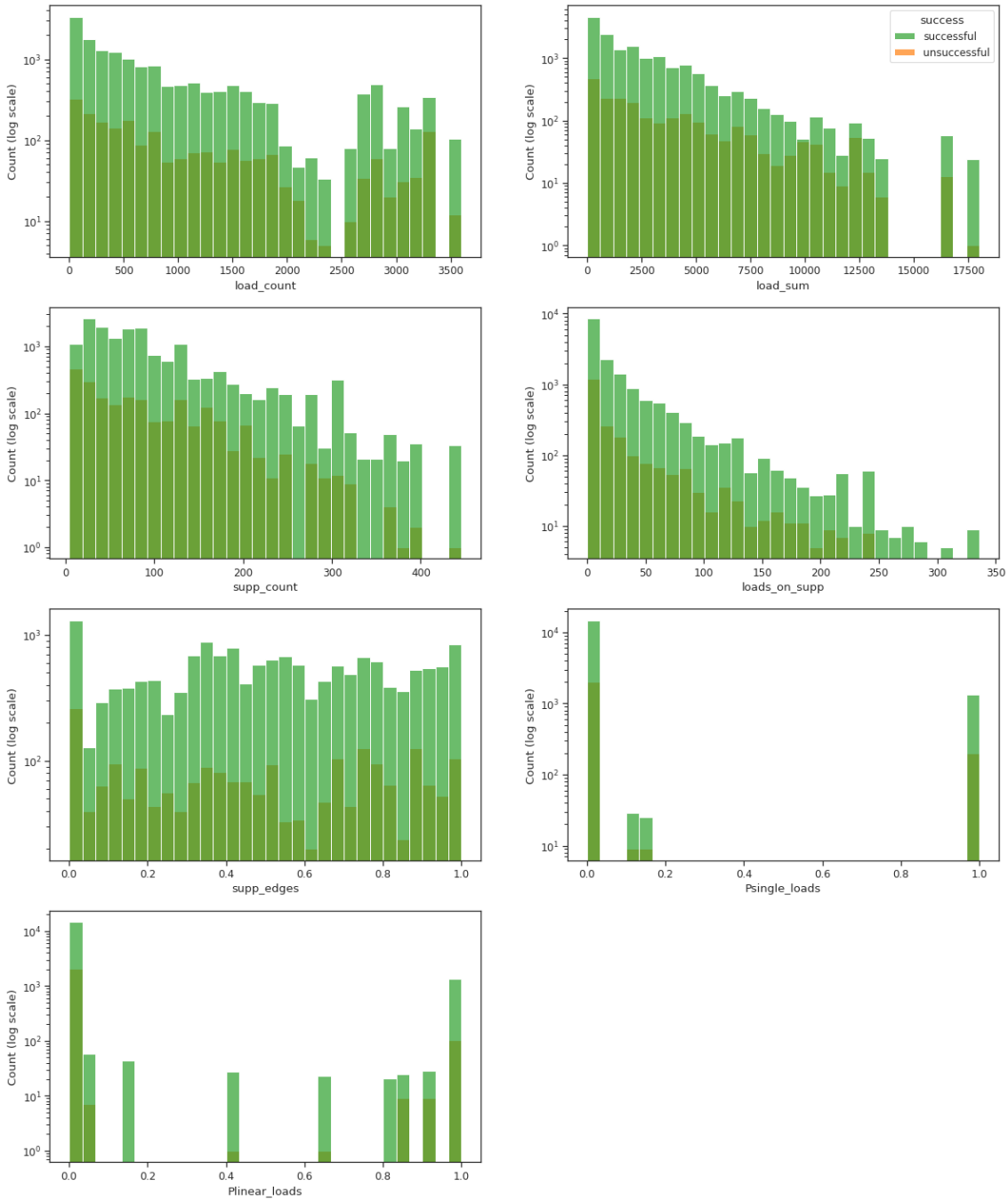


Figure 5.43: Overlapped histograms of the engineered numerical features of the Non-Linear dataset for Model B

Figure 5.44: Pair plot of the engineered numerical features of the Non-Linear dataset for Model B

