Physics-Driven Machine Learning for Real-Time Simulations of Medical Devices

Mestriner, R.J.

Award date:
2022

Physics-Driven Machine Learning for Real-Time Simulations of Medical Devices

Roelof Joachim Mestriner

Master's Thesis
Eindhoven University of Technology
Faculty of Mechanical Engineering
Mechanics of Materials Group
Eindhoven

Submitted on April 13, 2022

Thesis supervisor:	prof.dr.ir. O. (Olaf) van der Sluis (TU/e & Philips Research)
Second supervisors:	dr.ir. J.J.C. (Joris) Remmers (TU/e)
	dr.ir. M. (Marco) Baragona (Philips Research)
	dr.ir. P.H.M. (Peter) Timmermans (Philips Research)
	dr. B.J. (Bart) Bakker (Philips Research)

Abstract

Driven by a desire for real-time simulations of medical devices such as catheters and guidewires, an investigation is done towards a machine learning approach for real-time simulations of deformations of solids. A convolution neural network model, based on the U-Mesh framework, is developed as surrogate for finite element modelling. This model is evaluated for simple elastic deformation problems of small deformations in a cantilever beam and shows great potential for reductions in computational cost towards real-time simulations, while maintaining a sufficient accuracy. As addition to this model, a hybrid modeling approach is introduced in the form of physics-driven regularization. A regularizer in the form of the analytical solution to the problem is implemented in the existing model and shows major improvements in accuracy, computational cost and data dependency. Additional tests were conducted towards more generally applicable regularization terms, but show that more research is necessary for a better understanding.

Contents

Nomenclature

Symbol	Quantity	Unit
α	Sensitivity	-
$\bar{e}$	Mean absolute error	m
$\beta_{1,2}$	Hyperparameters of Adam optimizer	-
Γ	3-D Euclidean space	m^3
Γ_{CB}	3-D domain of the cantilever beam	m^3
γ_{top}	Discrete domain of nodes on the top surface	-
γ_{wall}	Discrete domain of nodes that are clamped in the wall	-
$\hat{y}_i$	Component of prediction	-
$\langle \dots \rangle$	Macaulay brackets	-
$\underline{\hat{\mathbf{U}}}$	Predicted deflection, tensor form	m
$\underline{\hat{\mathbf{y}}}$	Prediction, tensor form	-
$\underline{\mathbf{F}}$	Traction forces, tensor form	N
$\underline{\mathbf{U}}$	Deflection, tensor form	m
$\underline{\mathbf{y}}$	Ground truth, tensor form	-
$\hat{u}_i$	Component of predicted deflection	m
u_i	Component of deflection	m
Ω	Grid domain	-
$\omega_{x,y,z}$	Grid resolution in x, y, z-direction	-
ϕ	Activation function	
$\sigma(e)$	Standard deviation	-
θ_A	Slope at the tip	$^\circ$
$\theta_{A,i}$	Slope at the tip for a certain point load	$^\circ$
a	Distance from wall to point load	m
b_j	Bias of a neuron	-
bs	Batch size	-
c	Number of channels in first layer	-
$C_{1,2,3,4,5,6}$	Integration constants	-
d	Derivation operator	-
$d_{x,y,z}$	Mesh size in x, y, z-direction	-
E	Young's Modulus	Pa

F	Force	N
f	Number of filters	-
F_i	Point load	N
$f_{x,y,z}$	Filter size in x, y, z-directions	-
h	Height of the cantilever beam	m
h_j	Activation of a neuron	-
I	Second moment of area	m^4
i, j	Indexation	-
k	Network depth in steps	-
l	Length of the cantilever beam	m
lr	Learning rate	-
M	Bending moment	Nm
m	Number of point loads	-
$M_{A,i}$	Bending moment at the tip of the beam for a certain point load	Nm
M_A	Bending moment at the tip of the beam	Nm
N	Number of samples	-
n	Nodal points	-
$n_{x,y,z}$	Nodal points in x, y, z-direction	-
Q	Shear force	N
R	Curvature	m^{-1}
r	Residual of beam equation	N
u	Deflection	m
u_A	Deflection at the tip	m
$u_{A,i}$	Deflection at the tip for a certain point load	m
u_{fit}	Fitted form of the predicted deflection	m
w	Width of the cantilever beam	m
$w_{j,d}$	Weighted connection between two neurons	-
x, y, x	Cartesian coordinate directions	-
x_d	Input of a neuron	-
y_i	Component of ground truth	-

Abbreviations

ANN Artificial Neural Networks. 10

BC Boundary Conditions. 6, 37

BVP Boundary Value Problem. 9

CB Cantilever Beam. 3, 8

CNN Convolutional Neural Network. 12, 20

DNN Deep Neural Network. 1, 2, 10, 11

DOF Degrees of Freedom. 9

EB Euler-Bernoulli beam equation. 6

FE Finite Element. 1

FEA Finite Element Analysis. 15

FEM Finite Element Method. 1–3, 9, 12, 18–21, 28–30, 35, 38, 40, 42–44, 51–53

GCB Guided Cantilever Beam. 8, 38–40, 57

GPU Graphics Processing Unit. 1, 18

MAE Mean Absolute Error. 13, 21, 27, 29, 34, 35, 38–40, 57

MAMPE Mean Absolute Maximum Percentage Error. 22, 24, 25, 34, 37, 39

MAPE Mean Absolute Percentage Error. 22

ML Machine Learning. 1

MLP Multilayer Perceptron. 10–12

NNs Neural Networks. 10

PDR Physics-Driven Regularization. 14, 31, 34–38, 40, 41, 43, 54, 57

PINN Physics Informed Neural Network. 2, 31, 43

POD Proper Orthogonal Decomposition. 1

PReLU Parametric ReLU. 12

ReLU Rectified Linear Unit. 11, 21

RGB Red Green Blue. 12

RMS Root Mean Square Error. 13

ROM Reduced Order Modelling. 1

SGD Stochastic Gradient Descent. 11, 13

1. Introduction

1.1 Background

There are several engineering applications where the deformation of non-linear systems must be simulated in real-time. Some important examples are found in the field of medicine, such as guidance of medical devices, such as imaging catheters and guidewires, which can bring significant improvements to clinical practice (van Houten, 2018; UPSIM, 2022; van der Sluis, 2020).

The Finite Element Method (FEM) is a widely used method for simulating deformations of non-linear structures, known for its physics based accuracy, speed, robustness and extensive development. FEM simulations are currently used during the engineering design cycle to provide insights in among other things manufacturing, use of materials, design flaws, lifetime analysis, thermo-mechanical and micro-mechanical behaviour. Especially simulations in the medical field tend to be highly complex due to non-linear effects that are introduced by for example non-linear materials, history dependency, custom geometries, dynamic boundary conditions, large strains and contact algorithms, which lead to large computational costs (Miller et al., 2006; Kouznetsova, 2020).

There have been many previous attempts at reducing the computational cost of Finite Element (FE) simulations to achieve real-time simulations. Many works have focused on parallel computing techniques, for example by utilizing the ample amount of cores in a Graphics Processing Unit (GPU) (Comas et al., 2008). Other works have focused on speeding up simulation times by lowering the computational complexity of the problem through a reduction of the model's degrees of freedom by Reduced Order Modelling (ROM). One of the most common methods for model order reduction is Proper Orthogonal Decomposition (POD) (Niroomandi et al., 2008, 2017). POD techniques calculate the solution to multiple complete models off-line and extract the modes that best represent the complete problem's solution. POD seems to perform with very good accuracy at real-time applications (Goury and Duriez, 2018), but in some cases is not able to capture high degrees of non linearity correctly (Bhattacharjee and Matouš, 2016) .

Data-driven methods, or Machine Learning (ML), has recently begun to revolutionize many fields including computational mechanics (Frank et al., 2020). Given enough ground truth data, machine learning algorithms can map a function's input to its output without requiring any physical formulation, effectively acting as a black box that is generally very fast compared to conventional methods. Since FEM can provide as much ground truth data as required, it seems interesting to train machine learning algorithms with such virtually generated, or synthetic, data to speed up the simulations up to real-time. There are several methods to apply ML, but a Deep Neural Network (DNN) tend to be the most accurate and preferable for computational mechanics, since it is able to extract high-level representations of complex processes (Badarinath et al., 2021; Phellan et al., 2021).

A promising example of a DNN approach for real-time simulations of non-linear deformations is the U-mesh framework developed by Mendizabal (2020), which is based on the U-net architecture (Ronneberger et al., 2015). This work shows great potential as surrogate for FEM and presents great speed-ups.

Hybrid models proof to be an effective middle ground where some knowledge, typically in the form of physical laws, is combined with data-driven models, such as neural networks, to perform simulations at lower computational costs and higher convergence rates than regular physics-based or data-driven methods (Sluis et al., 2018; van der Sluis, 2020). Hybrid models show potential for being less dependent on data. Moreover, hybrid models are generally more robust and better explainable than regular models. Finally, hybrid models potentially converge faster; meaning that the computational costs are lower and the final accuracy is better. A great example of such a hybrid model is the Physics Informed Neural Network (PINN) introduced by Raissi et al. (2019). This method is able to solve nonlinear physics problems in an efficient manner by expanding a simple DNN through auto-differentiation, constraining the network to only make predictions that are conform to a desired physical law. Other methods, such as Physics-Driven Regularization (PDR) (Nabian and Meidani, 2020), implement a less constraining and more steering approach towards hybrid modelling. In this method, a regularizer is used in order to discourage solutions that are physically not feasible, which is a simple addition that is generally applicable to most networks that describe physical phenomena.

1.2 Research objective

This research aims to explore the potential of a neural network-based approach as surrogate for finite element simulations of deformations of elastic solids. The proposed method could possibly aid in real-time modelling of medical devices. The goal is not to develop a fully working model that can be used in a real application, but instead to perform a proof of concept for this modeling technique and explore potential additions in the form of hybrid modelling techniques. Simple problems will be used to yield insights on potential computational improvements and problems that may rise during the development of such a model.

1.3 Project organization

In this work, we propose to use the U-Mesh method of Mendizabal (2020) as a first step towards more advanced simulations of medical devices. This approach is chosen since it closely resembles the well established U-Net, and shows potential for a similar approach on FEM data. First it is tried to replicate their model. The model is then evaluated on a simple elastic deformation problem; the deflection of a cantilever beam. Computational experiments are performed on analytically and computationally generated data sets. Finally, an addition is made by implementing physics-driven regularization to the network. This proposed hybrid modelling approach may improve performance of the model by offering knowledge based on the analytical solution of the problem. Some extra experiments are performed to explore the limits of this method.

The fundamental methodologies that are needed to understand the proposed method are treated in Chapter 2. This includes the definition of the problem and two methods to solve it. The development of the model, including all considerations on design choices, is presented in Chapter 3; followed by both computational experiments to validate the performance of the model. The addition of physics-driven regularization is explored in Chapter 4. All findings are discussed in Chapter 5.

2. Methods

In this chapter, all used methodologies are explained to a point needed to understand the development of the model and to interpret the results that are achieved. First, a problem is defined that is used to evaluate the proposed method through computational experimentation. The derivation of the analytical solution to the problem is presented, which will be used later on for generating a data set. Then an introduction is given on computational mechanics and FEM, which are go-to modelling techniques for simulating deformations of solids and are used later on for generating the second data set. Finally a thorough introduction is given on the fundamentals of machine learning, which builds up to an explanation of the U-Mesh approach.

2.1 Problem formulation

The problem that is used to evaluate the proposed method is defined as the elastic deflection of a Cantilever Beam (CB) for small deformations. The solution to such a problem is fully known and does not induce complex non-linear behaviour, resulting in fast calculations; analytically as well as numerically. Medical devices, such as catheters and guidewires, could also be represented, as a first order assumption towards medical applications, with this beam-like formulation.

We consider the small deformation of a CB that is defined in a 3-dimensional Euclidean space, $\Gamma_{CB} \in \Gamma$; the geometry is denoted on a right handed Cartesian basis. The beam is made of an elastic isotropic material ($E = 500$ Pa), and is homogeneous. The beam has a square profile of size $h \times w$, and a length of l, as depicted schematically in Figure 2.1; where $h = 1$ m, $w = 1$ m and $l = 4$ m. The beam is subjected to a fixed boundary γ_{wall} on one end, and loaded with n random sampled point loads on the top side γ_{top} in y-direction. The desired solution of this problem is defined as the displacement (or deflection) in the y-direction over the length of the beam.

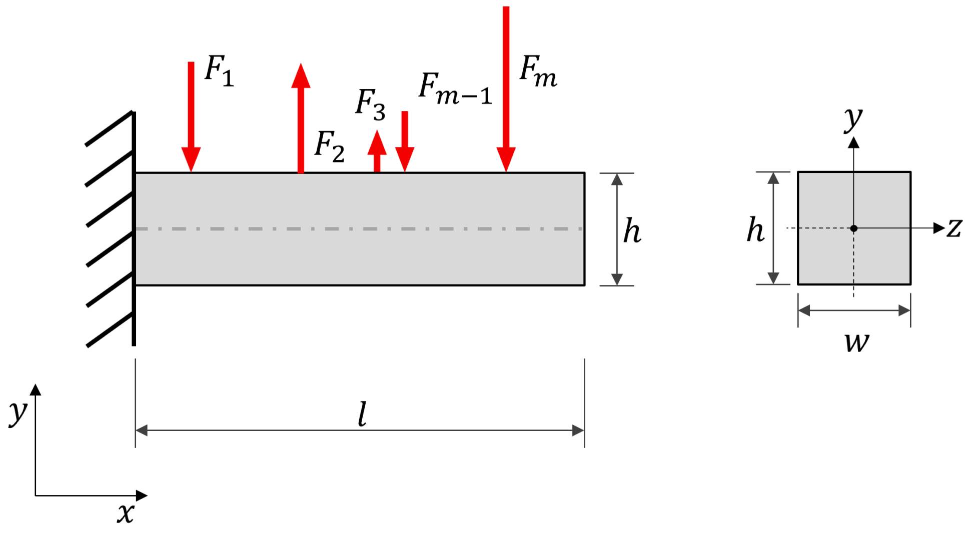


Figure 2.1: Schematic representation of a point-loaded cantilever beam

The analytical solution to this problem is defined as a continuous 1-D representation of the deflection in the y direction along the centroid of the beam ($y = z = 0$). The numerical solution consists of 3-D deflection vectors that are defined on discrete nodal points in the spatial domain of the beam. Both solutions are explained in sections 2.1.1 and 2.2.1 respectively.

To compare the solutions at certain coordinates along the length of the beam, it was chosen to define uniformly distributed nodal points, n, at which both methods are evaluated.

For the analytical approach, the beam is discretized along the length of the beam with uniform intervals of $dx = \frac{l}{n-1} = \frac{4}{256-1} = 0.016$, defining $n = 256$ nodal points that are located in the centroid of the beam. Each point load, F_i, is applied at a node $n_i \in \{2, \ldots, n\}$ and has an amplitude $-1 < F_j < 1$ [N]. The forces are not applied at node $n = 1$ since that node is clamped in the wall and will not result in any deformations.

For the numerical approach, the beam is discretized along all three spatial dimensions. In order to have the same amount of nodes for both methods, it was decided to discretize the domain with uniform intervals of $dx = \frac{l}{n-1} = \frac{4}{16-1} = 0.27$, $dy = \frac{w}{n-1} = \frac{1}{4-1} = 0.33$ and $dz = \frac{h}{n-1} = \frac{1}{4-1} = 0.33$, ensuring a $16 \times 4 \times 4$ distribution of nodal points. The point loads are applied on the free nodes at the top surface, $N \in \gamma_{top}$. Both discretized domains are presented visually in Figure 2.2 and will be used later as input structure for the model.

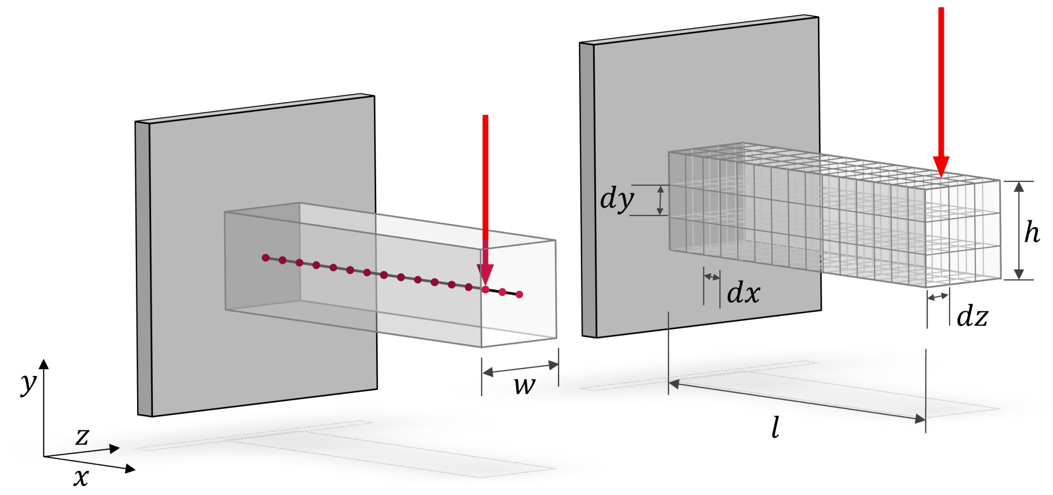


Figure 2.2: Schematic representation of the domain discretization in 1-D and 3-D

The variations in the input conditions for the analytical and numerical solution methods will also result in varying results. For example, the nodal distribution of the numerical method will bring about torsional components, while the analytical solution does not take torsion into consideration. Even though both solutions come from the same initial problem, they are considered as different problems throughout the rest of this thesis.

2.1.1 Analytical solution to the problem

There are several ways to derive the analytical solution to this problem. A common approach is to use the method of superposition on multiple common statically determinate beams (Dutch: *vergeetmenietjes*)(Fenner and Reddy, 2012). We use this method to generate the data set that is used in section 3.2. A step-by-step explanation of this approach is presented below. The following assumptions are made in order to justify the use of certain formula's:

1. All strains and rotations are small
2. The beam is slender and homogeneous
3. The profile of the beam is constant over the length of the beam

4. The material behaves elastically and isotropic
5. There is no shear deformation

From linearity it follows that the deflection of a single beam with multiple point-loads, as depicted in the left of Figure 2.3, can be expressed as the sum of the deflections of multiple single point-loaded beams. Hence, the beam presented in the right of Figure 2.3 is considered for the rest of the derivation. All indexes i concerning the superposition are not included in the derivation for a more clear notation.

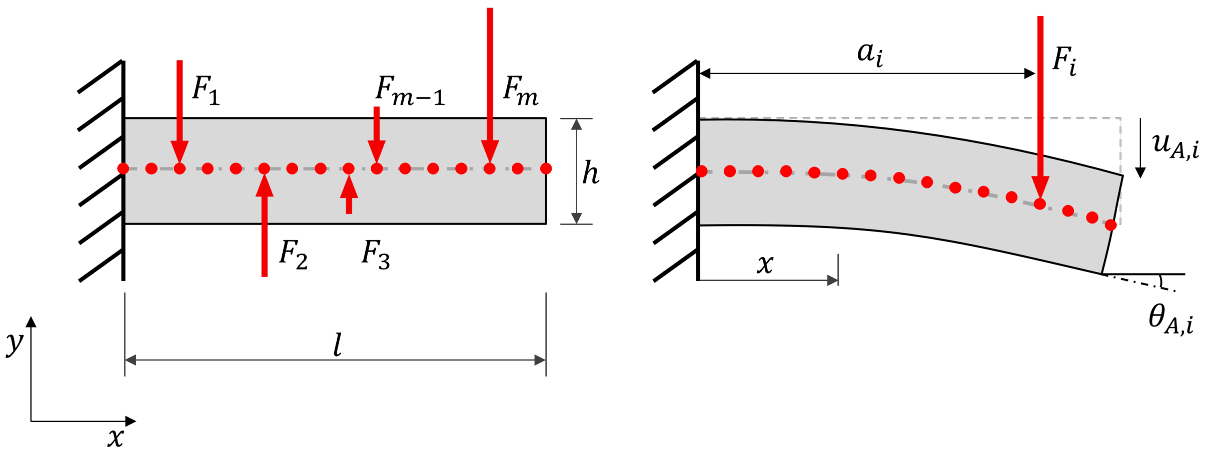


Figure 2.3: Schematic representation of the analytical cantilever beam problem. Left: cantilever beam loaded with n random point loads. Right: Deflection component of the cantilever beam subjected to a single point load F_i.

The beam is subjected to a single point-load at distance a from the clamp, dividing the beam into two sections along its length: $x < a$ and $x > a$. A force is defined to be positive if it points upward, along the positive y direction; meaning that the depicted point load F_i is negative. This point load creates an internal bending moment M that is defined to be positive if the top of the beam where to be compressed; meaning that in this case it is negative for all $x < a$. Since this beam is statically determinate, we can calculate the bending moment M using the equilibrium of forces and moments

$$\sum F = 0 \qquad\qquad \sum M = 0 \tag{2.1.1}$$

The bending moment can be expressed as

$$M(x) = F\langle a - x\rangle \tag{2.1.2}$$

where $\langle \dots \rangle$ denote Macaulay brackets, defined as

$$\langle a - x\rangle = \begin{cases} a - x & \text{if } x < a \\ 0 & \text{if } x > a \end{cases} \tag{2.1.3}$$

Using linear elasticity theory (Hooke's law) and infinitesimal strain theory, a relation between the bending moment, M, and the curvature, R, is established; known as the general form of the moment-curvature relationship (Fenner and Reddy, 2012; Geers and Schreurs, 2015)

$$M(x) = \frac{EI}{R} = EI\frac{d^2u}{dx^2} \tag{2.1.4}$$

Where u is the deflection, E is the Young's modulus and I the second moment of area of the cross section of the beam.

$$I = \iint y^2 dydz \tag{2.1.5}$$

Differentiating Equation 2.1.4 once yields the shear force, Q, and differentiating it twice yields the force F. The latter equation is also known as the Euler-Bernoulli beam equation (EB) and it describes the relation between the applied force and deflection of the beam.

$$Q(x) = EI\frac{d^3u}{dx^3} \qquad F(x) = EI\frac{d^4u}{dx^4} \tag{2.1.6}$$

Solving the EB beam equation would lead to the solution of this problem. But since the expression for the bending moment is already known, it is far more logical to solve Equation 2.1.4. This can be done in multiple ways. A common method is Macaulay's method (or double integration method). Combining the expressions for the bending moment in Equations 2.1.2 and 2.1.4 yields

$$EI\frac{d^2u}{dx^2} = F\langle a - x\rangle \tag{2.1.7}$$

or

$$EI\frac{d^2u}{dx^2} = \begin{cases} -Fx + Fa & \text{if } x < a \\ 0 & \text{if } x > a \end{cases} \tag{2.1.8}$$

Repeated integration gives the general solutions for slope, θ, and deflection, u, respectively as

$$EI\frac{du}{dx} = \begin{cases} -\frac{F}{2}x^2 + Fax + C_1 & \text{if } x < a \\ C_3 & \text{if } x > a \end{cases} \tag{2.1.9}$$

$$EIu(x) = \begin{cases} -\frac{F}{6}x^3 + \frac{Fa}{2}x^2 + C_1x + C_2 & \text{if } x < a \\ C_3x + C_4 & \text{if } x > a \end{cases} \tag{2.1.10}$$

Integrating does however yield integration constants that are resolved by introducing the Boundary Conditions (BC). The BC for this problem are that both the deflection and slope of the beam are zero at the build-in end of the cantilever, at $x = 0$. It is also known that both terms of each equation should be equal to each other at $x = a$. Substituting these conditions into Equations 2.1.9 and 2.1.11 yields $C_1 = C_2 = 0$, $C_3 = \frac{F}{2}a^2$ and $C_4 = -\frac{F}{6}a^3$. Substituting these terms in Equation 2.1.11 yields the solution for the deflection of a cantilever beam subjected with a single point load.

$$u(x) = \begin{cases} \frac{F}{6EI}(3ax^2 - x^3) & \text{if } x < a \\ \frac{F}{6EI}(3a^2x - a^3) & \text{if } x > a \end{cases} \tag{2.1.11}$$

Finally, a formulation for the cumulative deflection of m point loads is derived in Young and Budynas (2002)

$$u(x) = \sum_{i=1}^{m} \left(u_{A,i} - \theta_{A,i}(l - x) + \frac{F_i}{6EI} \langle a_i - x \rangle^3 \right) \tag{2.1.12}$$

where θ_A, i and u_A, i represent the slope and the deflection at the tip of the beam for each point load respectively

$$u_{A,i} = \frac{F_i}{6EI}\left(3la_i^2 - a_i^3\right) \qquad \theta_{A,i} = \frac{F_i a_i^2}{2EI} \tag{2.1.13}$$

Figure 2.4 shows a diagram of all resulting physical quantities for a cantilever beam loaded with a single point load of -1 N at $a = 3.2$ m. The shear force is not continuously differentiable along the length of the beam, which agrees with the discrete formulation of a point load.

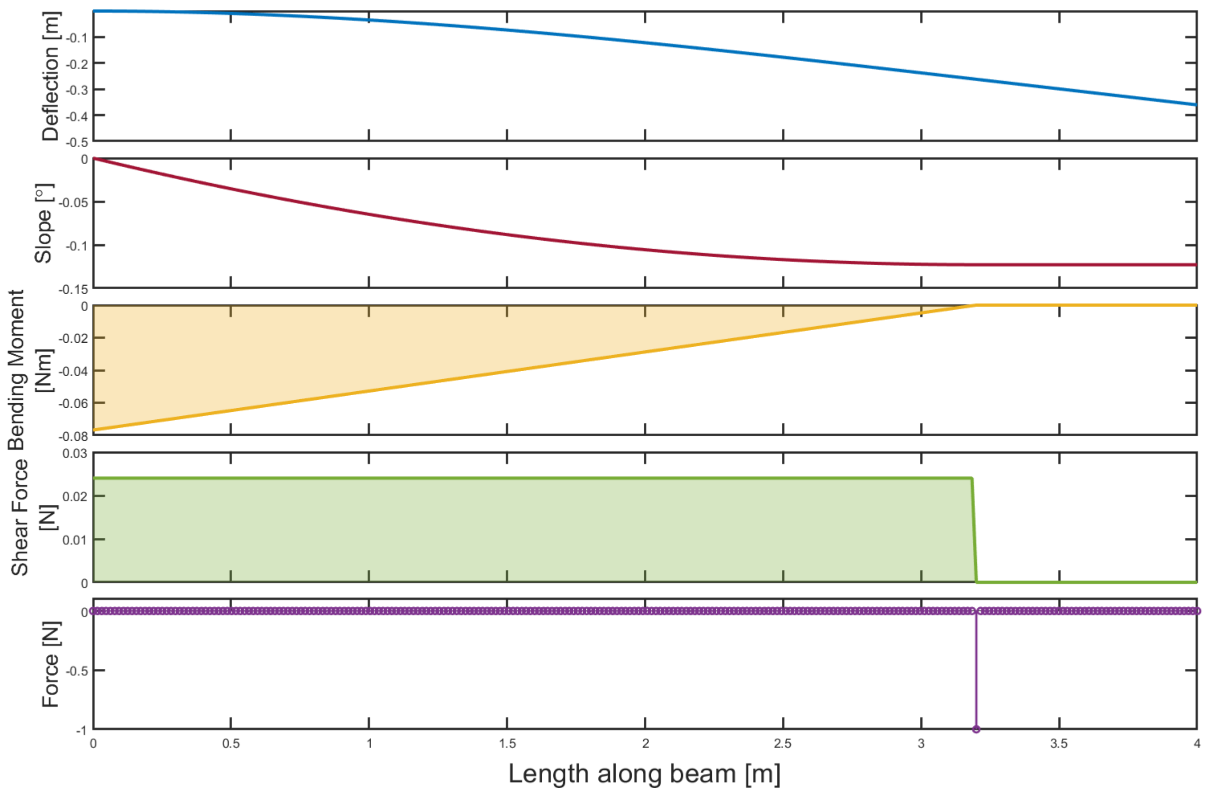


Figure 2.4: Resulting physical quantities for a free-end cantilever beam that is subjected to a single point load with amplitude -1 N at $a = 3.2$ m.

2.1.2 Guided cantilever beam

The boundary value problem can be altered quite simply by introducing different loading or boundary conditions. An example is to introduce a guided support at the tip of the beam (Figure 2.5) that constrains the slope to be zero.

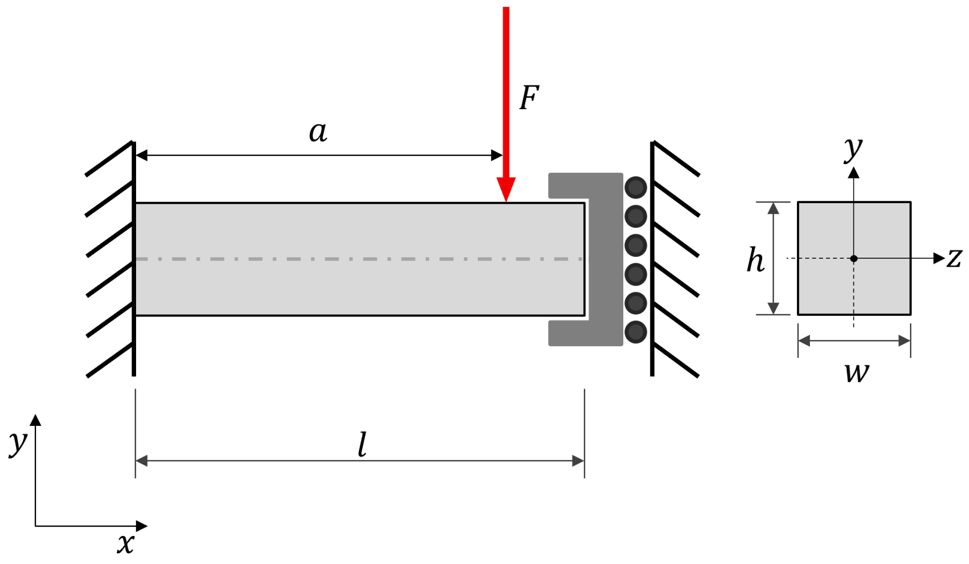


Figure 2.5: Schematic representation of a cantilever beam loaded with a single point load that has a guided support at the tip of the beam, that constrains the slope to be 0

The general solution to this problem is also derived in Young and Budynas (2002) and is formulated as

$$u(x) = \sum_{i=1}^{m} \left(u_{A,i} + \frac{M_{A,i}x^2}{2EI} + \frac{F_i}{6EI} \langle a_i - x \rangle^3 \right) \tag{2.1.14}$$

where

$$u_{A,i} = \frac{F_i}{12EI} (a_i)^2 (3l - 2a_i) \qquad M_{A,i} = \frac{F_i (a_i)^2}{2l} \tag{2.1.15}$$

where $M_{A,i}$ is the bending moment at the tip of the beam for each point load. The typical deflections for the free-end CB and Guided Cantilever Beam (GCB) are compared for the same loading conditions ($n = 1, F = -1, a = 3.2$m) in Figure 2.6. The general form of both deflections closely resemble each other, since both are defined by the same general polynomial. The resulting deflections for the same loading conditions differ nevertheless significantly.

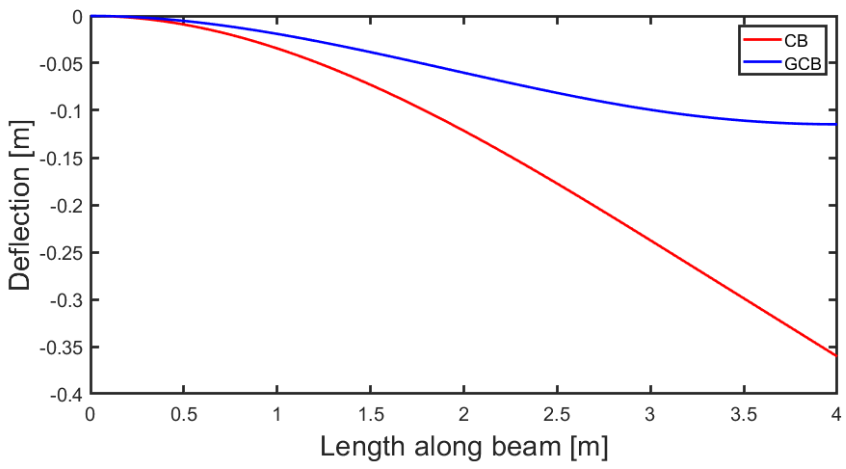


Figure 2.6: Typical deflections for a free-end and guided cantilever beam for a single point load ($n = 1, F = -1, a = 3.2$ m)

2.2 Computational mechanics

The vast majority of analysis in structural mechanics is based on the finite element method because of its accuracy and ability to simulate a large range of materials on complex domains. FEM works by finding a numerical approximation of a differential equation, or Boundary Value Problem (BVP). The BVP consists of a domain with Dirichlet and Neumann boundary conditions, some physical equilibrium equation, a constitutive law and a strain measure. The constitutive law presents a linkage between the strain and stress, and the strain measure couples the strain to displacements. The domain is then discretized (Figure 2.2) in nodes and elements through the use of shape functions and isoparametric mapping which results in a linear system of equations. The linear system of equations is then integrated numerically and solved (Kouznetsova, 2020).

Solving the system of equations results in a stress-strain (or force-displacement) measure for each node on the domain. The forces and displacements of each node are the desired end products of the simulation and will also be used as input data for the U-mesh architecture.

In order to perform a full FEM simulation however, several choices have to be made regarding the input conditions. Examples of these choices are the type of constitutive law to use, what strain measure to use, which element types to use, what the size of the mesh is and which solver type to use. More complex problems tend to need more a complex description of these conditions, which generally leads to significantly larger computation times. Fortunately, our problem definition is simple, and has relatively straightforward input conditions.

2.2.1 Numerical solution to the problem

It was chosen to perform the numerical simulations in `ABAQUS` (Duval et al., 2014), since it offers a simple and relatively fast scripting input that is based around `Python`(Van Rossum and Drake Jr, 1995), which makes it easy to generate large datasets with varying input criteria (Dassault Systèmes, 2011). The mesh consists of 256 nodes in a $(16, 4, 4)$ arrangement, which make up 135 `C3D8I` continuum elements, each defined with 8 nodes of which each have 3 translational Degrees of Freedom (DOF). All 16 nodes at γ_{wall} have an `ENCASTRE` boundary condition; meaning that all 3 translational DOF are set to 0. More details regarding the documentation of `ABAQUS` are found in Duval et al. (2014).

All point loads are defined in the y direction and attach to one of the 60 remaining free nodes at the top boundary of the beam, γ_{top}. Since the width of the beam is discretized uniformly into 4 nodes along the z direction, it is impossible to apply a force right above the centroid of the beam, resulting in torsion along the length of the beam; even if the point load only has a y component, all 3 components of the deflection are garnered. A typical deflection generated by the numerical approach is presented visually in an exaggerated manner in Figure 2.7. The numerical approach is validated against the analytical solution in A.1.

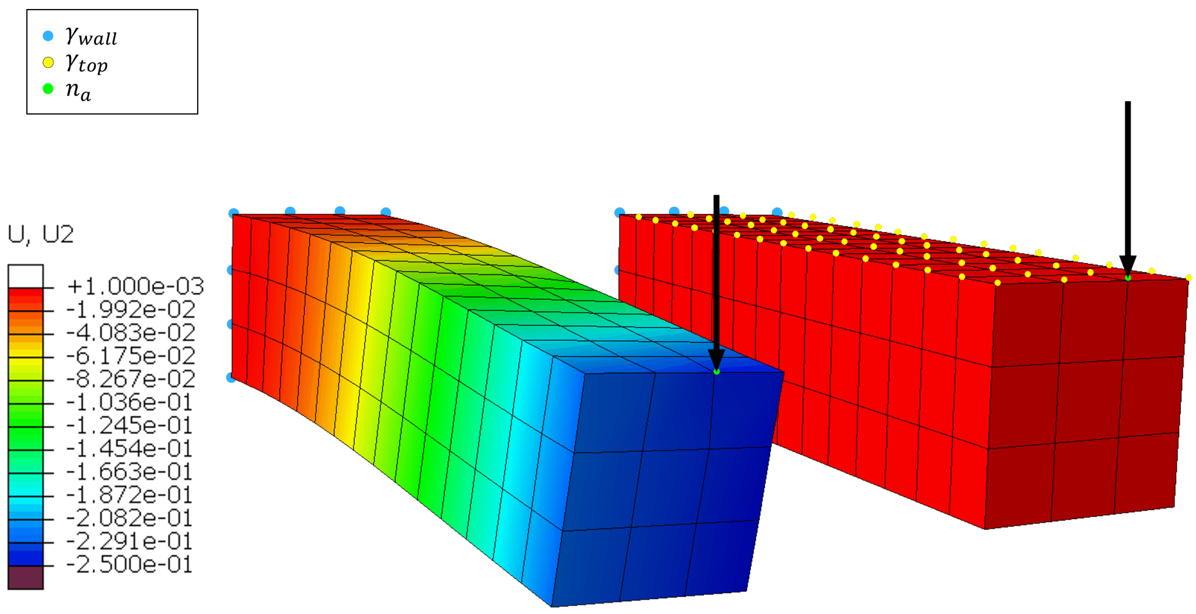


Figure 2.7: Visual representation of the finite element solution to the elastic deformation of a cantilever beam subjected to a single point load ($F = -1, a = 4$ m). Left: deformed state, right: undeformed state.

2.3 Machine learning fundamentals

In this section, the fundamentals of machine learning are explained to a point needed to understand the development of the model and to interpret the results that are achieved. The development of the model itself, including several design choices, is treated in Section 3.1. At its most simple level, machine learning is the process of parsing labeled (supervised) or unlabeled (unsupervised) data, recognizing patterns, and then predicting a classification or making a regression based on those patterns. These classification or prediction tasks can be solved using a variety of approaches from the field of machine learning. The fundamental theory that is needed for our approach is explained in this section. A general rendition of artificial neural networks is presented first, followed by a description on the training and validation process. The concept of convolution, which is a central building block in image processing neural networks, is explained and followed by an introduction to physics-driven generalization; finalizing with an explanation on the networks architecture.

2.3.1 Artificial neural networks

Neural Networks (NNs) are part of a family of machine learning techniques. An Artificial Neural Networks (ANN) is a type of directed graph that is modeled after biological neural networks. An ANN consists of neurons (vertices) and weights (edges) that represent a sequence of equations that is able to approximate the underlying relationships between input and output data in a given data set. If this sequence is strictly feedforward, the network is commonly referred to as a Multilayer Perceptron (MLP); a simple schematic representation is of a MLP is presented in Figure 2.8 where colored circles represent neurons in their corresponding layer and black lines represent the weighted connections. A DNN is a class of artificial neural network that has many hidden layers of which the first layers are used to extract low-level features from relatively unprocessed input data without the need for pre-processing the input data. The remaining layers are then able to extract high-level features from the low-level features.

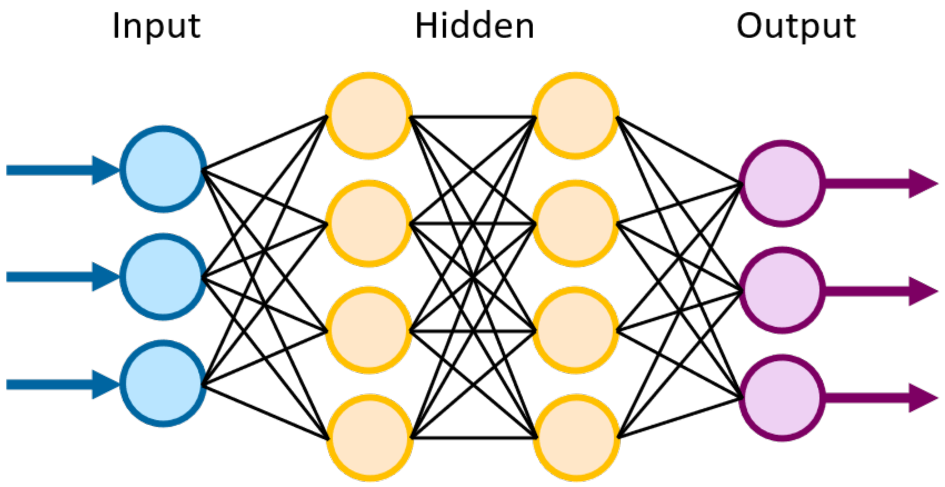


Figure 2.8: Schematic representation of aMLP

Each neuron consists of a function h_j of the input $x = (x_1, ..., x_d)$, weighted by a vector of connection weights $w_j = (w_{j,1}, ..., w_{j,d})$, completed with a bias b_j, and associated to an activation function ϕ such that

$$h_j(x) = \phi\left(w_{j,1}x_1 + ... + w_{j,d}x_d + b_j\right) \tag{2.3.1}$$

The bias b_j allows to shift the activation function curve up or down to act as a certain threshold. The activation function ϕ introduces a non-linearity to the signal and converts it to a specific output range for better convergence. A simplified example of this calculation is presented in Appendix A.2. All neurons in the sequence eventually determine the output signal of the model. The goal of training a neural network is to tweak all network parameters (weights and biases) so that the output matches to a desired outcome. This is often done by introducing a Stochastic Gradient Descent (SGD) optimization algorithm that is driven by a backbone, known as backpropagation, which calculates the gradients of the output over all neurons using the chain rule. The training process is described in more details in section 2.3.2.

There are several choices for activation functions; each with their own pros and cons. The Sigmoid activation function is a popular activation function since it is continuously differentiable and converts all input ranges to $[0;1]$. It is however computationally expensive because of the exponential term and it suffers from vanishing gradients, making it less ideal for DNN's. The problem of vanishing gradients is inherent to the backpropagation optimization. As we proceed backwards in the , the gradients become smaller and smaller, making training of the first layers more difficult. Training the model with vanishing gradients might be expensive and results in low prediction accuracy. The Rectified Linear Unit (ReLU) activation function is another popular choice that is not affected by vanishing gradients. It is computationally inexpensive and speeds up the convergence of the SGD optimization algorithm. Often linear activations are used to transform the signal to a preferred output. Both the sigmoid and the ReLU function are presented visually in Figures A.3a and A.3b respectively.

$$\mathrm{sigmoid}(x) = \frac{1}{1+e^{-x}} \qquad \mathrm{ReLU}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \qquad \mathrm{lin}(x) = x \tag{2.3.2}$$

Because the ReLU function and its derivative are both equal to 0 for negative values, no information for the concerned neuron can be acquired in the negative domain. To solve this problem, a slight positive bias, ρ, can be applied to the negative domain of the ReLU activation function to ensure that each neuron is activated: where ρ is either a fixed parameter set to a small positive value known as the LeakyReLU activation function, or a parameter that needs tuning resulting in the Parametric ReLU (PReLU) activation function.

$$\mathrm{PReLU}(x) = \begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \tag{2.3.3}$$

The size and structure of the network is known as the architecture. The size of the input and output layer is typically defined by the data set that is used, i.e. the shape of its input data and of the required output of the model. The amount of hidden layers and the number of neurons for each layer are design choices.