Chapter 6

Conclusions and Future Work

The main objective of this dissertation is to present a faster but still reliable approach for calculating internal stresses in shell structures. Nowadays this task is commonly solved by using computational models based on the finite-element method. This method is an analytical method, widely used in engineering, not only for structural analysis but also for a wide range of applications within the mechanical engineering discipline (such as aeronautical, biomechanical, and automotive industries), which is typically time-consuming and demanding in terms of computational resources. This method has as its fundamental principle the discretization of a larger system into a mesh of smaller elements (hence the name "finite-elements"). This discretization principle, which is the basis of this work, is deeply linked to images in general, in the sense that they are a discretization of the spacial domain. Taking advantage of this resemblance, and since Convolutional Neural Networks (CNN) are widely known for their great potential for image processing, the main idea is to extend the use this subtype of Neural Networks to predict bending stresses on shell structures using FEM models as ground truth. To this end, simply put, a dataset of 18 000 models was built from the ground up and calculated following two calculation approaches with different levels of mathematical complexity to serve as basis for the learning procedure. Thenceforth, a pre-designed model from the literature, for solving a different problem also related with the finite-element method (Model A) was experimented in this particular problem, and a new model (Model B), as an upgrade of the previous, was designed and put into practise.

After a thorough analysis of the results, it becomes clear that both Deep Learning models are able to produce very good results in the vast majority of the test instances. For instance, Model B, the top performer, is able to achieve a success rate of roughly 90% on the Linear dataset and 88% on the Non-Linear dataset, which is as improvement of approximately 20% over Model A. Regarding processing times, the greater performance of Model B, compared to Model A, comes at a cost of 65% more computation time. Still, the difference in processing times between the DL models and the FEM models is appreciable. Even taking into account that each DL model needs to be ran twice to get a full representation of bending moments of an instance, *i.e.* to predict

bending stresses along both X and Y directions, Model B showed to be about 4.4 times faster than the Linear FEM calculations, and about 405 times faster than the Non-Linear FEM calculations, considering mean times.

On the other hand, it is also clear that, despite not many, there are some examples in which the models do not perform well. The fact that these examples cannot be conveniently circumscribed in terms of their input features, in other other words, cannot be easily distinguished among the "population", makes it unfeasible to use these models, at their current state, even in basic structural analysis tasks, simply because one would not know what to expect in terms of errors. However, the evidence that the majority of the instances were able to attain neglectable errors should not be disregarded, as it leaves the door open for future improvements.

As an attempt to improve generalization, the dataset in analysis was object of augmentation according to some usual data augmentation techniques in computer vision, such as rotation and flipping. Even so, the increase on the ability of generalization obtained from data augmentation can only go so far, once it derives from the original dataset. Since the features extracted by the models are unknown, it is unclear that the original dataset is enough to provide the model with sufficient diversity or how much it would benefit from a larger and more diversified dataset.

As previously stated, Non-Linear FEM models, are solved by means of a iterative calculation routine. These routines typically start from an initial state, perform the calculations and use the result as initial state for the iteration to come, repetitively, until its error falls within a given tolerance. Having this in mind, it becomes evident that the more accurate is the initial state of the problem, the faster will be its calculation process. Thereby, having a mechanism based on the present approach, capable of instantly providing an educated guess for the initial state could be highly rewarding in terms of number of iterations and consequently in the calculation time.

As concluded in Chapter 5, Model B, an upgrade attempt over Model A, actually achieves better results than Model A. Therefore, as a suggestion for future works, it would be important to understand whether more complex CNN architectures could still improve on the results obtained in the present work within low computation time. Other architectures may also be suitable for this particular problem, such as GANs (Generative Adversarial Networks). Another suggestion for a potential future work would be to understand whether Transfer Learning (the technique of reusing model layers from a network designed to solve a specific problem, in other problem) [6] from classic computer vision models would produce good results in this particular problem. Models such as VGG, Inception and ResNet were trained on real images rather than finite-element models, although it is unclear whether the present problem could benefit from low-level features extracted by these pre-trained models.

References

[1] Charu C Aggarwal et al. Neural networks and deep learning. *Springer*, 10:978–3, 2018.

[2] Hamed Habibi Aghdam and Elnaz Jahani Heravi. *Guide to Convolutional Neural Networks*. Springer International Publishing, 2017.