Fully connected networks, such as the MLP, are not always the best design choice when it comes to efficiency since it requires the maximum amount of network parameters that need to be tuned; slowing the learning process and introducing a high risk of overfitting. Instead, many networks, typically in the area of image or time-series processing, make use of convolutional layers that are able to extract detailed features from only a select window of neurons . This window (also filter or kernel) is parsed over the input layer ensuring that every input neuron is used at least once. A classic example of such a Convolutional Neural Network (CNN), is the U-net framework developed by Ronneberger et al. (2015) which is used for semantic segmentation of Red Green Blue (RGB) images. CNN's could greatly reduce the amount of network parameters compared to MLP's, while still being able to properly extract the desired and detailed features. This works because the neighbouring neurons in image and time-series processing are correlated, which is extracted as local feature by the convolution block. Subsequent layers make combinations of the local features from predecessing layers which leads to higher order local features, and eventually to a global feature map: the model output. Especially in networks with large input arrays, such as high definition RGB images, is the reduction in computation time for convolution immense compared to fully connected.

In order to use convolution filters, it is expected that the input data is structured in a grid-like formation due to filter's spatial representation of the inputs. FEM fortunately operates inherently on such a structure, which is the main reason why the proposed method is such a good fit to the problem. A visual example of a single convolution step in a 3-dimensional gridded network is depicted in Figure 2.9. A more in depth explanation on the convolutional layer is given in Appendix A.3

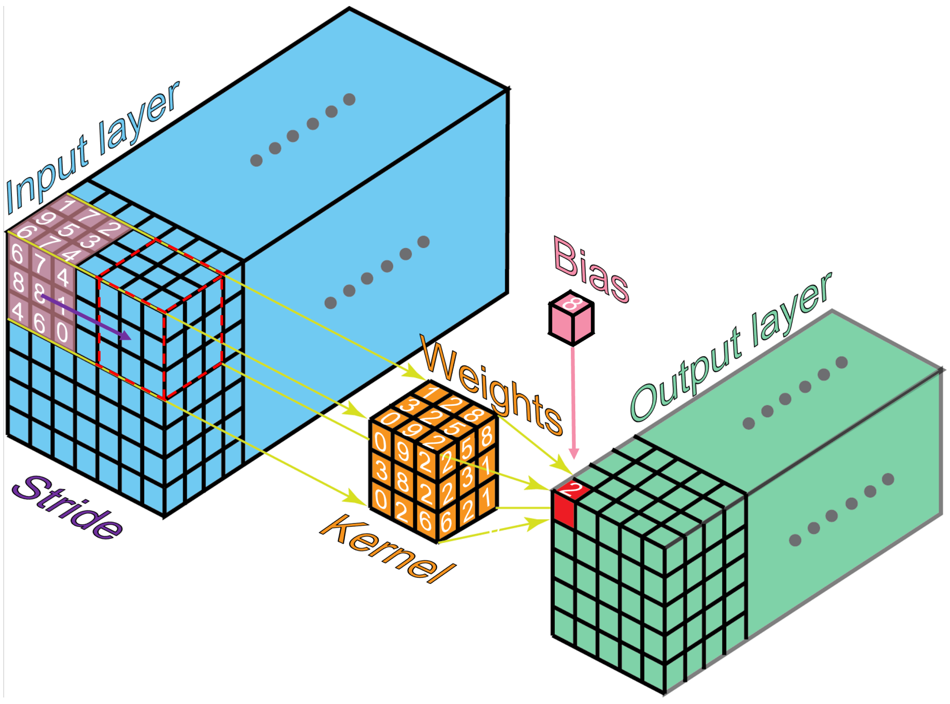


Figure 2.9: Schematic representation of a single 3-D convolution step. Based on: `www.towardsdatascience.com`

2.3.2 Training and validation

Training is done by introducing a loss function that compares the model results (or prediction), $\underline{\hat{\mathbf{y}}}$, to the ground truth data, $\underline{\mathbf{y}}$, resulting in a scalar loss value that depends on all network parameters. Loss functions are typically defined in such a way that the loss is minimal when the model results are equal to the ground truth. Common loss functions for regression are Root Mean Square Error (RMS) or Mean Absolute Error (MAE).

$$\mathrm{RMS}(\underline{\mathbf{y}}, \underline{\hat{\mathbf{y}}}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \qquad \mathrm{MAE}(\underline{\mathbf{y}}, \underline{\hat{\mathbf{y}}}) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.3.4}$$

The network has a good representation of the input data when the loss is minimal for all input data. The loss is minimized with an optimizing algorithm (or optimizer). Common optimizers, such as SGD or Adam (Kingma and Ba, 2015), find the minimum of the loss function by iteratively calculating the gradient of the loss function over all network parameters and by updating the network towards the direction of the steepest gradient. The gradients are calculated through backpropagation, which is generally the fastest way to do this.

The step size with which the optimizer finds a local minimum is determined by the learning rate, which is also a variable that is chosen. Picking the right learning rate tends to be difficult. If the learning rate is too small, no major differences will occur in the loss and thus the network will not converge. If the learning rate is too large, it is plausible that the optimizers steps over the local minimum and is not able to converge towards it. Both these concepts are presented visually in Figures A.8 and A.7. Generally speaking, a learning rate of $1e-3$ is a good starting point for both the SGD and Adam optimizer. From there, the ideal learning rate could be found by trial and error. Using a decaying learning rate, thus decreasing the learning rate over a training session, could help in converging in an efficient manner. The Adam optimizer uses such a decaying learning rate, which often makes it a go-to optimizer for most networks.

The data set is commonly divided uniformly over several mini-batches that each contain a set amount of samples. Instead of updating the network parameters for each sample, the optimizer tunes the network after each mini-batch which results in a more general model. Once the full data set is used, it is shuffled and new mini-batches are formed; continuing the training session. Using the full data set once is referred to as an epoch. A typical training example uses 80% of the data for training, and 20% for validating the model during the training session. So for example, a data set consists of 1000 samples, 800 for training and 200 for validation after each epoch. A batch size of 10 is chosen, which results in 80 batches per epoch. The number of epochs is typically very large, for example 1000. The network is then trained on a total of 80.000 different combinations of batches (each of 10 samples) and validated on 200 samples.

Initialization at the start of training it is needed so that all weights and biases have some starting value. There are several possible ways to initialize these starting values, which have a certain effect on the convergence speed and final accuracy of the network. Common initialization algorithms are zeros initialization, random initialization or Glorot (or Xavier) uniform initialization (Glorot and Bengio, 2010).

After each epoch the network tries to make a prediction on the validation set, which is data that has not been used during training; avoiding a biased outcome. This prediction is then used for validation of the network. Validation can be performed by the same validation metric as the loss, but also with other metrics. This prediction is purely used to give an idea how well the network is already performing, and not to train the network further. The validation could be used to set a threshold at which the network predicts accurate enough to stop training. The validation loss is typically larger than the training loss and varies more during training. The choice for validation metrics is up to the designer of the network and could be as simple as the mean error and mean standard deviation. In addition to the validation metrics, it is always a good idea to have some visual confirmation of the prediction compared to the true output which yields better understanding of the situation.

2.3.3 Physics-driven regularization

Extra rules concerning the training process of the network parameters could be introduced by implementing a regularizer in the network. A regularizer basically is an extra term added to the loss function that is implemented to discourage certain solutions of the network. Regularizers are commonly used to prevent overfitting, but in some cases it could also be used to discourage solutions that are physically not feasible. This hybrid modelling technique, known as Physics-Driven Regularization (PDR) (Nabian and Meidani, 2020), allows the network to converge faster and results in better performance. This is done by adding a rule that heavily penalizes predictions that are physically impossible, such as the orange beam deflection depicted in Figure 2.10, which is a intermediate prediction of a network during a training session. This prediction clearly does not (yet) represent the given input data (blue), but it also does not represent a realistic deflection at all. According to the minimum total potential energy principle, the beam should always deform in a shape that minimizes the elastic strain energy; which results in a smooth deflection over the beam with the least elongation as possible. In continuum mechanics, a constitutive equation is used to formulate this physical relation between forces and displacements, resulting in a solution that matches this principle. A neural network does not have such information hard coded in its roots, but adding it as regularizer is a step in the right direction.

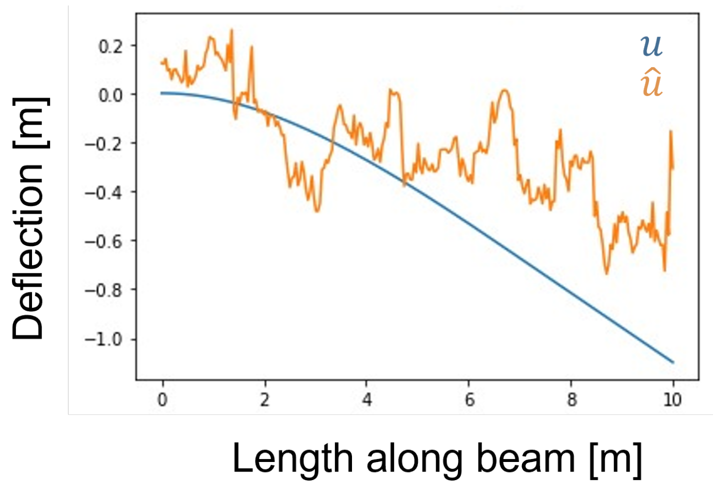


Figure 2.10: Example of a physically not feasible prediction of the elastic deformation of a cantilever beam, $\hat{u}$, compared to its true deflection u

A simple example of such a physics-driven regularization rule would be to implement a conservation law. A more specific application could be to use the equilibriums of forces, moments and/or moments of inertia. When the applied use case is known, as in our example with a cantilever beam, one could also opt for an analytical formulation of known deformation behaviour; in this case the relation between the loads and deflections is fully described with the beam equation (2.1.6).

One could also introduce a more general rule that is maybe not a full physical law, but is true nonetheless in a physical example. An example of such a rule could be regularization that discourages large differences in deformations between consecutive nodal points for elastic deformations.

2.3.4 Architecture

Mendizabal (2020) proposes a framework by the name of 'U-mesh' that is based on the U-net architecture. The U-mesh framework is able to perform complex and complete volume deformation calculations of arbitrary shapes and allows for extremely fast and accurate simulations. Such a network can learn to mimic a desired bio-mechanical model based on Finite Element Analysis (FEA) generated input data, and predict deformations at high rates with very good accuracy. U-mesh has an architecture that is similar to an auto-encoder. Auto-encoders typically transform the input space into a low-dimensional representation with an encoding path, and expand it back to its original size through a decoding path (Roewer-Despres et al., 2018). Constraining the latent feature vector space to be small, ensures that the network will learn salient features from the high dimensional input data. This method is intuitively similar to space reduction in POD. Additional copy and crop connections, the grey arrows in figure 2.11, transfer detailed information along matching levels from the encoding path to the decoding path.

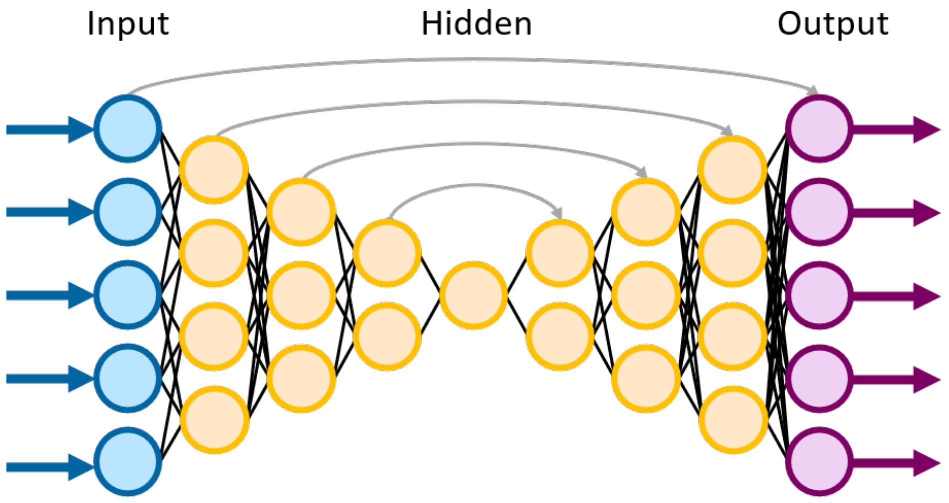


Figure 2.11: Simplified schematic representation of U-mesh architecture

U-mesh uses a 4d-array (or `tensor`) of traction forces $\underline{\mathbf{F}}$ as input data. Each traction force is described by a 3-D vector at a nodal point in 3-D space. The domain Ω is partitioned uniformly in each direction by a grid of resolution $\omega_x \times \omega_y \times \omega_z$. The input data is accordingly described by an array of $\omega_x \times \omega_y \times \omega_z \times 3$. The network produces an equal sized array of displacements $\underline{\mathbf{U}}$ as output.

U-mesh makes use of 3 types of operation layers: convolutional layers, pooling layers and upsampling layers. These operations are explained in more detail in Appendix A.3. The number of consecutive operation layers determines the network depth. As shown in Figure **??**, the network is built up of 3 sections: the encoding path, the bottleneck and the decoding path. The encoding path consists of k sequences, or steps, of two convolutional layers and a max pooling layer. The number of filters, f, is doubled for each step. This also doubles the number of output feature maps, or channels c. The spatial dimensions are halved for each step by the pooling layer. The bottleneck consists of two extra convolutional layers that lead to $(c \times 2^k)$ output feature maps. The decoding path is symmetrically to the encoding path to ensure transformation back to the original input size. It is made up of k sequences of a `transposed convolution` layer that is used for upsampling, followed by two convolutions. During each step of the decoding path, the number of filters are halved and the spatial dimensions are doubled. The features from the same step of the encoding path are and concatenated to the upsampled feature maps. There is a final convolutional layer to transform the last feature map to the desired number of channels of the output.

The size of the network is determined by hyperparameters k and c. Instead of expressing the network depth as the amount of subsequent layers, the number of steps k is used. For example, a network of depth 4, has 4 sequences of 3 layers in the encoding and the decoding path, 1 sequence of 3 layers in the bottleneck and an additional sequence of 3 layers at the output, resulting in $(4 + 1 + 4 + 1) * 3 = 30$ layers.

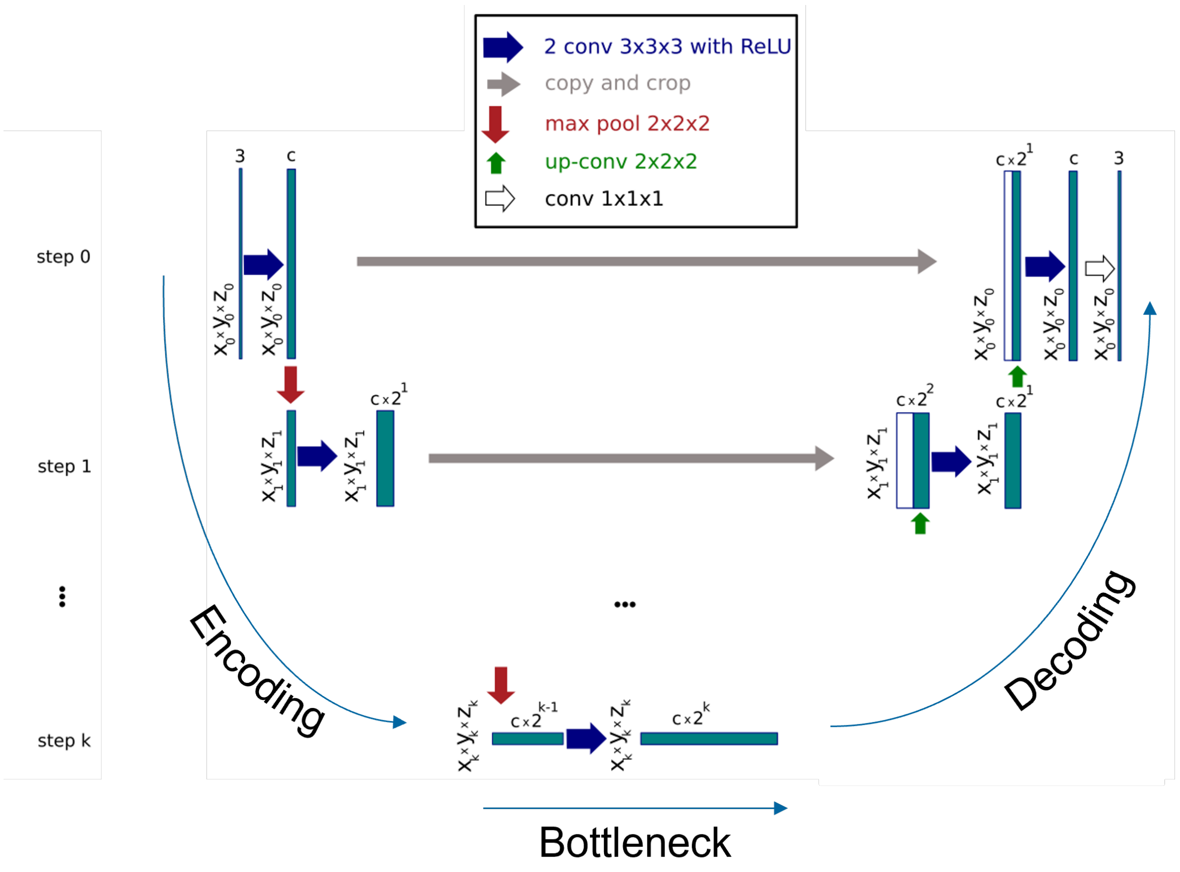


Figure 2.12: General U-mesh network architecture for an object with a resolution of $X_0 \times Y_0 \times Z_0$ nodes, c channels in the first layer and k steps (Mendizabal, 2020)

3. Computational experiments and results

Since the U-mesh framework of Mendizabal (2020) was not open source at the time of this work, it was decided to replicate their modelling approach as good as possible. This means that all design choices are in the first place made so that it resembles their work the best. The replicate, from now on referred to as *the model*, is conceptually the same as U-mesh. Still, multiple deviations were eventually introduced. The development of the model is denoted elaborately in Section 3.1 and shows the reasoning behind the decisions that were made concerning the various hyperparameters. The section is finalized with a summary of all used hyperparameters. The model is then tested in two computational experiments, where the first experiment uses a data set that is generated analytically and the second generated with FEM.

3.1 Applied modelling approach

Figure 3.1 shows a schematic overview of the applied method. The input data may be generated analytically or numerically. The model then makes a prediction of the deflection, $\hat{u}$, based on the input traction forces, F. The prediction is then compared to the ground truth deflection in the loss function, which is then used to update the network parameters through optimization. The final output of the model is a prediction of the deflection that represents the ground truth.

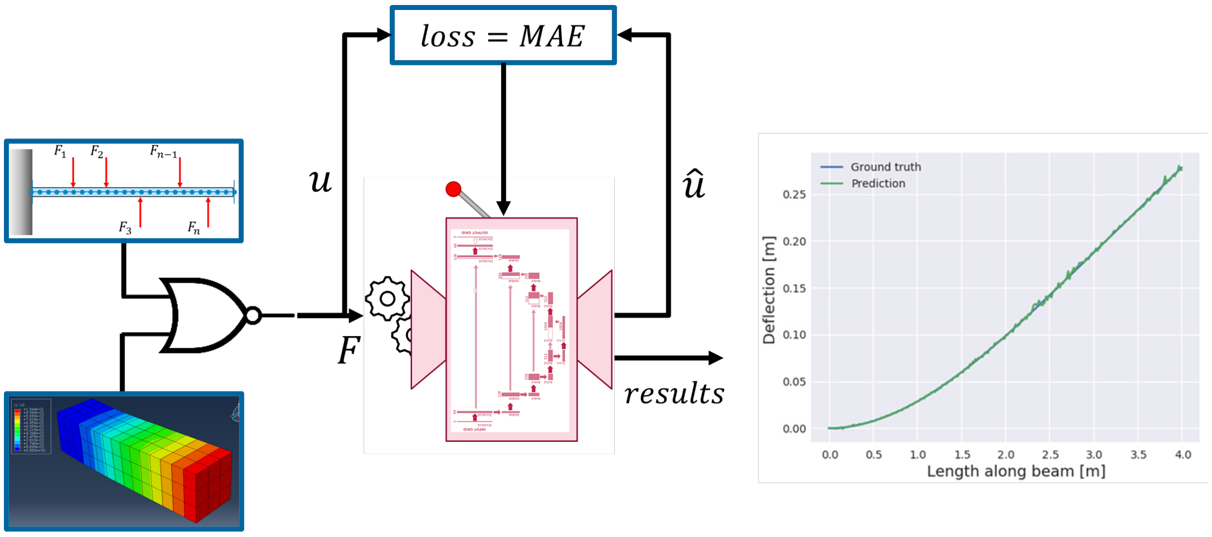


Figure 3.1: Schematic overview of applied method

3.1.1 Digital environment

The model is developed in `Python 3.7` using the `Keras` library used in `TensorFlow 1.15.0` (Abadi et al., 2016) and `PyCharm 2020.3.5` in a `Linux` environment. All training sessions and calculations are performed on a Nvidia Tesla T4 GPU (Nvidia, 2018).

3.1.2 Data structure

The structure of the input and output data is generalized for both use cases so that the model is able to process both without major adaptations and for easy post processing of the results. `TensorFlow` makes use of the `tensor` class, which is a multidimensional array that is able to store data much like the `np.array` class, but then immutable. `Tensors` are also backed by the accelerator memory, like GPU's, which gives the opportunity for much faster computing

(Abadi et al., 2016; **?**). Note that this definition of a `tensor` is not the same as a tensor used in continuum mechanics; which is defined as a (multi)-linear mapping between two lower order tensors such as vectors (Geers and Schreurs, 2015). To discriminate both in this thesis, all `TensorFlow` classes are denoted with a different font and all equations are denoted following the mathematical notation rules of Table A.1.

As mentioned before, the input data needs to be in a grid-like formation for convolution. The cantilever beam was evaluated at the discrete nodal points in space (as defined in section 2.1), which could be used as a grid structure. The desired grid structure and the nodal points happen to coincide in our experiments due to the rectangular nature of the cantilever beam. This is however often not the case for more complex geometries; meaning that an extra translation step between the mesh and the grid structure is needed.

The input, $\underline{\mathbf{F}}$, is defined as a sparse 5-dimensional `tensor` of shape $(N, \omega_x, \omega_y, \omega_z, c)$ of traction forces. N is the number of samples in the data set and is also defined as the size of the data set. A sample corresponds to the input load and the deflection of a single beam. ω_x, ω_y and ω_z define the starting dimensions of the grid points in each Cartesian direction. The amount of channels, c, is determined by the number of directions in which the loads and deflections are directed. Each desired Cartesian component of the deflection or force is defined in its own input or output channel. For example: for loads and deflections that are defined in only the y-direction, c is equal to one.

It is important to introduce as less bias as possible in the data set, since it greatly affects the outcome of the model. It was therefore chosen to generate the input conditions of each sample randomly within a region of interest. The amplitudes of the forces are generated randomly between -1 and 1 N to conform to the assumption of small strain deformations. The nodal point on which each force is applied, n_a, is sampled randomly between all eligible nodes; $n_a \in \{2, \ldots, n\}$ for the analytical data and $n_a \in \gamma_{top}$ for FEM data. To better understand the effects of various machine learning related design choices, it was decided to initially only use a data set with samples that are subjected to a single point load. The output `tensor` $\underline{\mathbf{u}}$ is defined as the displacement (or deflection) of the cantilever beam and has the same dimensionality as $\underline{\mathbf{F}}$.

Another important factor to ensure less bias, is to generate ample training data. Mendizabal (2020) concludes that the network is not able to predict reliably for loading cases that are not included in the train set. It is therefore important to ensure that each node is subjected to at least a couple of varying loads. With this in mind, the amount of samples in the whole data set is chosen to be 6000. The data set is then divided in a train set of 4800 samples, and a set of 1200 samples for validation. A separate set of 1500 samples is generated for the final testing of the model.

3.1.3 Architecture

As mentioned earlier, the architecture of the model is a replicate of U-Mesh. While the architecture style is similar, there are still many design choices in the form of hyperparameters left that affect the outcome of the model. The amount of network parameters that need to be tuned is influenced heavily by these hyperparameters, which directly affects the training time and accuracy; the prediction time is not affected that much. Some hyperparameters are properties of the different layers that form the network. Others are the network depth, the number of filters and the type of initialization.

Layer types

The network consists of the following layer types: `Conv3D`, `MaxPooling3D`, `Conv3DTranspose`, `Concatenate` and `Dense`. These layers alternate according to the scheme as depicted in Figure A.10, forming a network that has an encoding, bottleneck and decoding path. The operations of each layer are explained in detail in Appendix A.3. Layer specific hyperparameters are `Poolsize`, `Stride` and `Kernelsize`.

Network depth and channels

The number of steps, k, and the number of channels, c, control the accuracy of the prediction. Higher values of k and c lead to a deeper and more complex network that is suitable for difficult meshes at the expense of longer computational times and higher requirements of training data.

It is common practice in CNN's used for imaging problems, to use multiple filters in the first convolution layer that extract low-level features from the data, instead of manually pre-processing the input data; this number of filters inherently determines the number of channels. In contrast to these imaging problems where desired features could be very local, our problem is in need of a global output feature map since each point load effectively influences the deflection in all nodal points of a sample. It is therefore important to ensure that throughout the network each input neuron is eventually connected to each output neuron. This is done by choosing a proper network depth k in combination with the kernel and pooling size. We assume as a rule of thumb that the network needs to be deep enough so that the kernel covers all neurons in the bottleneck.

Our data is not in need of pre-processing since it is fully generated based on known equations and therefore has no inconsistencies. It is therefore in theory not necessary to add extra filters to the first convolutional layer. A few tests were performed and it was quickly confirmed that the model does not show any need for multiple channels and managed to converge neatly with the bare minimum; $c = 2$ for the analytical data and $c = 6$ for FEM data, in contrast to $c = 64$ used by Mendizabal (2020).

Several network depths were also tested for the analytical input data, in a trade-off between accuracy and training time (Table 3.1); where $\bar{e}$ is the mean absolute error and $\sigma(e)$ the standard deviation over the error. It was chosen to proceed all training sessions with a relatively deeper network since the training times are very reasonable for all tested depths. It was however also chosen to make the network not deeper than necessary for the pooling operations. This means concretely that $k = 7$ for the analytical data and $k = 2$ for FEM data.

Table 3.1: Test results for various network depths k

k	$\bar{e}$	$\sigma(e)$	Train time [s]
1	0.0401	0.0015	161
2	0.0251	0.0060	214
3	0.0016	0.0002	265
4	0.0006	0.0002	311
5	0.0008	0.0002	363
6	0.0009	0.0002	438
7	**0.0006**	**7.1e-5**	**629**

Kernel and pooling size

The size of the kernel plays a major role in convolution and controls the accuracy and the speed of the network. A small kernel size, compared to the input structure, enables to extract more local features with convolution, while larger sizes yield more global features. Increasing the kernel size also increases the amount of trainable parameters, resulting in a larger computational costs. Some physical phenomena are not extracted better when comparing the global. A kernel of size $(3, 3, 3)$ is a good choice for the 3-D FEM input data, since it includes all direct neighbours of a neuron for each parsing step.

Since the analytical input data has a 1-D input structure, there is no use for convolution in the y and z direction. In practice, this results in a oddly shaped kernel of $(f_x, 1, 1)$, where f_x is the size of the kernel along the length of the beam. A few tests were performed to find the ideal value and $f_x = 9$ seemed to perform the best; resulting in a kernel size of $(9, 1, 1,)$ for the 1-D analytical input data.

Activations

Our network uses ReLU activations within all hidden layers and a linear function to activate the output layer. ReLU activations are generally the best choice for deep neural networks and are also the default activations used in Mendizabal (2020) and Ronneberger et al. (2015).

Initialization

All network parameters need to have some initial value when the network starts training. A couple of different initialization methods were tested and the default setting for the initialization of these parameters in `Tensorflow`, Glorot uniform initialization (Glorot and Bengio, 2010), seemed to perform the best; hence it is used for all training sessions.

3.1.4 Performance measures

Loss function

The choice for a loss function is substantial, since it determines how the performance of the network is judged to steer it towards better performance. A good first choice for a loss function in regression problems is the MAE loss. The loss function is optimized for each mini batch during the training process by expressing the loss as a scalar average of the batch. All computational experiments are performed for a batch size of $bs = 64$, since it seemed to perform best. Extra experiments for $bs = 32$ are also performed to see if significant better results could be retrieved for twice the computational cost. The loss function is defined as

$$\text{MAE}(\underline{\mathbf{u}}, \hat{\underline{\mathbf{u}}}) = \frac{1}{n}\sum_{i=1}^{n} |u_i - \hat{u}_i| \tag{3.1.1}$$

with $\underline{\mathbf{u}}$ and $\hat{\underline{\mathbf{u}}}$ the true and predicted deflections respectively, n the amount of nodes in all samples of the batch ($n = bs \cdot n_x \cdot n_y \cdot n_z$). In Chapter 4, an addition to the loss is introduced as a physics-driven regularizer.

Validation metrics

The mean absolute error $\bar{e}$ (also referred to as the error), and its standard deviation, $\sigma(e)$, are used to present results as a single scalar value as an absolute measure for the accuracy of the model. However, these measures do not always give clear insights on the performance of the model during training. The performance of the network should not only be described in an absolute fashion, but also as a function of the amplitude of the deflection. There are large differences between the amplitudes of the deflections within a data set. It seems thus logical to define some validation metric that is defined as a Mean Absolute Percentage Error (MAPE) from the ground truth.

$$\text{MAPE}(\underline{\mathbf{u}}, \underline{\hat{\mathbf{u}}}) = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{u_i - \hat{u}_i}{u_i} \right| \tag{3.1.2}$$

This formulation however does not work as intended, since each sample has at least one node with a deflection of zero, resulting in a division by zero. The metric is instead defined as the Mean Absolute Maximum Percentage Error (MAMPE), which compares the error to the maximum deflection of a sample.

$$\text{MAMPE}(\underline{\mathbf{u}}, \underline{\hat{\mathbf{u}}}) = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{u_i - \hat{u}_i}{u_{max}} \right| \qquad u_{max} = \max(|\underline{\mathbf{u}}|) \tag{3.1.3}$$

MAMPE will be used throughout all training sessions as validation metric to raise a good sense on how well the model is able to predict for all deflections

3.1.5 Optimizer

All training sessions are performed with the Adam optimizer, since it is straightforward to implement, computationally efficient, requires little memory and it's parameters require little tuning. A few tests were performed to find the ideal starting learning rate, which was chosen to be set at 0.002. The hyperparameters β_1 and β_2 are both set to 0.7. The network seemed to perform better when multiple training sessions were executed consecutively with decreasing values for the initial learning rate. Thus even though the Adam optimizer decays the learning rate by itself, it was better to add some decay manually on top of that. The learning scheme that is used is presented in table 3.2.

Table 3.2: Learning scheme

Epochs	Learning rate	Loss
300	0.002	MAE
200	0.0001	MAE
20	0.00005	MAE

Instead of setting an amount of epochs that defines the end of a training session, it is also an option to set a desired accuracy threshold for the training to be completed. But since the goal is to provide a proof of concept and compare various methods, it was decided to follow the defined learning scheme.

3.1.6 Summary

The Table below recaps all final modelling choices once more and gives a clear overview of the used settings during the computational experiments in the following sections. A distinction is made between the two different data sets in the top part of the table, while the bottom part of the table presents the learning scheme that is used for both experiments. A visual overview of the network architecture of both models is presented in Figures A.10 and A.11.

Table 3.3: Summary of applied hyperparameters and learning scheme throughout experimentation

Hyperparameters	Analytical	FEM
Train set size	4800	4800
Validation set size	1200	1200
Test set size	1500	1500
Batch size	64	64
Geometry	(256,1,1)	(16,4,4)
Channels	1	3
Repetitions	6	6
Kernel size	(9,1,1)	(3,3,3)
Pooling size	(2,1,1)	(2,2,2)
Network Depth	7	2
Trainable parameters	2,164,395	24.705

Optimizer	Adam	$\beta_1 = 0.7$	$\beta_2 = 0.7$

Step	Epochs	Loss	Learning rate
1	300	MAE	0.002
2	200	MAE	0.0001
3	20	MAE	0.0005

3.2 Experiment: Analytically Generated Input Data

The first data set was generated analytically instead of numerically, since it offers more flexibility for trial and error throughout the development of the model, while still being able to paint a clear picture on the expected results of the model. This section is divided in three parts: generation of the input data, the process of training and validating the model, results and intermediate conclusions.

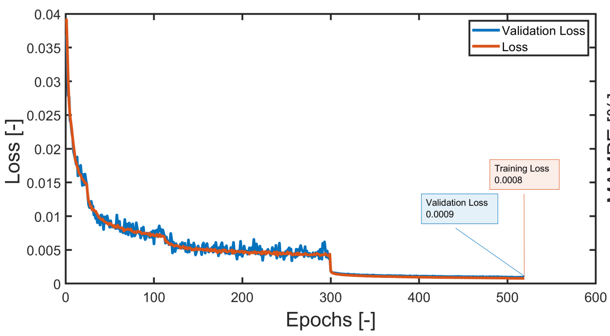
3.2.1 Input data

The first data set is generated with an algorithm that was developed in `MATLAB 2020b` (Mathworks Inc., 2020). This algorithm is based on the analytical solution presented in 2.1.1. A full data set is generated in 0.5 seconds and consists of 2 matrices of 6000 samples of 256 nodal values; 1 sparse matrix that contains the input forces and 1 matrix that contains the respective deflections, as defined in Section 2.1. The forces are generated randomly between $-1 < F < -0.1$ and $0.1 < F < 1N$, and are not applied on the 10 nodes closest to the wall to reduce the ratio between the smallest and largest absolute output deflections. A visual representation of the data set is depicted in Figure B.8.

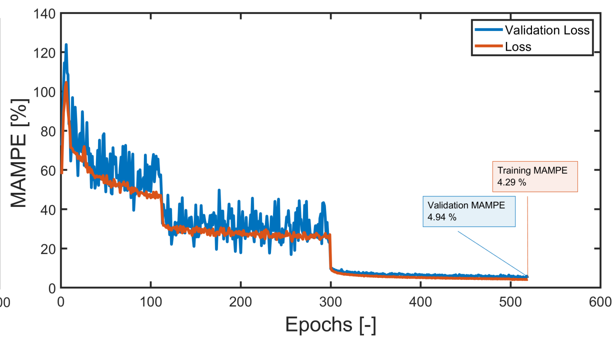
The deflection at the first node of each sample is equal to zero and all other entries are filled with values, generally descending or ascending along the length of the beam, depending on the loading conditions. For a single point load, the absolute maximum deflection is at the end of the beam. This maximum deflection of each sample still differs quite rigorously due to the random nature of the applied load, resulting in a ratio of approximately 4.000 between the smallest and largest end deflection.

3.2.2 Training and validation

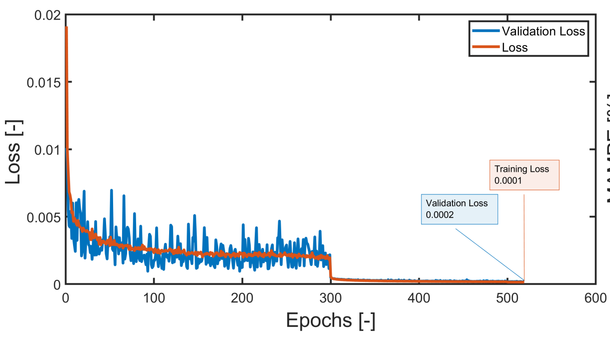
Figure 3.2 depicts the course of the loss and the MAMPE over the training session (number of epochs). Approximately 2.2 million parameters are trained in a training time of around 10 minutes. In both figures the training loss and the validation loss are shown. It is clear to see that the validation loss behaves volatile and performs a little worse compared to the training loss; which is as expected, since the model has not been trained on the data that is presented during validation. The validation loss thus gives a better prediction on how the model will behave after training. A significant decrease in the loss is apparent at 300 epochs. This is due to the change in learning rate that is imposed by the learning scheme. The loss converges nicely over time and is able to reach satisfactory values (Table 3.4) at a quick pace. This is however not always the case. Because of the stochastic nature of the training, it occasionally happens that the network converges poorly (or not at all). This also affects the final accuracy of the network significantly (e.g. the validation MAMPE could range from 2.3% to 14.9%, even for converged networks). Each experiment has therefore been performed at least 6 times and all final results are averaged. An artifact of this averaging is seen in Figure 3.2b where there is a clear sudden decrease after the 100 epochs mark. This sudden decrease is caused by a single training where the network started to converge in a later stage. The computational experiments have also been repeated with a batch size of 32 to see if there were still improvements to be made for a simple trade-off.



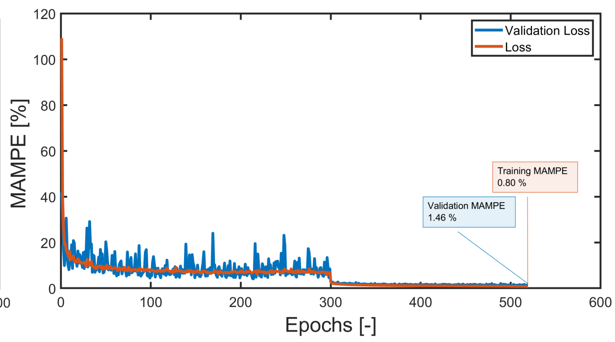
(a) Loss for batch size 64



(b) MAMPE for batch size 64



(c) Loss for batch size 32



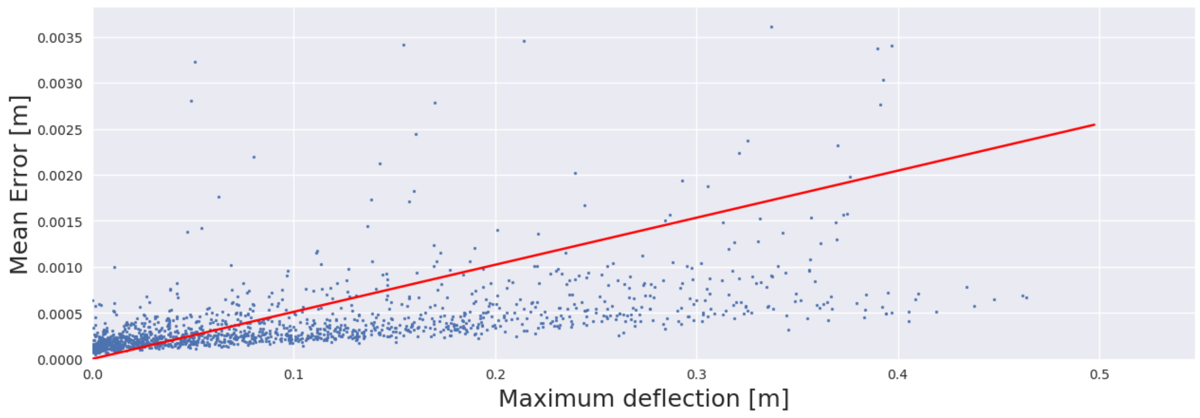
(d) MAMPE for batch size 32

Figure 3.2: The course of the training and validation metrics for the experiment with analytically generated data. All training sessions are performed with the hyperparameters shown in the left column of Table 3.3, and are tested for batch sizes of 64 and 32.

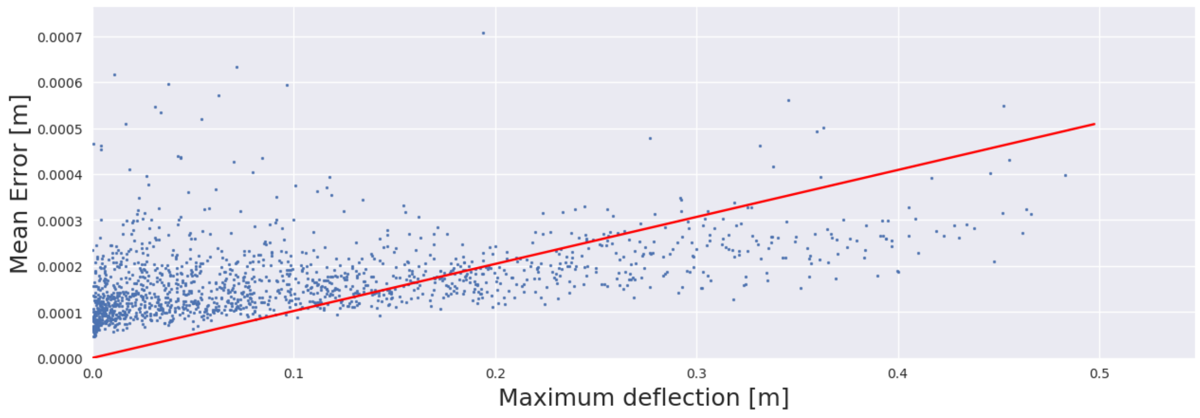
3.2.3 Results

The mean error $\bar{e}$ over the test set is equal to 0.0006 ± 0.0007 m for a maximal deformation of 0.512 m. This is very comparable to the results of Mendizabal (2020) in terms of accuracy. The prediction time per sample is 2.8 ms, which is again similar. The prediction time was measured by measuring the wall-time needed to predict the whole test set of 1500 samples, and dividing it by the number of samples. Our method has a far lower training time to reach these results. This difference may be caused by multiple sources, such as the difference in digital environment, the different input data structures and other design choices such as hyperparameters settings. Since improving the training time is of no major interest in this research, no further scrutiny will be pursued to compare both methods.

In Figure 3.3, a sensitivity analysis of the error to the amplitude of deformation is depicted. A linear least squares line fit, with slope $\alpha = 0.005$, depicts the relation between the maximal deformation and the error. The sensitivity is thus very small and comparable to Mendizabal (2020). Figure 3.4 shows a visual example of the predicted deflection of a single sample of the test set. The prediction is generally very good, apart from some fluctuations at the tip of the bar, which is most likely an artifact from the learning process. All final results are presented in Table 3.4.