[3] Md Zahangir Alom, Tarek M Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Mahmudul Hasan, Brian C Van Essen, Abdul AS Awwal, and Vijayan K Asari. A state-of-the-art survey on deep learning theory and architectures. *Electronics*, 8(3):292, 2019.

[4] Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J. Santamaría, Mohammed A. Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data*, 8, 2021.

[5] Jose Luis Blanco, Steffen Fuchs, Matthew Parsons, and Maria João Ribeirinho. Artificial intelligence: Construction technology's next frontier. https://www.mckinsey.com/business-functions/operations/our-insights/artificial-intelligence-construction-technologys-next-frontier, 2018. Accessed: 2022-02-25.

[6] Jason Brownlee. A gentle introduction to transfer learning for deep learning. https://machinelearningmastery.com/transfer-learning-for-deep-learning/, 2019. Accessed: 2022-08-24.

[7] Jason Brownlee. How to choose an activation function for deep learning. https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/, 2021. Accessed: 2022-05-24.

[8] Jason Brownlee. Weight initialization for deep learning neural networks. https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/, 2021. Accessed: 2022-06-01.

[9] Huy Bui. From convolutional neural network to variational auto encoder. https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/, 2020. Accessed: 2022-05-21.

[10] Bindi Chen. 7 popular activation functions you should know in deep learning and how to use them with keras and tensorflow 2. https://towardsdatascience.com/7-popular-activation-functions-you-should-know-in-deep-learning-and-how-to-use-them-with-keras-and-27b4d838dfe6, 2021. Accessed: 2022-05-24.

[11] Philippe G Ciarlet and Jacques-Louis Lions. *Handbook of Numerical Analysis: VOL II: Finite Element Methods.(Part 1)*. North-Holland, 1991.

[12] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[13] Ray W Clough. The finite element method in plane stress analysis. In *Proceedings of 2nd ASCE Conference on Electronic Computation, Pittsburgh Pa., Sept. 8 and 9, 1960*, 1960.

[14] Richard Courant. Variational methods for the solution of problems of equilibrium and vibrations. *Bulletin of the American Mathematical Society*, 49:1–23, 1943.

[15] Siddharth Das. Cnn architectures: Lenet, alexnet, vgg, googlenet, resnet and more.... https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5, 2017. Accessed: 2022-02-19.

[16] Rachel Draelos. The history of convolutional neural networks – glass box. https://glassboxmedicine.com/2019/04/13/a-short-history-of-convolutional-neural-networks/, 2019. Accessed: 2022-02-17.

[17] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

[18] Kheirie Elhariri. Convolutional neural networks for image classification. https://medium.com/analytics-vidhya/convolutional-neural-networks-for-image-classification-c403159e81af, 2021. Accessed: 2022-05-21.

[19] elunic AG. Mit ai.see™ die vorteile von deep learning in der qualitätskontrolle nutzen. https://www.elunic.com/de/mit-ai-see-die-vorteile-von-deep-learning-in-der-qualitaetskontrolle-nutzen/, 2021. Accessed: 2022-02-28.

[20] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[21] Shang-Hua Gao, Ming-Ming Cheng, Kai Zhao, Xin-Yu Zhang, Ming-Hsuan Yang, and Philip Torr. Res2net: A new multi-scale backbone architecture. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(2):652–662, 2021.

[22] A. Géron. *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017.

[23] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. Accessed: 2022-02-19.

[25] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. Ghostnet: More features from cheap operations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1580–1589, 2020.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-Decem:770–778, 2016.

[28] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[29] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[30] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[31] Yang Hu, Guihua Wen, Mingnan Luo, Dan Dai, Jiajiong Ma, and Zhiwen Yu. Competitive inner-imaging squeeze and excitation for residual network. *arXiv preprint arXiv:1807.08920*, 2018.

[32] David H. Hubel. Single unit activity in striate cortex of unrestrained cats. *The Journal of Physiology*, 147:226–238, 1959.

[33] David H. Hubel and Torsten N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148:574–591, 1959.

[34] Haoliang Jiang, Zhenguo Nie, Roselyn Yeo, Amir Barati Farimani, and Levent Burak Kara. Stressgan: A generative deep learning model for two-dimensional stress distribution prediction. *Journal of Applied Mechanics, Transactions ASME*, 88, 5 2021.