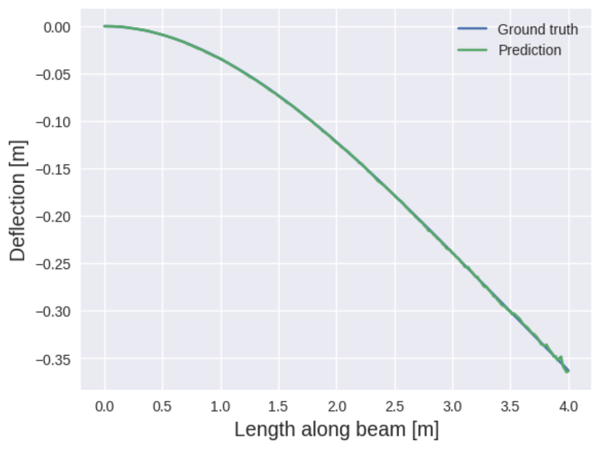


(a) Sensitivity for batch size 64, $\alpha = 0.00511$

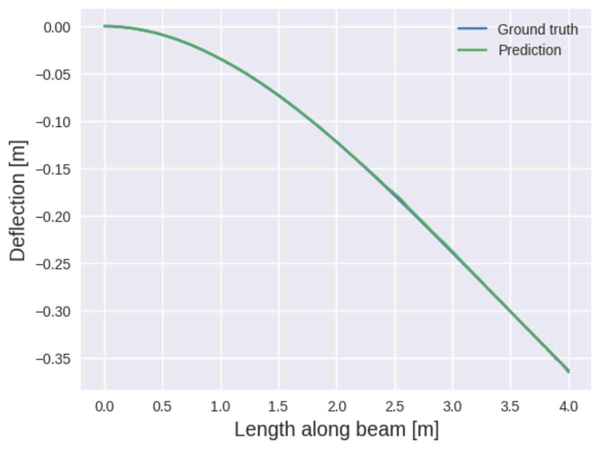


(b) Sensitivity for batch size 32, $\alpha = 0.00102$

Figure 3.3: Sensitivity analysis between the mean error and the maximum deflection for the experiment with analytically generated data. Each dot represents a single sample in the test set. The red line represents a least squares linear fit through all points.



(a) Batch size 64



(b) Batch size 32

Figure 3.4: Prediction of the deflection $\hat{u}$, compared to the ground truth u for a single example in the test set for the experiment with analytically generated data.

Table 3.4: Intermediate results for the experiment with analytically generated data.

Method	Loss	Input shape	Batch size	$\bar{e}$ [m]	$\sigma(e)$ [m]	α	Prediction time [ms]	Train time [min]
Mendizabal (2020)	MAE	(16,4,4)	4	0.0007	0.0006	0.00352	2.5	35
Analytical	MAE	(256,1,1)	64	0.0006	0.0007	0.00511	2.9	9.5
	MAE	(256,1,1)	32	0.0002	0.0002	0.00102	5.8	17.5

3.2.4 Discussion

The model is able to extract the desired behaviour from the train data in a fairly short time. Testing the model shows that its accuracy is within the region of interest. The accuracy of our model seems not to be influenced by the amplitude of the deflection very much. The time it takes for the model to make a prediction is quite fast, but not faster than the method used to generate the input data. The model seemed to perform a lot better with a smaller batch size in this experiment. Even though this should be theoretically also the case, there is still some coincidence involved regarding the amount of improvement compared to larger batch size.

A common limitation for neural networks is their lack to extrapolate reliably outside of the training data. The model had for example problems with predictions where the load was placed on a node that had not seen a load in the train set before. This entails that for any problem, a large unbiased data set needs to be present; making the model very data dependent.

The model sometimes tended to have trouble with converging. It is not clear why this behaviour occurs, but it is probably due to a small bug in the source code that could not be found. Due to the varying outcomes that result from this behaviour, it is crucial to perform many repetitions of an experiment and validate each result to guarantee reproducability. For the latter aspect, it would also be good idea to make use of a random seed.

The original problem was defined with multiple point-loads per sample. This was sadly not tested anymore since other modelling aspects required more attention. It would be interesting to still study the effect of more input values on the convolution.

3.3 Experiment: FEM Generated Input Data

The first experiment proved that the network is capable of generating adequate results. The original goal of this method is however to speed up the prediction time. This is however impossible compared to the analytically generated input data, since it is already super fast. In order to make a proper comparison in the computational gain and to proof the models capability to extract the deformation behaviour of a 3 dimensional geometry, an experiment is performed with FEM generated data.

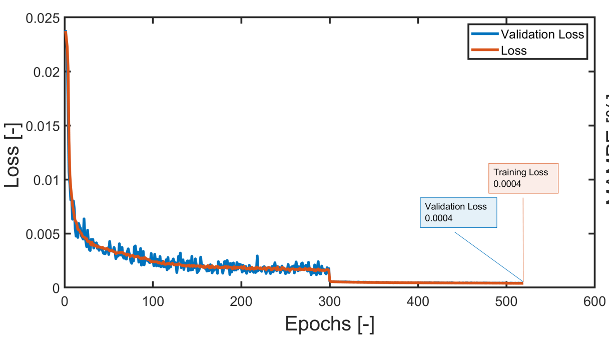
3.3.1 Input data

This data set is generated in `ABAQUS` according to Section 2.2.1. Since FEM uses a lot more steps in solving the problem, and each sample is solved sequential, the time to generate a full data set is much longer compared to the analytical method. The time needed to generate a full data set is about 12 hours, which is about 7.2 seconds per sample. Note that this is wall time, and not processing time. Solving the actual systems of equations probably would not take that much time for this problem, however also all communication costs should be considered to make a fair comparison. The computation speed is also influenced by the digital environment and the bandwidth and signal strength of the internet connection.

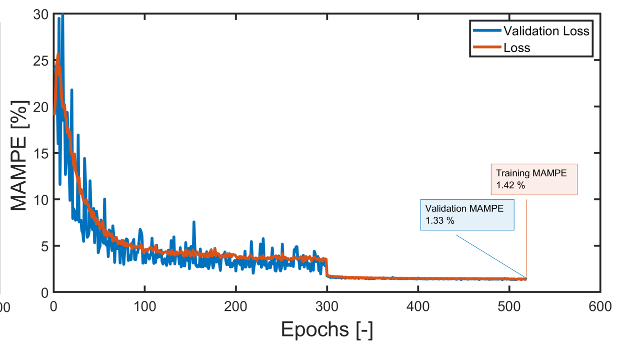
A full data set consists of 4 matrices of 6000 samples of 256 nodal values; 1 sparse matrix that contains the input forces and 3 matrices that contain the respective deflections, as defined in Section 2.1. Contrary to the analytical data set, FEM data set has applied traction forces with amplitudes in the range of $-1 < F < 1$, leading to a ratio of approximately 8000 between the smallest and largest end deflection.

3.3.2 Training and validation

The hyperparameters in the right column of Table 3.3 are used during this experiment.The time needed to perform a training session following the learning scheme is only about 4.5 minutes; which is significantly shorter due to the smaller amount of trainable parameters in the network. Figures 3.5 shows the course of the loss during training with a batch size of 64; the results for a batch size of 32 are found in Appendix B.1. Once again the network converges nicely for both batch sizes.



(a) Loss



(b) MAMPE

Figure 3.5: The course of the training and validation metrics for the experiment with FEM generated data. All training sessions are performed with the hyperparameters shown in the right column of Table 3.3

3.3.3 Results

The results of this experiment are shown next to the previous results in Table 3.5. This experiment seems to perform with a better accuracy than in the previous results. The prediction time per sample is also improved to almost twice as fast and more than 5000 times faster than generating the data in FEM. The sensitivity analysis (Figure B.2) once again shows low sensitivity of the model to the deformation amplitude. Figure 3.6 shows an example of prediction along the length of the beam compared to the corresponding true deflection. Halving the batch size did not seem to improve much for this experiment; all visual results for a batch size of 32 are shown in Appendix B.1.

Table 3.5: Intermediate results for the experiment with FEM generated data.

Method	Loss	Input shape	Batch size	$\bar{e}$ [m]	$\sigma(e)$ [m]	α	Prediction time [ms]	Train time [min]
Mendizabal (2020)	MAE	(16,4,4)	4	0.0007	0.0006	0.00352	2.5	35
Analytical	MAE	(256,1,1)	64	0.0006	0.0007	0.00511	2.9	9.5
	MAE	(256,1,1)	32	0.0002	0.0002	0.00102	5.8	17.5
FEM	MAE	(16,4,4)	64	0.0005	0.0004	0.00128	1.4	4.5
	MAE	(16,4,4)	32	0.0004	0.0005	0.00118	3.0	9

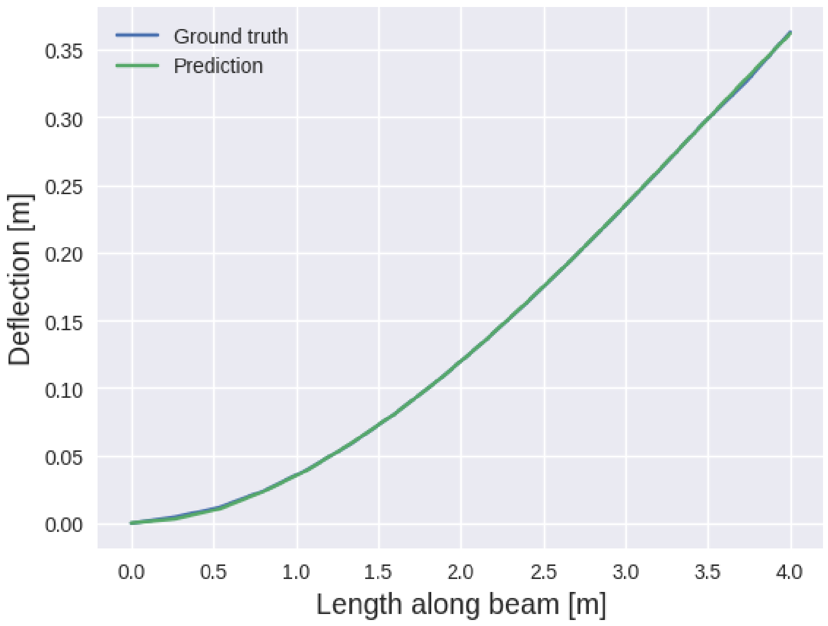


Figure 3.6: Prediction of the deflection $\hat{u}$, compared to the ground truth u for a single example in the test set for the experiment with FEM generated data and batch size 64.

3.3.4 Discussion

The computational experiment above shows that the model is able to correctly extract the desired deformation behaviour for a simple 3-D problem. This experiment had notably better results than the previous experiment for learning time, accuracy and prediction time. The decrease in learning time is most likely caused by the significantly less trainable parameters. It is not fully clear why the accuracy is improved, but there are a couple of potential causes.

By structuring the input data in a 3-D shape, the network is given more implicit information on the deformation behaviour of neighbouring nodes through convolution, which could result in a better model. It could also be the case that the accuracy is improved because of the kernel size that is larger with respect to the input data, making it easier for the network to make connections between the input and output layers. Finally, the improvement in accuracy could be caused by the fact that the data in this experiment also has some small torsion components. This torsion could be recognized by one of the filters which may result in more feature maps that originate from the same physical laws and eventually result in a better extraction of the desired behaviour.

The prediction time is lower than in the experiments before, but most notably far superior to FEM. This potential speed up is the core reason that the proposed method could improve real life applications. An important caveat to state however, is that a more fair and elaborate measurement on prediction time should be carried out for both the model as for FEM. The first step is to use the same hardware for bot methods. It would also be better to separately keep track of wall time and processing time since many communication costs are currently unaccounted for. Using larger problems could also make the influence of communication costs negligible.

4. Application of Physics-Driven Regularization

The problems that are being solved by the model are of a physical nature. All input data is fully generated according to physical laws, which inherently entails that the model should only extract physically feasible information from the data. Generally, this is also the case. However, during the learning process the model still produces many intermediate predictions that are physically impossible. Often though, the desired physical information is not totally unknown to the developer of the model; the solutions to the problems presented earlier are even fully known. This knowledge could be implemented into the model to aid the learning process, defining a hybrid modelling approach. Hybrid models could improve the learning time, convergence rate, robustness and accuracy of the model and could be used to make the model better explainable and generalizable.

In this section, an exploration is done to the impact of a hybrid modeling addition to the U-mesh model. Methods such as PINN are of high interest but seem to be complicated and narrowed as simple addition to the U-mesh framework. Instead, it was chosen to use the PDR method of Nabian and Meidani (2020) as a first step, since it is a fairly simple addition that is generally applicable to most neural networks that describe physical phenomena. The main goal is to implement a working prototype to the model and to explore the boundaries of PDR.

4.1 Implementation of PDR

Since the problem is fully known, there is some comfort in choosing a fitting additional term that fully describes the desired solution. In a perfect network, the prediction of the deflection $\hat{u}$ should be equal to the true deflection u. If that is the case, then logically the residual of the beam Equation (eq. 2.1.6) should be equal to zero for the prediction. This entails that the residual form of the Euler-Bernoulli is a good candidate to be used as regularization term in the loss function.

$$r = EI\frac{d^4\hat{u}}{dx^4} - F(x) \tag{4.1.1}$$

where r is the residual and $\bar{u}$ is the predicted deflection. Adding this residual form of the beam equation to the loss function thus forces the network to converge to a solution that is sound with the beam equation by penalizing all other solutions. Each term in the loss function is weighted by a factor λ, which is set by the model architect. λ could be set as trainable parameter, but is generally set as a manual tuner. For now, λ is set to be 1.

$$\text{Loss} = \text{MAE} + \lambda r \tag{4.1.2}$$

A schematic overview of the proposed method is depicted in Fugirue 4.1.

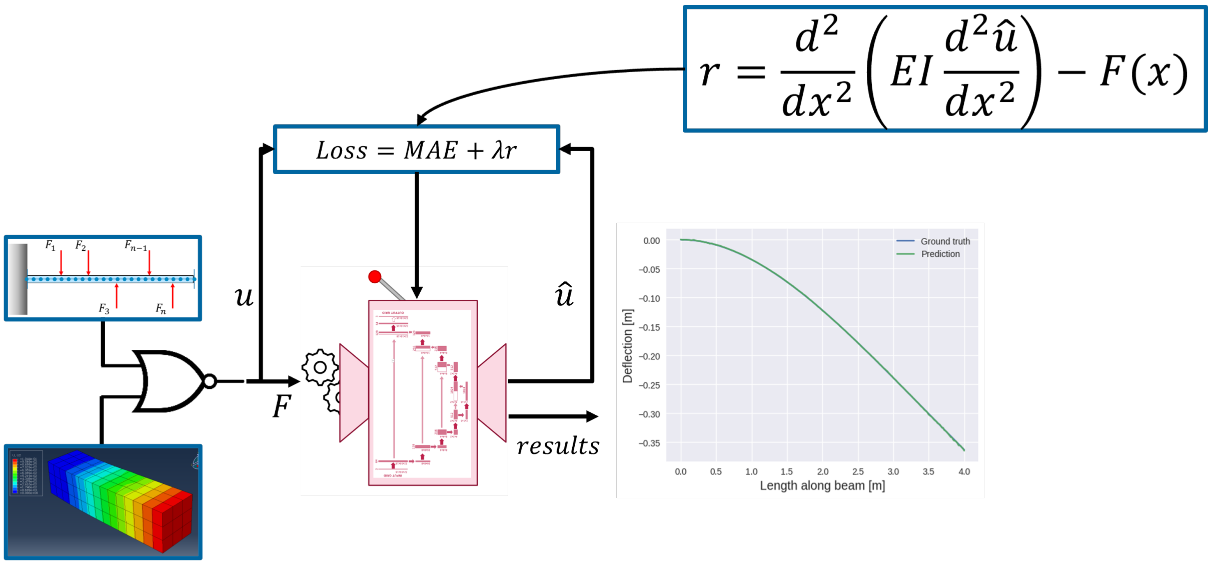


Figure 4.1: Overview of proposed physics-driven regularization method.

To calculate the residue, the predicted deflection needs to be differentiated 4 times over the length of the beam. Since the predicted deflection is only available as discrete values, it seems attracting to perform numerical differentiation. The problem with this approach is however, that the prediction throughout the whole learning process is noisy. A visual example of such a prediction is shown in the figure below, where the blue line represents the true deflection due the the force prescribed by the red arrow, and the black line the noisy prediction of the deflection. Numerical differentiation of noisy data is very unstable and elevates the results to a whole different order of magnitude. Since the learning process is recursive, the model never converges.

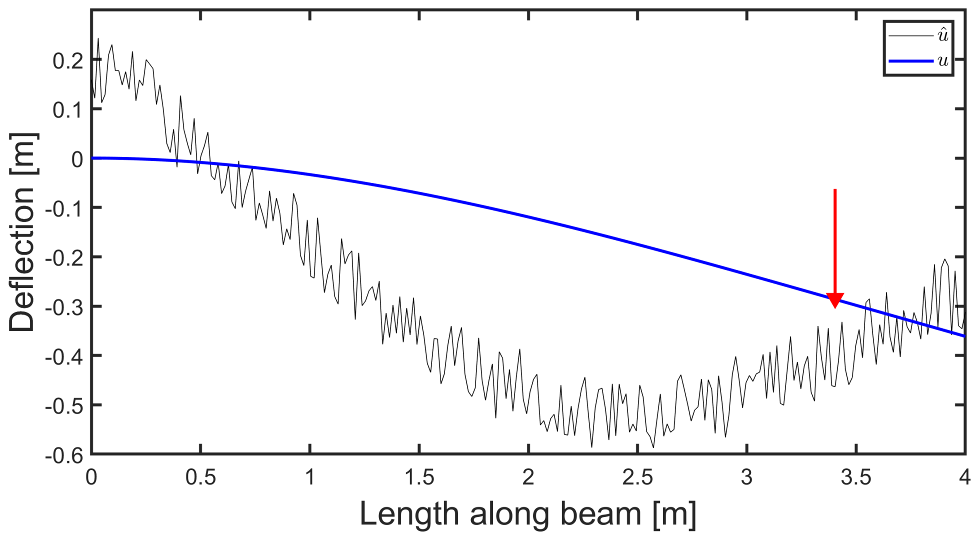


Figure 4.2: Example of a noisy intermediate prediction of the deflection $\hat{u}$ compared to the ground truth u resulting from the point-load depicted by the red arrow.

To solve this problem, a fitting algorithm is introduced to construct a continuously differentiable prediction of the deflection u_{fit}; the derivation of this function is presented in Appendix C.

$$u_{fit} = \begin{cases} C_4 x^2 \left(x - 3a\right), & x \leq a \\ C_4 a^2 \left(a - 3x\right), & a \leq x \end{cases} \tag{4.1.3}$$

with

$$C_4 = \frac{\sum_{i=1}^{n_a} \hat{u}_i \left(x_i^2 \left(x_i - 3a\right)\right) + \sum_{i=n_a}^{n} \hat{u}_i \left(a^2 \left(a - 3x_i\right)\right)}{\sum_{i=1}^{n_a} \left(x_i^2 \left(x_i - 3a\right)\right)^2 + \sum_{i=n_a}^{n} \left(a^2 \left(a - 3x_i\right)\right)^2} \tag{4.1.4}$$

where n_a is the node where the load is applied with distance a from the wall, $\hat{u}_i$ is the predicted deflection and x_i the position along the length of the beam for each node.

This fit is then used in calculating the residual of the beam equation during training. The 4^{th} order derivative of u_{fit} is equal to zero for all x, except for $x = a$ where it is equal to $-6C_4$

$$\frac{d^4 u_{\text{fit}}}{dx^4} = \begin{cases} 0, & x < a < x \\ -6C_4, & x = a \end{cases} \tag{4.1.5}$$

The residual of the beam equation then becomes

$$r = EI \frac{d^4 u_{fit}}{dx^4} - F(x) \qquad \Rightarrow \qquad r = -6DEI - F(x) \tag{4.1.6}$$

In the derivation of this solution, we force the fitting line to always have the shape of a correct deflection. Often however, the prediction does not have a realistic shape at all. So even though the error between the fit and prediction (e_{fit}) is minimalized, it should still be compensated for. This is done by also adding this error term to the loss function.

$$e_{fit} = \frac{1}{n} \sum_{i=1}^{n} |\hat{u} - u_{\text{fit}}| \tag{4.1.7}$$

The loss is finally defined as

$$\text{Loss} = \frac{1}{N} \sum_{j=1}^{N} \left(\text{MAE} + \lambda_1 r + \lambda_2 e_{fit}\right) \tag{4.1.8}$$

where N is the amount of samples in the batch and n the amount of nodes in each sample

4.2 Computational Experimentation of PDR

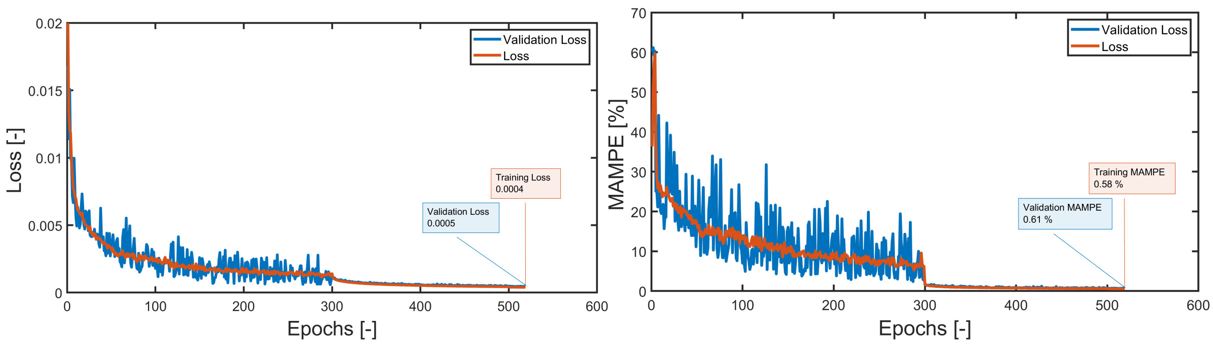
The following computational experiments have been performed with the loss function as derived in the Section above. The goal of this experiment is to provide a proof of concept for the proposed addition of physics-driven regularization, and to compare its results with the original model. This experiment uses the analytically generated data and the corresponding network configuration shown in the left column of Table 3.3. The learning scheme is altered to introduce the regularized loss function, as shown in Table 4.1

Table 4.1: Summary of settings used for the PDR computational experiment

Optimizer	Adam	$\beta_1 = 0.7$	$\beta_2 = 0.7$
Step	Epochs	Loss	Learning rate
1	300	MAE	0.002
2	200	PDR	0.0001
3	20	PDR	0.0005

4.2.1 Training and validation

At first, the model had some problems with the new loss function and was not able to converge. This problem was solved by first steering the model in the right direction through training some epochs without regularization before introducing the regularizer. This approach worked well and the model seemed to converge even faster than the previous method, as shown in the example of Figure 4.3. The loss often increases first once the regularizer is introduced. This increase is logical since the new loss function is an addition of multiple terms with absolute values; the course of the MAMPE shows that there is also no decline in performance. The time needed to perform a training session is about 10.5 minutes, which is just a bit longer than the previous method. The additional calculations needed for the regularization do not entail large additional computation costs since all calculations are analytically and no iterations are needed.



(a) Loss (b) MAMPE

Figure 4.3: The course of the training and validation metrics for the experiment with physics-driven regularization. All training sessions are performed with the hyperparameters shown in the left column of Table 3.3 and the learning scheme shown in Table 4.1

4.2.2 Results

The results of this experiment are added next to the previous results in Table 4.2. The most notable difference is found in the error, which 3 times smaller than without regularization.

This improvement in accuracy also translates to a shorter learning time needed for an equal accuracy compared to the original loss function. The predicted deflections in Figure 4.4 seem to have less fluctuations and are overall smoother than before. The prediction time per sample is increased a bit, but not significantly. Halving the batch size did not seem to improve much for this experiment; all visual results for a batch size of 32 are presented in Appendix B.2.

Table 4.2: Intermediate results for the experiment with physics-driven regularization.

Method	Loss	Input shape	Batch size	$\bar{e}$ [m]	$\sigma(e)$ [m]	α	Prediction time [ms]	Train time [min]
Mendizabal (2020)	MAE	(16,4,4)	4	0.0007	0.0006	0.00352	2.5	35
Analytical	MAE	(256,1,1)	64	0.0006	0.0007	0.00511	2.9	9.5
	MAE	(256,1,1)	32	0.0002	0.0002	0.00102	5.8	17.5
FEM	MAE	(16,4,4)	64	0.0005	0.0004	0.00128	1.4	4.5
	MAE	(16,4,4)	32	0.0004	0.0005	0.00118	3.0	9
PDR	PDR	(256,1,1)	64	0.0002	0.0002	0.00160	3.2	10.5
	PDR	(256,1,1)	32	0.0002	0.0003	0.00199	3.1	17

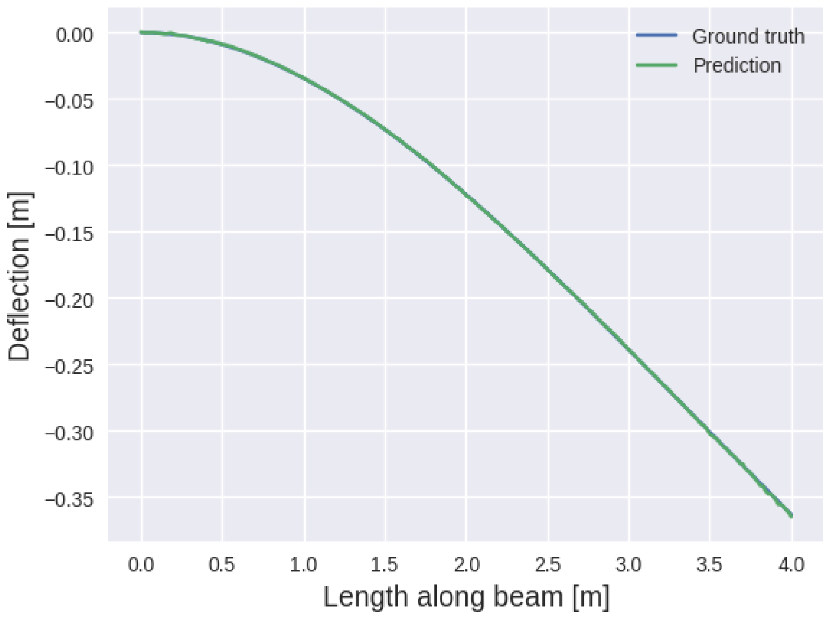
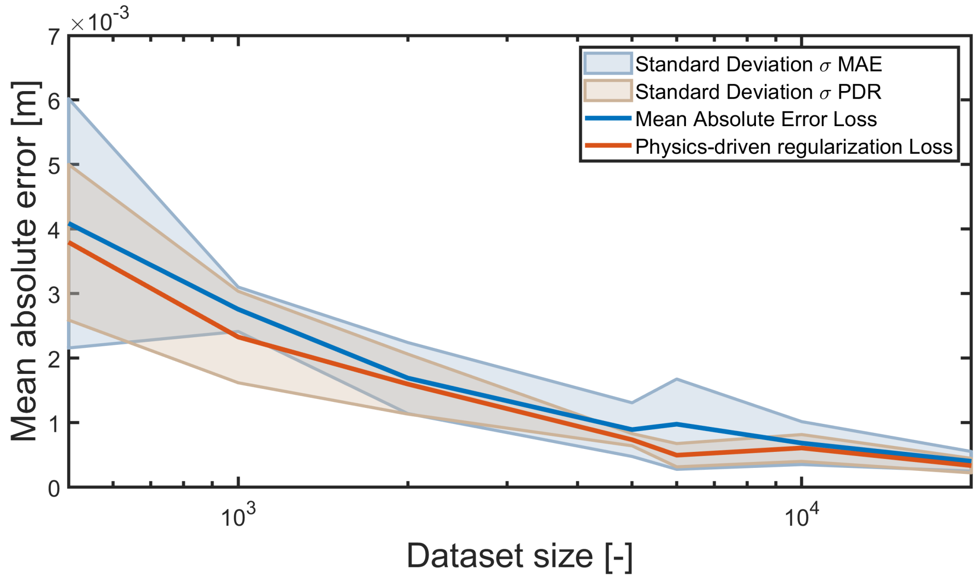
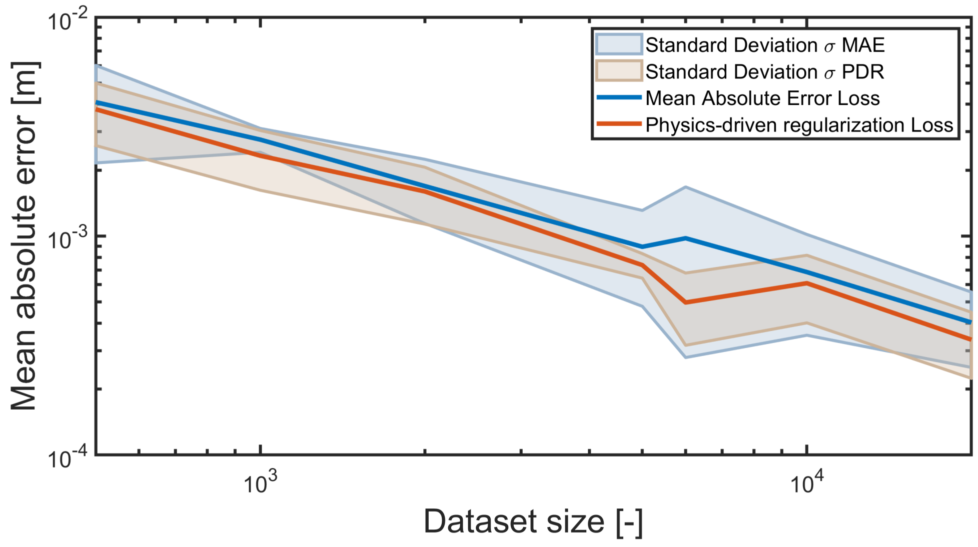


Figure 4.4: Prediction of the deflection $\hat{u}$, compared to the ground truth u for a single example in the test set for the experiment with physics-driven regularization and batch size 64.

A study to the effects of the data set size on the accuracy has been performed to see if the applied PDR is able to improve the model regarding the need for sufficient train data. Both the original loss function as the regularized loss function were tested for multiple data set sizes; $N = [500, 1000, 2000, 5000, 6000, 10.000, 20.0000]$. Each set size is trained at least 6 times for each loss. Figure B.8 depicts the mean error over all repetitions $\pm$ its standard deviation for each data set size. Both figures 4.5a and 4.5b show the same information, but the error of the latter is plotted on a logarithmic scale for a better aspect. It is clear to see that the regularized loss function seems to perform better on all data set sizes. This entails that for the same accuracy, the data set could be smaller. This is a nice improvement, since it is not very common to generate as much training data as desired. As with other training sessions though, multiple times the network was not able to converge within the learning scheme for both losses. The results in Figure B.8 would be too irregular if these outliers were to be included in the mean error. Instead, they were excluded from the mean and replaced with additional repetitions. It is clear that there is a lot of coincidence involved due to the stochastic nature of the model. It is therefore of important to perform ample training sessions to guarantee reproducibility of the results.



(a) Linear scaling of the accuracy



(b) Logarithmic scaling of the accuracy

Figure 4.5: Accuracy resulting from various data set sizes compared for an unregularized and regularized model.

4.2.3 Discussion

The proposed method of PDR seems to be improving the accuracy, training time and data dependency of the network. The mean error was decreased by a factor 3 within the same amount of epochs without major trade-offs regarding the train and test time. These results seem to agree with the expected outcome of the experiment, since the solution to the problem is implemented in the network completely, meaning that the network must converge to the true solution.

However, in most scenarios the solution to the problem is not fully known, and requires a more general added term. It seems interesting to experiment with a regularization term that only partly matches with the desired solution. An approach to such an computational experiment is explained in the following section.

4.3 Partial physics driven regularization

In this section, two different approaches are considered to experiment with partly matching regularization. The first approach uses a regularization term that describes a more general physical phenomena that is true for the problem but could also be used for other problems. Examples of such terms are the use of boundary conditions, material laws or a more general rule that describes the shape of the deflection. The other approach is to use the same regulalizer as before, but to introduce it to input data that does not fully match its solution.

4.3.1 Experiment: Boundary Conditions

Fortunately, it is rather straightforward to construct a partially matching term from the full solution, meaning that no new theory needs to be introduced. Instead of using the residual of the beam equation as term in the regularizer, the known boundary conditions are used directly as input. Since PDR only steers the network based on a single scalar value, there is much freedom on which BC's to combine and how much they weigh in to the loss; including the freedom to specify which nodal values to include. From the problem is known that the deflection of the beam is equal to 0 at the wall.

$$u(x = 0) = 0 \tag{4.3.1}$$

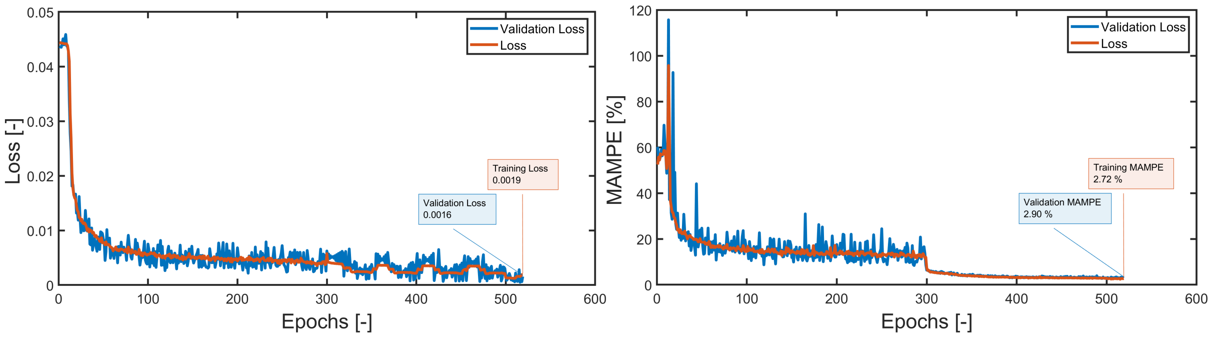
Since this value is already supposed to be zero, it could be immediately added as regularization term to the loss function, which is then defined as

$$\text{Loss} = \text{MAE} + \lambda \hat{u}(0) \tag{4.3.2}$$

Training, results and discussion

Since there is full certainty that this regularization term should be equal to zero for all samples, it was decided to experiment with a large weight; $\lambda = 100$. The results during training however, seem to vary tremendously for each experiment. The regularizer seems to perform quite well when the network was already converged neatly in the first 300 epochs, but not so well when this was not the case. It was decided to perform a total of 12 repetitions for this experiment, to get a better understanding of the effects of the regularizer. Figure 4.6 shows the course of the loss over a single training.

The loss seems to fluctuate repeatedly once the regularizer is introduced, while the MAMPE is hardly affected. A potential reason for this behaviour could be that the regularizer only directly influences a single neuron, while the rest of the loss is based on the average over all neurons. It could occur that the model predicts certain solutions throughout the training that are more in favor to one of both terms. Another potential reason could be that the focus on a single heavy weighted neuron could disrupt the convergence of all other neurons, meaning that that the total loss will never properly converge. The average results are added to table 4.3 and show no improvements compared to previous experiments. Using a regularization term that affects a larger domain of the network would probably yield clearer results. Further research is needed for a better understanding on this topic.



(a) Loss (b) MAMPE

Figure 4.6: The course of the training and validation metrics for the experiment with boundary conditions as regularizer. All training sessions are performed with the hyperparameters shown in the left column of Table 3.3 and the learning scheme shown in Table 4.1

Table 4.3: Intermediate results for the experiment with a boundary condition as regularizer.

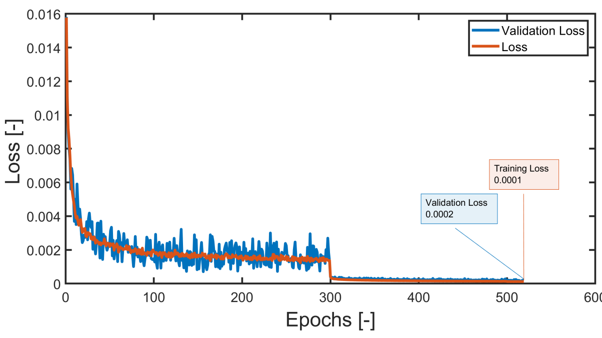
Method	Loss	Input shape	Batch size	$\bar{e}$ [m]	$\sigma(e)$ [m]	α	Prediction time [ms]	Train time [min]
Mendizabal (2020)	MAE	(16,4,4)	4	0.0007	0.0006	0.00352	2.5	35
Analytical	MAE	(256,1,1)	64	0.0006	0.0007	0.00511	2.9	9.5
	MAE	(256,1,1)	32	0.0002	0.0002	0.00102	5.8	17.5
FEM	MAE	(16,4,4)	64	0.0005	0.0004	0.00128	1.4	4.5
	MAE	(16,4,4)	32	0.0004	0.0005	0.00118	3.0	9
PDR	PDR	(256,1,1)	64	0.0002	0.0002	0.00160	3.2	10.5
	PDR	(256,1,1)	32	0.0002	0.0003	0.00199	3.1	17
PDR, u(0)=0	PDR	(256,1,1)	64	0.0045	0.0112	0.00344	2.9	9.5

4.3.2 Experiment: Guided cantilever beam

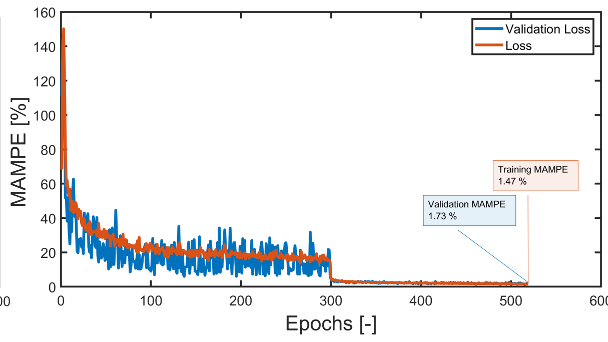
The goal of this experiment is to see how well the model is able to deal with inconsistencies between the regularizer and the given train data. Instead of altering the regularization term again, it was decided to alter the input data. This was done by generating solutions for a cantilever beam that is guided at the tip, instead of being free. The analytical solution to this problem is presented in Section 2.1.2. This deflection broadly resembles the previous deflection since they are both derived from the same equation and are described by a 3^{rd} order polynomial. They however also differ in multiple ways. For example, the overall deflection of the GCB is smaller for the same input force and the slope at the tip is equal to zero. The regularizer is the same as described in equation 4.1.8.

Training, results and discussion

It is clear to see from figure 4.7 that the unregularized network is able to extract the deflection behaviour from the new input data without problems. The regularized network however, does show some problems. The loss in Figure 4.8a shows a large increase when the regularizer is introduced, which is normal, but the loss does not seem to decrease much after that; resulting in loss and MAMPE values at the end of training that are significantly larger than before. This behaviour is confirmed by the results acquired by testing; which are added to Table 4.4. The accuracy of the regularized network is approximately a factor 7 larger than without regularization; the prediction time is also doubled.

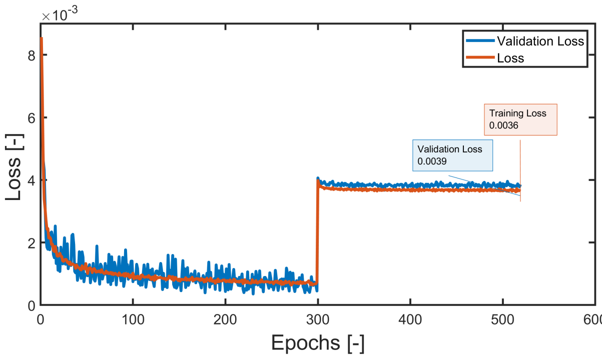


(a) Loss

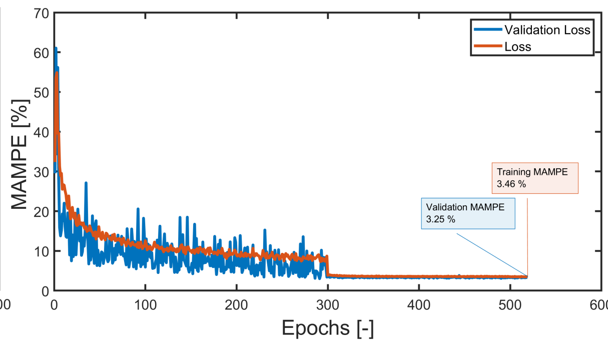


(b) MAMPE

Figure 4.7: The course of the training and validation metrics for the GCB experiment with MAE loss



(a) Loss



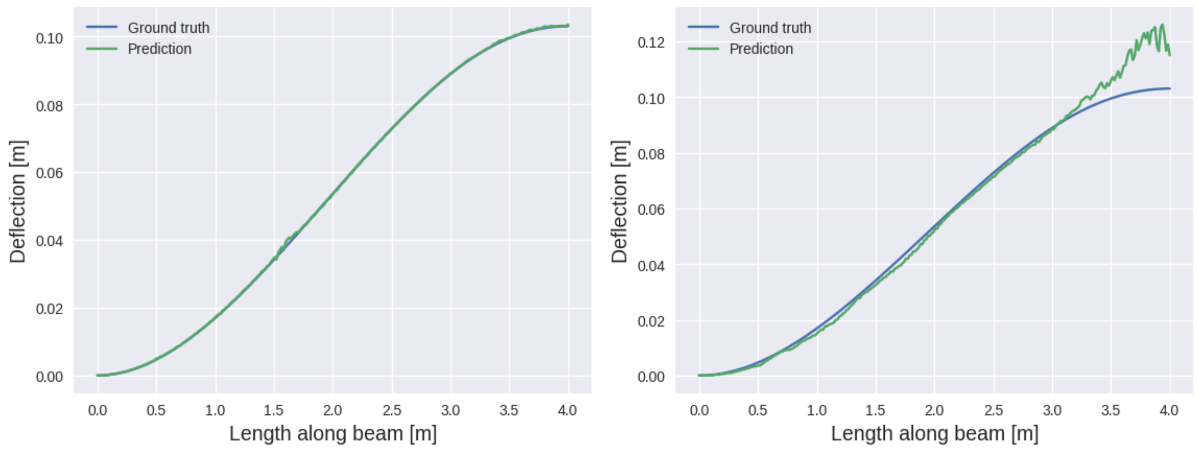
(b) MAMPE

Figure 4.8: The course of the training and validation metrics for the GCB experiment with regularized loss

Table 4.4: Overview of all results trough computational experimentation.

Method	Loss	Input shape	Batch size	$\bar{e}$ [m]	$\sigma(e)$ [m]	α	Prediction time [ms]	Train time [min]
Mendizabal (2020)	MAE	(16,4,4)	4	0.0007	0.0006	0.00352	2.5	35
Analytical	MAE	(256,1,1)	64	0.0006	0.0007	0.00511	2.9	9.5
	MAE	(256,1,1)	32	0.0002	0.0002	0.00102	5.8	17.5
FEM	MAE	(16,4,4)	64	0.0005	0.0004	0.00128	1.4	4.5
	MAE	(16,4,4)	32	0.0004	0.0005	0.00118	3.0	9
PDR	PDR	(256,1,1)	64	0.0002	0.0002	0.00160	3.2	10.5
	PDR	(256,1,1)	32	0.0002	0.0003	0.00199	3.1	17
PDR, u(0)=0	PDR	(256,1,1)	64	0.0045	0.0112	0.00344	2.9	9.5
GCB	MAE	(256,1,1)	64	0.0002	0.0002	0.00531	3.0	10
	PDR	(256,1,1)	64	0.0014	0.0018	0.00358	6.2	11

The decline in performance is confirmed visually in the prediction of Figure 4.9b. The predicted deflection largely follows the true deflection but deviates at the tip, following the shape of a free cantilever beam. That is because the regularizer is derived in such a way that it enforces to fit the deflection of a free cantilever beam over the predicted deflection of the guided cantilever beam. This phenomenon is depicted visually in Figure 4.10, where the black line represents a fabricated possible prediction of the true deflection in blue. The red line represents the smoothed fit of the prediction, and clearly shows that its shape does not correspond to the true deflection. Both the MAE loss as the regularizer try to steer the prediction towards a different solution. Instead of aiding each other, they are in a tug of war.