[35] Franklin Johnson, Alvaro Valderrama, Carlos Valle, Broderick Crawford, Ricardo Soto, and Ricardo Ñanculef. Automating configuration of convolutional neural network hyperparameters using genetic algorithm. *IEEE Access*, 8:156139–156152, 2020.

[36] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, 53:5455–5516, 2020.

[37] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, pages 1097–1105, 2012.

[39] Suki Lau. Learning rate schedules and adaptive learning rate methods for deep learning. https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1, 2017. Accessed: 2022-05-24.

[40] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[41] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278–2323, 1998.

[42] Liang Liang, Minliang Liu, Caitlin Martin, and Wei Sun. A deep learning approach to estimate stress distribution: a fast and accurate surrogate of finite-element analysis. *Journal of The Royal Society Interface*, 15(138):20170844, 2018.

[43] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[44] Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate. *arXiv preprint arXiv:1902.09843*, 2019.

[45] Warren Mcculloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.

[46] Marvin L. Minsky and Seymour A. Papert. *Perceptrons*. MIT Press, Cambridge, 1969.

[47] Will Nash, Tom Drummond, and Nick Birbilis. A review of deep learning in the study of materials degradation. *npj Materials Degradation*, 2(1):1–12, 2018.

[48] Zhenguo Nie, Haoliang Jiang, and Levent Burak Kara. Stress field prediction in cantilevered structures using convolutional neural networks. *Journal of Computing and Information Science in Engineering*, 20, 8 2020.

[49] Artem Oppermann. What is deep learning and how does it work? https://towardsdatascience.com/what-is-deep-learning-and-how-does-it-work-2ce44bb692ac, 2019. Accessed: 2022-02-28.

[50] Coding Prof. How to calculate the weighted mean absolute percentage error in r. https://www.codingprof.com/how-to-calculate-the-weighted-mean-absolute-percentage-error-in-r/, 2022. Accessed: 2022-07-17.

[51] Paul-Louis Pröve. Squeeze-and-excitation networks. https://towardsdatascience.com/squeeze-and-excitation-networks-9ef5e71eacd7, 2017. Accessed: 2022-02-23.

[52] Talha Quddoos. Neural network basics: Loss and cost functions. https://medium.com/artificialis/neural-network-basics-loss-and-cost-functions-9d089e9de5f8, 2021. Accessed: 2022-05-24.

[53] Bastian E. Rapp. Chapter 32 - finite element method. In Bastian E. Rapp, editor, *Microfluidics: Modelling, Mechanics and Mathematics*, Micro and Nano Technologies, pages 655–678. Elsevier, Oxford, 2017.

[54] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[55] Abhijit Guha Roy, Nassir Navab, and Christian Wachinger. Recalibrating fully convolutional networks with spatial and channel "squeeze and excitation" blocks. *IEEE transactions on medical imaging*, 38(2):540–549, 2018.

[56] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[57] Rathna S. Convolution neural networks (cnn). https://medium.com/@rathna211994/convolution-neural-networks-cnn-b6fe90214b1e, 2019. Accessed: 2022-05-21.

[58] Karl Heinrich Schellbach. Probleme der variationsrechnung. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, pages 293 – 363, 1851.

[59] Sagar Sharma. What the hell is perceptron? https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53, 2017. Accessed: 2022-03-02.

[60] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–14, 2015.

[61] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[62] Charles C. Tappert. Who is the father of deep learning? In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 343–348, 2019.

[63] Huu-Tai Thai. Machine learning for structural engineering: A state-of-the-art review. *Structures*, 38:448–491, 2022.

[64] M. J. Turner, R. W. Clough, H. C. Martin, and L. J. Topp. Stiffness and deflection analysis of complex structures. *Journal of the Aeronautical Sciences*, 23(9):805–823, 1956.

[65] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[66] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:5987–5995, 2017.

[67] Yinghan Xu. Faster r-cnn (object detection) implemented by keras for custom data from google's open images dataset v4. https://towardsdatascience.com/faster-r-cnn-object-detection-implemented-by-keras-for-custom-data-from-googles-open-images-125f62b9141a, 2018. Accessed: 2022-02-20.

[68] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[69] Łukasz Skotny. The difference between linear and nonlinear fea? https://enterfea.com/difference-between-linear-and-nonlinear-fea/, 2022. Accessed: 2022-06-22.