(a) MAE loss (b) PDR

Figure 4.9: Prediction of the deflection $\hat{u}$, compared to the ground truth u for a single example in the test set for the experiment with PDR and a GCB data set

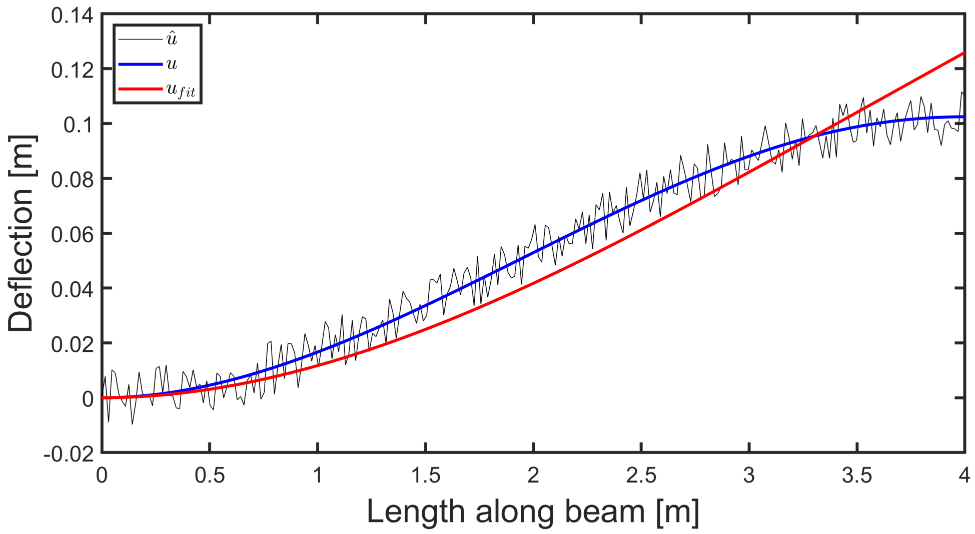


Figure 4.10: Enforced fitting of free cantilever beam deflection over guided cantilever beam predictions resulting in a mismatch

In this specific case it is not an improvement to include PDR that does not fully agree to the input data. However, that does not mean that it will never work. For example, it could still be case that the regularizer aids the network in earlier stages of training by providing a general shape that somewhat resembles the desired solution. It could also be the case that perhaps certain neurons are really benefitted by this regularizer; for example the node at the wall still has a correct description by the regularizer. Further research should be conducted to better understand the influence of various cases where the regularization does not fully agree with the presented problem.

5. Discussion

5.1 Part I: Development of the model

The first objective of this thesis was to evaluate a machine learning based approach that could aid or replace FEM in an accurate manner at real-time rates. This objective originates from the desire for the development of digital twin models of medical devices, such as catheters and guidewires, to aid medical treatments. The machine learning method under investigation is based on the U-mesh framework developed by Mendizabal (2020), who have defined a U-net like convolutional neural network that acts as surrogate for FEM.

A model was developed and evaluated on a simple problem concerning the elastic deformation of a cantilever beam. The first computational experiments were performed on analytically generated input data and showed that the model was able to make predictions with accuracies and prediction times within an acceptable range that is comparable to the model of Mendizabal (2020). The subsequent experiments were performed on a data set that was generated with FEM which showed that the model was also able to operate with 3-D geometries. Both the accuracy and the prediction time were improved significantly compared to the previous experiment, but the reason for this improvement is not yet fully understood.

Despite the promising results of the experiments, there are some limitations worth mentioning. Many choices regarding the development of the model have been made to replicate the work of Mendizabal (2020) as closely as possible for easy comparison. This would, however, sometimes result in less ideal design choices, such as the usage of point loads. This often leads to unrealistic and discontinuous result, since a point load is an description of a force over an infinitesimal small area. Instead, a distribution of loads could be used for continuous results.

The proposed method did have major improvements in computation time compared to FEM. However, it should be noted that the measurements of the speed up could be performed a lot fairer. The hardware used for both methods should be identical. It would be better to separately keep track of wall time and processing since the high communication costs are currently unaccounted for.

In later stages of the project, the model sometimes tended to have trouble with converging. It is not clear why this behaviour occurs, but it is probably due to a small bug in the source code that could not be found. Due to the varying outcomes that result from this behaviour, it is crucial to perform many repetitions of an experiment and validate each result to guarantee reproducability. It is also recommended to use a random seed for better reproducability.

5.2 Part II: Physics-Driven Regularization

The second objective of this thesis was to propose and research a possible hybrid modelling method as addition to the earlier developed model. This objective originates from the idea that for most physics-related applications, the already known physical properties could be used to a certain extent, to aid the model in various ways. The proposed method is to add physics-driven regularization (Nabian and Meidani, 2020) to the current model as it is a simple addition that does not interfere with the structure of the model, but gives room to explore various aspects of hybrid modelling. In the first computational experiment, the residual form of the beam equation was used as regularization term as proof of concept.

This experiment showed promising improvements on the accuracy, convergence rate and data dependency compared to the unregularized loss function, without major trade-offs in computational cost. A major caveat to this experiment was that the full solution to the problem was given to the regularizer, which is in most applications not a very likely scenario.

Two tests were performed with regularization terms that only partly match the desired solution of the model. In the first test a regularization term was used that describes a more general physical aspect, and in the second test a small mismatch was created between the training data and regularizer. Both tests seemed not to improve the model directly, but left some room open for further research.

5.3 Suggestions for future research

It would be interesting to research the models ability to extract the deformation behaviour of (slightly) more complex problems. Some suggestions for extensions are: large strain deformations, non-linear materials, plasticity, contact problems, complex geometries, anisotropic materials and laminates.

Another topic to consider is the development of a robust and efficient method to translate the acquired FEM results to an input format that the neural network is able to use. The work of Mendizabal (2020) mainly focusses on this aspect, since it is a crucial step to do properly. For the problems described in this thesis it was easy to directly translate the output to input, but this changes quickly for more complex geometries that are very common in health technology related topics.

It would be interesting to conduct further research into other unique approaches for deriving a physics-driven regularization term since PDR offers the freedom for almost any approach.

An interesting addition would be to include meta-data (e.g. varying stiffness) in the train data in order to teach the network a more general description of the problem such as the pure constitutive behaviour. In this way a trained model could be used in far more general setting to make predictions for given input variables much like FEM.

Instead of using a steering method such as PDR as hybrid modelling approach, it would be interesting to use a method that constrains possible solutions such as the physics-informed neural network (Raissi et al., 2019). PINN's are developed for solving differential equations, which is precisely what is needed for many mechanics problems.

A. Methods

A.1 Validation of applied numerical methods

The results of the FEM simulation were validated against the analytical solution. Two point loads of -0.5N were placed at equidistance to the left and to the right of the centroid of the beam. This cancels out the majority of all torsion components along the length of the beam, ensuring a deflection that is mostly in the y direction. Comparison between the analytical and numerical method shows that both deflections are almost the same, apart from a local difference at the location of the point load. This local difference comes from the fact that the numerical method has trouble with the definition of a point load. The global deflection is overall the same as with the analytical method and differs roughly 1.5% from the analytical result.

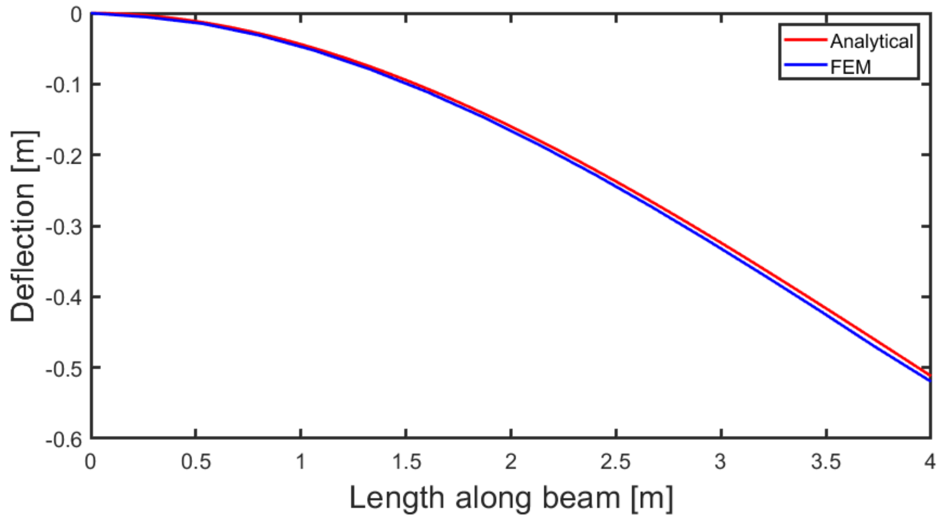


Figure A.1: Visual representation of activation function in a single neuron

A.2 Activations

Figure A.2 shows an example of the activation of a single hidden neuron (yellow) with two inputs (blue) and using the ReLU activation function following the calculation below. The output $h(x)$ is calculated according Equation 2.3.1; $h(x) = \text{ReLU}\,(0.74 \cdot 4.3 + 1.63 \cdot 6.2 - 5.2) = 8.09$

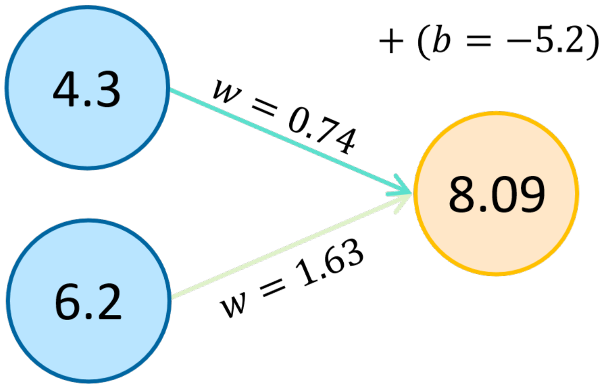
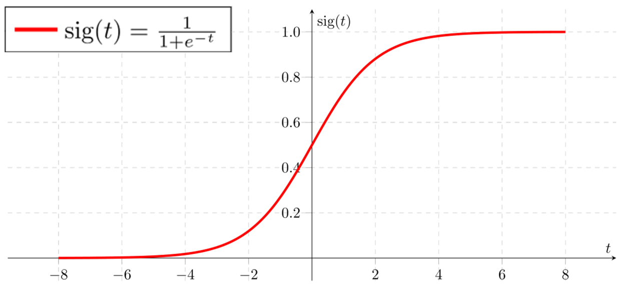
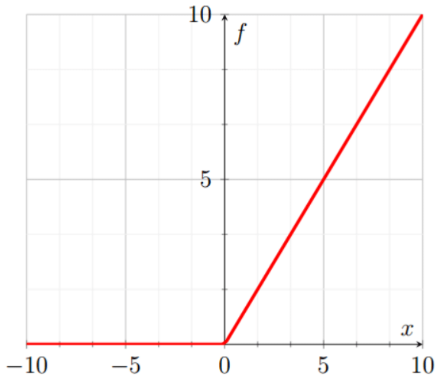


Figure A.2: Visual representation of activation function in a single neuron



(a) Sigmoid activation function



(b) ReLU activation function

Figure A.3: Visual representation of activation functions

A.3 Layer operations

In the following sections some explanations are given on the various layer operations that are used in our model. The examples that are given however, are for two-dimensional cases since they are far more common in the field of interest where convolution is used.

Dense layer

A dense layer is just another name for a fully connected layer, meaning that all input neurons are connected to a chosen number of output neurons.

Concatenate layer

A concatenate layer joins two input feature maps together over a chosen axis.

Convolutional layers

A convolutional layer extracts a feature map by parsing a window of weights over the input space. This window is known as the kernel (or filter) and is represented as a matrix of weights of a certain chosen size. Using this kernel, a convolutional layer is able to extract local features. The kernel parses over the input with a chosen step size, known as the stride s, and for each step the convolution between the kernel and the corresponding input patch is computed by element-wise multiplication. This is repeated until the whole input space is parsed. The shape of the resulting feature map is determined by the stride and the shape of the input space. Convolution tends to decrease the size of the feature map, since the kernel cannot parse over input values at the edge of the input space. This can be controlled by padding a layer of zero's around the input space.

Figure A.4 depicts a visual representation of a single convolution step for a zero-padded input space of 3 channels, having each a kernel of 3x3. Each kernel has a unique combination of weights and is able to extract its own salient features from the input space. It is also possible to have multiple kernels for each channel. Each filter still corresponds to its own unique feature map, since the weights are all also uniquely initialized. It is common practice to use many kernels at the start of the network to extract various features early on so that the input data does not need much pre-processing.

Even though this layer seems complex compared to a conventional (fully connected) layer, there is still a major improvement in training time. This is because a convolutional layer uses the same weights of the kernel for the full input space, which is most of the time an enormous reduction compared to a fully connected layer. For comparison: a single convolutional layer with 64 kernels of size (3,3,3) for an input size of 16x4x4 still only has roughly 2.000 trainable parameters, while a fully connected network has approximately 65.000 trainable parameters.

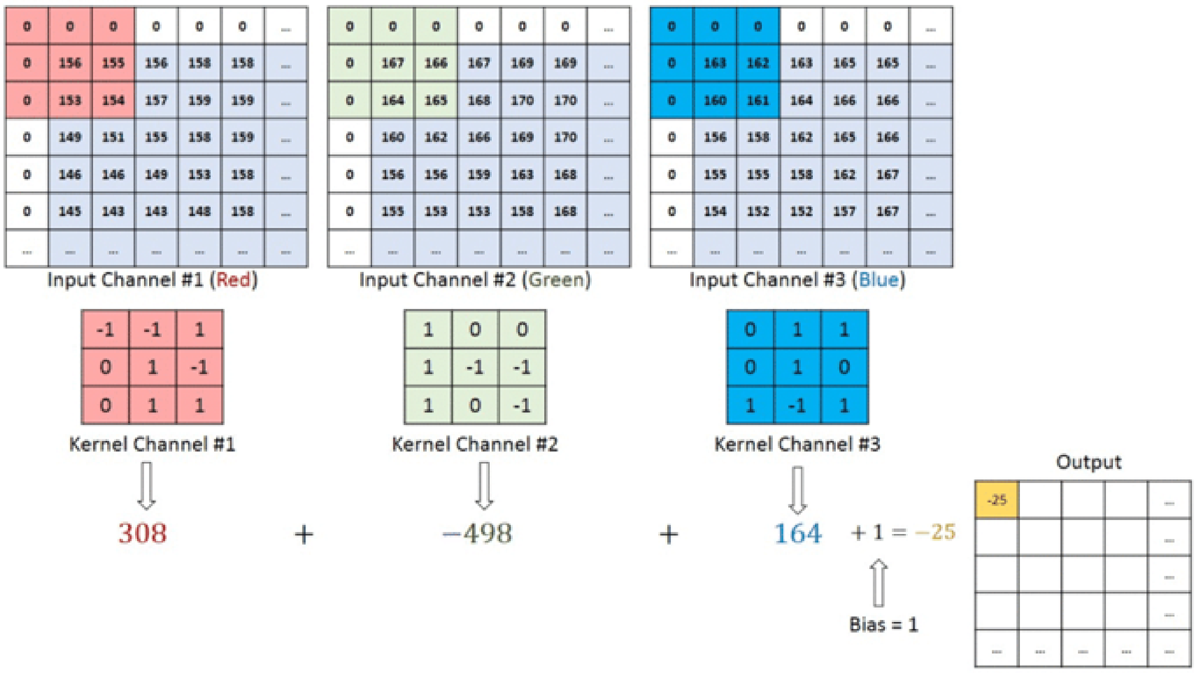


Figure A.4: Visual representation of a convolutional layer (`www.anderfernandez.com`)

Pooling layers

A pooling layer decreases the spatial dimension of the convolved features by calculating the mean or maximum (mean-pooling or max-pooling) on patches of the input (Figure A.5). This step forces to extract the most dominant features from the input feature map. This is a very important property of the pooling layer, and is the backbone for auto-encoder like architectures. Pooling layers also have a size and a stride that combined determine the shape of the output feature map.

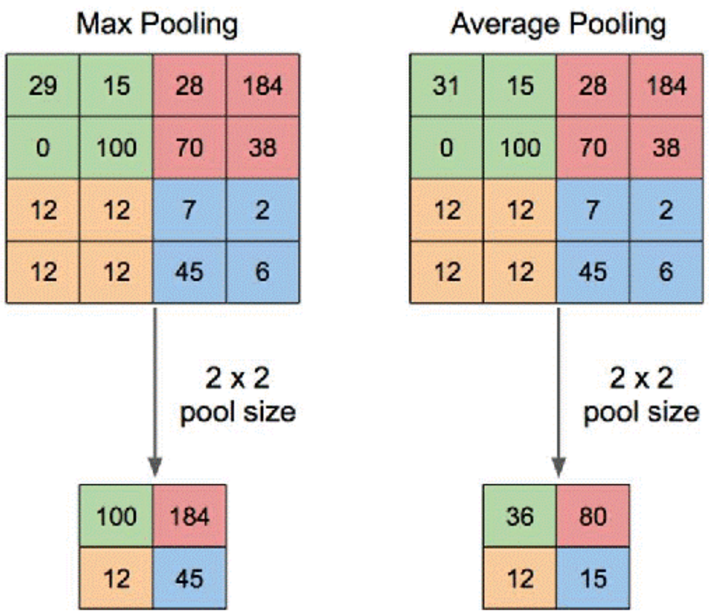


Figure A.5: Visual representation of max pooling and average pooling layer (?)

Up-sampling layers

Up-sampling layers are used to increase the size of the feature map back to the desired output resolution, which is equal to the input size in the case of a U-mesh architecture. Transposed convolution (Figure A.6) is a common upsampling method that also uses a kernel with weights. The desired feature map size is again determined by choosing an appropriate kernel size in combination with the stride.

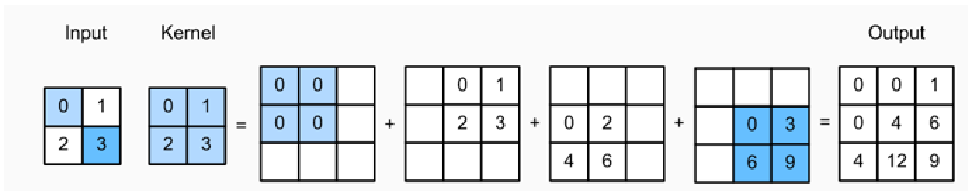


Figure A.6: Visual representation of a transposed convolution layer (`www.towardsdatascience.com`)

A.4 Optimizing

An optimizer is an algorithm that is able to minimalize the loss in an iterative process. The loss is a function of all networks parameters which entails that there is a minimum for a certain combination of parameter values. To find such a minimum, the loss makes use of a gradient descent step. To explain this more intuitively, we assume a loss that is dependent on a single parameter, as plotted in Figure A.7. This parameter is randomly initialized, resulting in a starting point on the function. The optimizer then calculates the gradient of the loss over this parameter through backpropagation, and alters the parameter in the direction of the steepest gradient. This is repeated multiple times until the loss has reached a minimum. It is important to note that this method is only able to find local minimums since it does not know whether the steepest gradient also leads to the global minimum of the loss. This also depends on the starting point of the loss, which shows the importance of stochastic processes such as initialization, random input generation and shuffling of the train set. This process works the same for a function of multiple variables.

The rate at which the optimizers steps towards the local minimum is known as the learning rate and is set by the network architect. A learning rate that is too low takes forever to converge, while a too high learning rate constantly overshoots the local minimum. An ideal learning rate is somewhere in between, and preferably decreases as the minimum is approached (Figure A.8).

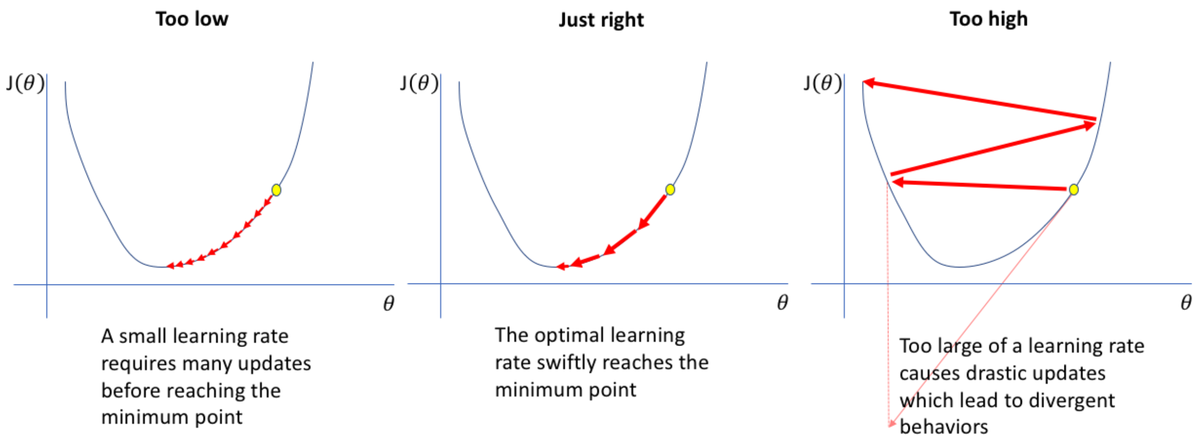

Too low
Just right
Too high
J(θ)
θ
A small learning rate requires many updates before reaching the minimum point
The optimal learning rate swiftly reaches the minimum point
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Figure A.7: Visual representation stochastic gradient descent optimization for various learning rates from: (`www.jeremyjordan.me`)

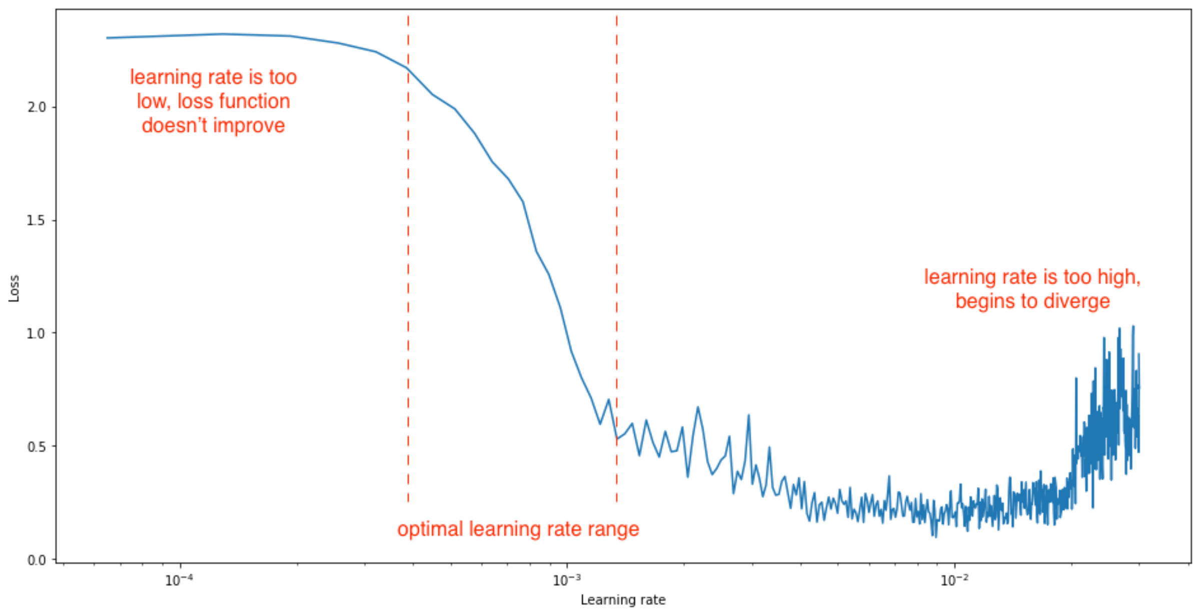


Figure A.8: Visual representation of the effect of selecting a proper learning rate from: (`www.jeremyjordan.me`)

A.5 Applied modelling approach

Notation rules

The following rules for mathematical notation are used throughout the thesis to distinguish various similar concepts.

Table A.1: Mathematical notations for `tensors` and tensors

Unit	Rule	Example
`n-D tensor`	Upright bold symbol with underline	$\underline{\mathbf{T}}$
`n-D tensor` component	Italic symbol with subscripts	$T_{i,\ldots,n}$
(Multi)-linear mapping tensor	Italic bold symbol	$\boldsymbol{T}$

Input Data

Figure B.8 presents the force matrix of the analytically generated data set of Section 2.1.1 visually in two different ways. Each each dot represents a sample. The x axis represents the location on the beam where the force is applied, a. The y axis of Figure B.4a represents the amplitude of the force, and the color represent the corresponding maximum deflections. This is the other way around for Figure B.4b, which gives a clearer view on how the varying force influences the end deflection.

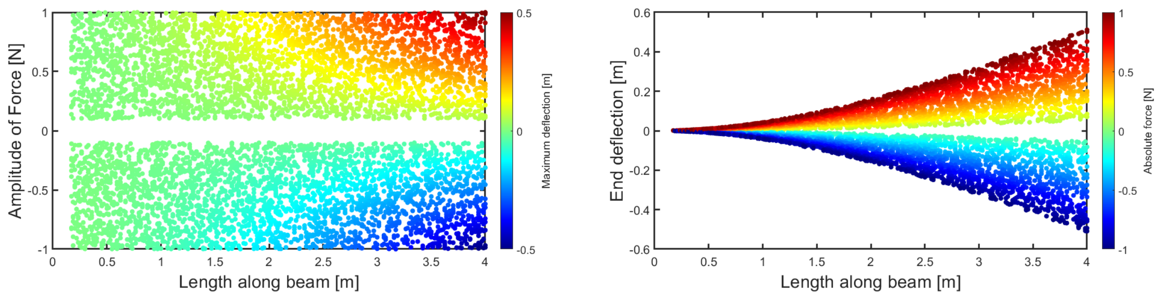


Figure A.9: Visual representations of the analytically generated data set. Each dot represents the deflection at tip of the beam for a point load at a location along the length of the beam. Two aspects are presented for a clearer overview

A.6 Applied methods

A.6.1 Model based on analytical input data

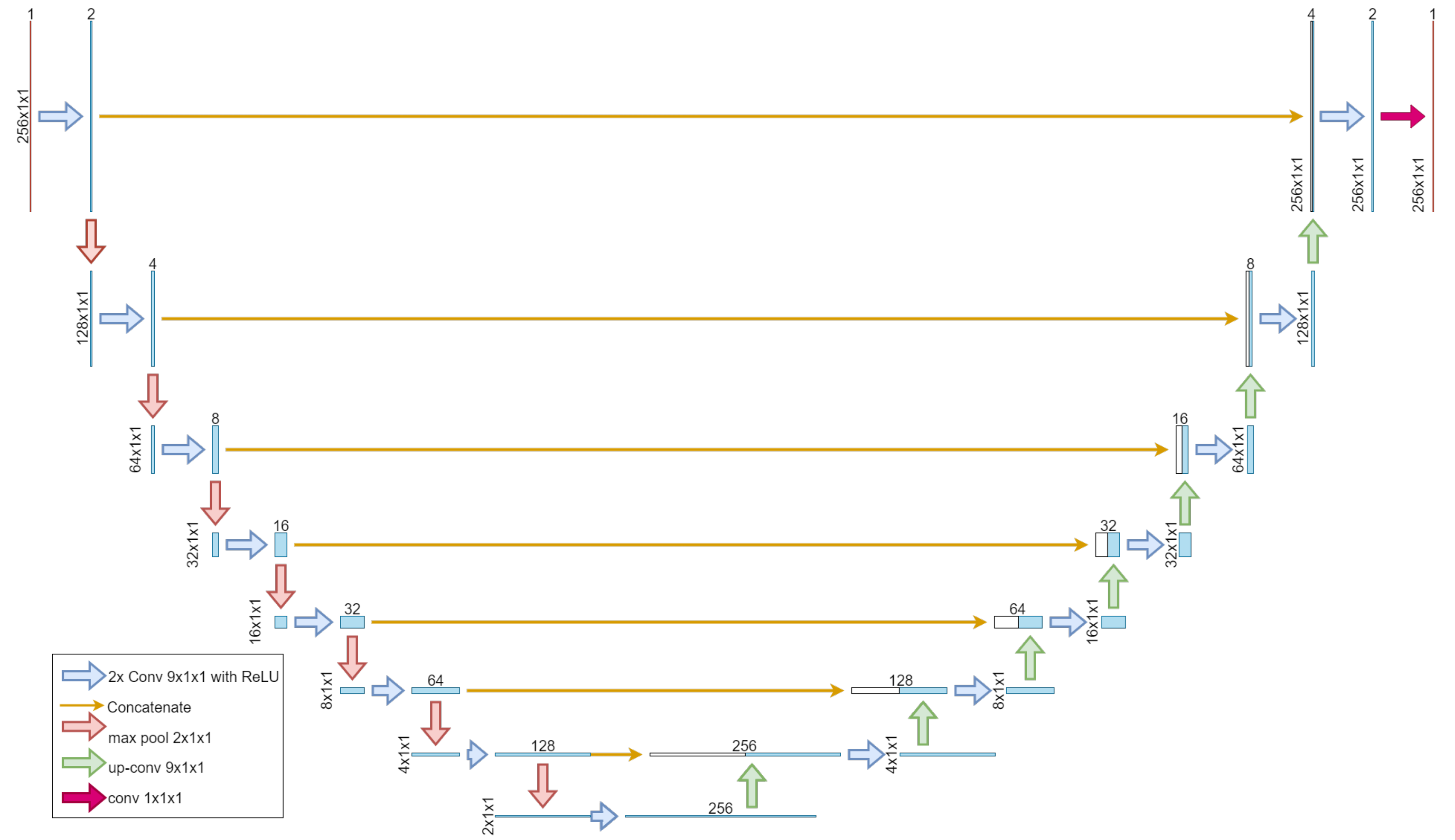


Figure A.10: Network architecture for the analytical generated data of size (256,1,1), with $c = 2$ channels in the first layer and $k = 7$ steps.

A.6.2 Model based on FEM input data

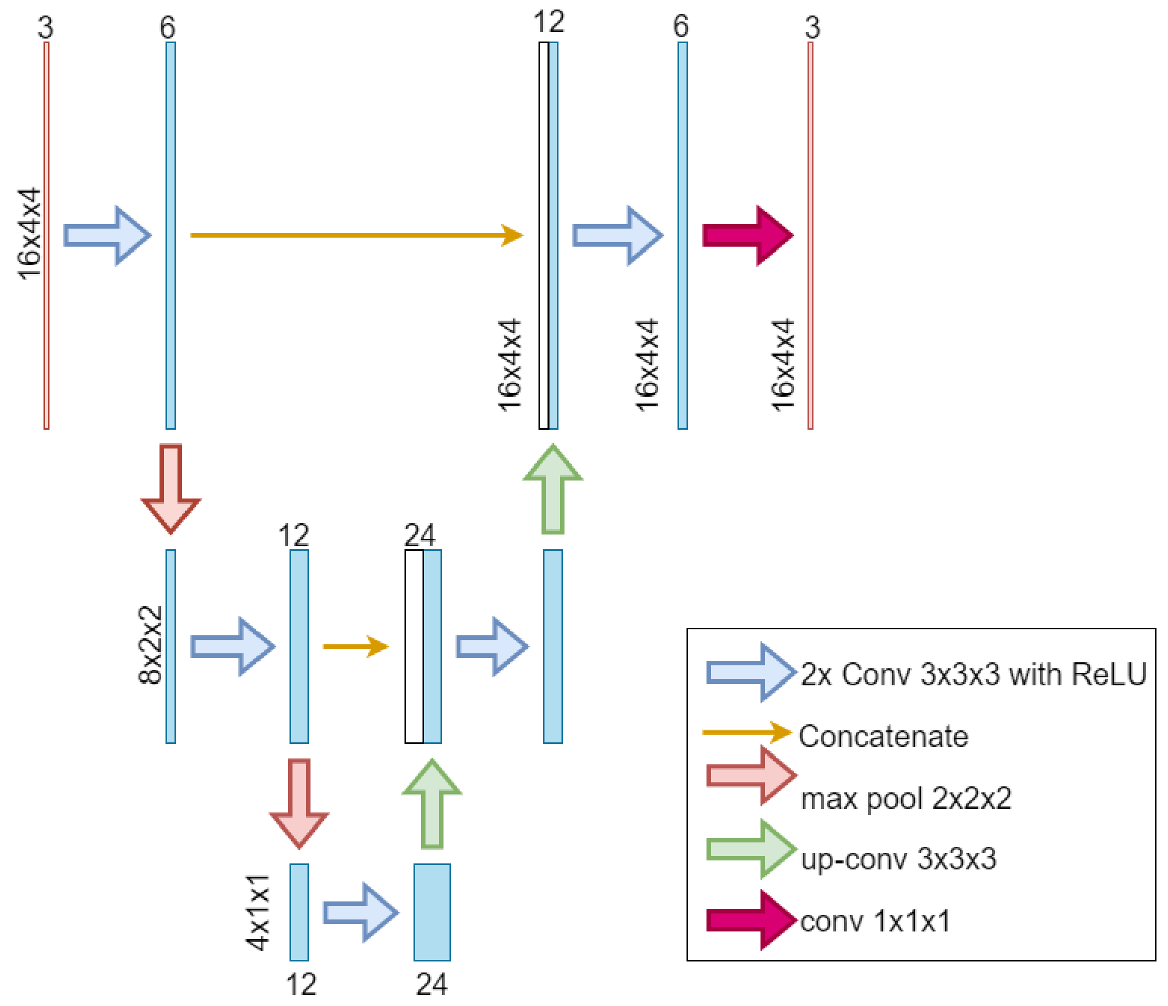
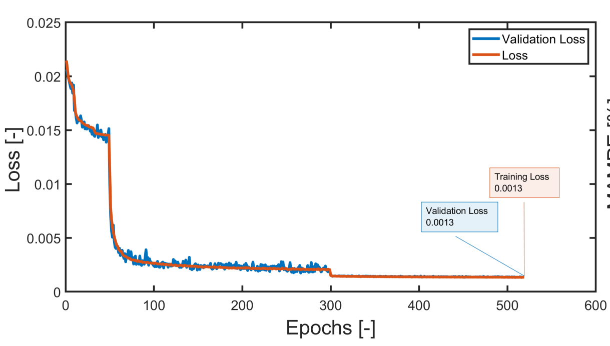


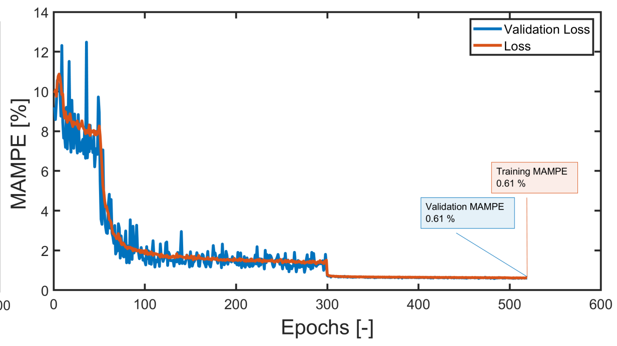
Figure A.11: Network architecture for the FEM generated data of size (16,4,4), with $c = 6$ channels in the first layer and $k = 2$ steps.

B. Additional Results

B.1 Experiment: FEM generated input data

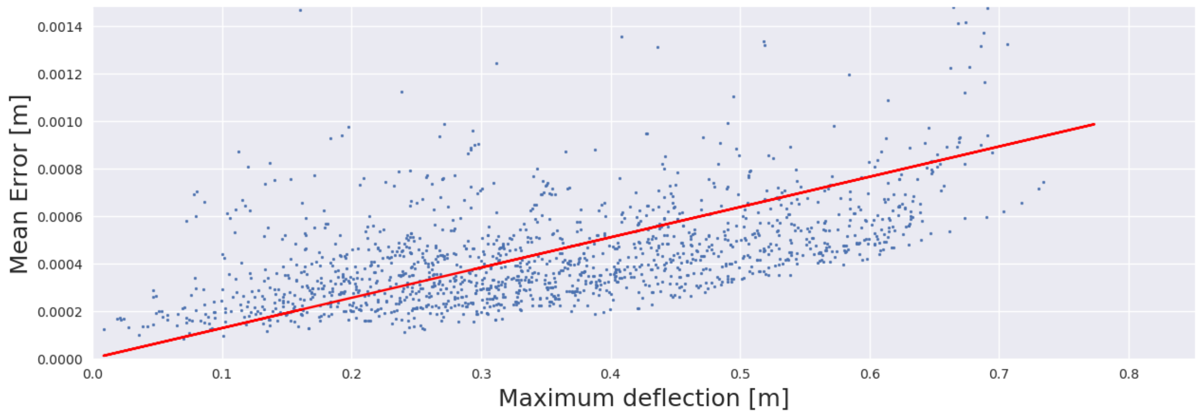


(a) Loss

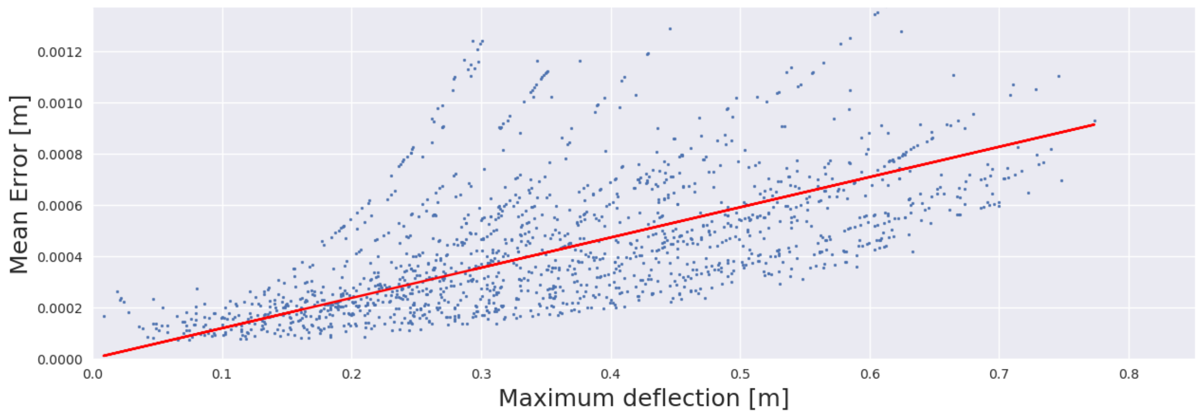


(b) MAMPE

Figure B.1: The course of the training and validation metrics for the experiment with FEM generated data with a batch size of 32. All training sessions are performed with the hyperparameters shown in the right column of Table 3.3



(a) Sensitivity for batch size 64, $\alpha = 0.00128$



(b) Sensitivity for batch size 32, $\alpha = 0.00118$

Figure B.2: Sensitivity analysis between the mean error and the maximum deflection for the experiment with fem generated data. Each dot represents a single sample in the test set. The red line represents a least squares linear fit through all points.

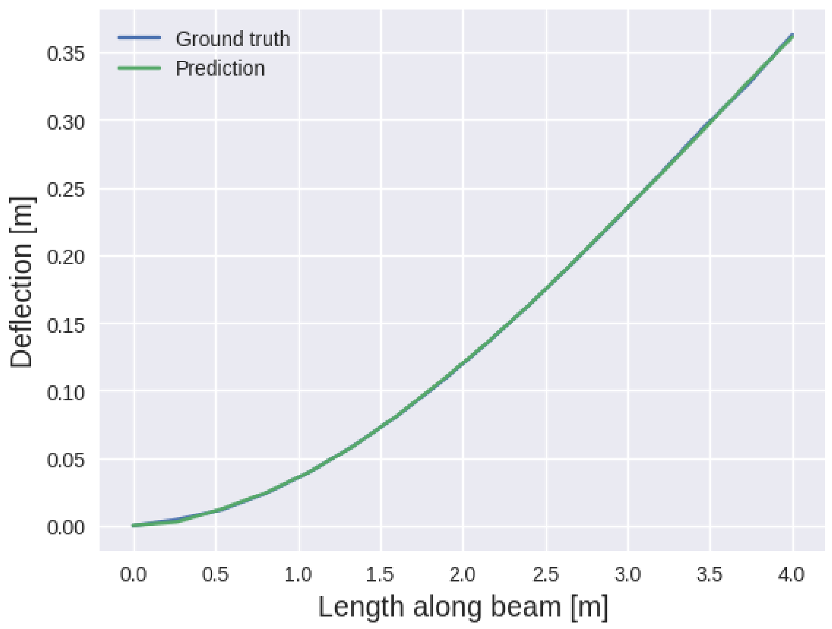
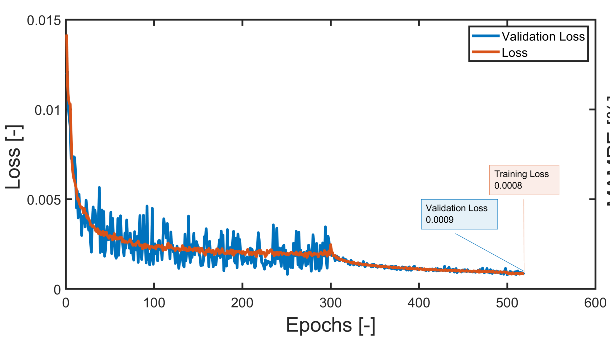
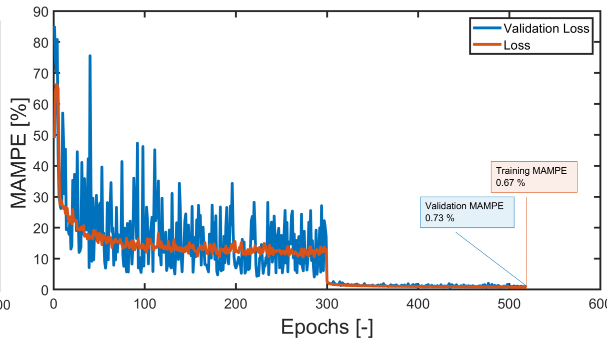


Figure B.3: Prediction of the deflection $\hat{u}$, compared to the ground truth u for a single example in the test set for the experiment with FEM generated data and batch size 32.

B.2 Experiment: Physics-driven regularization

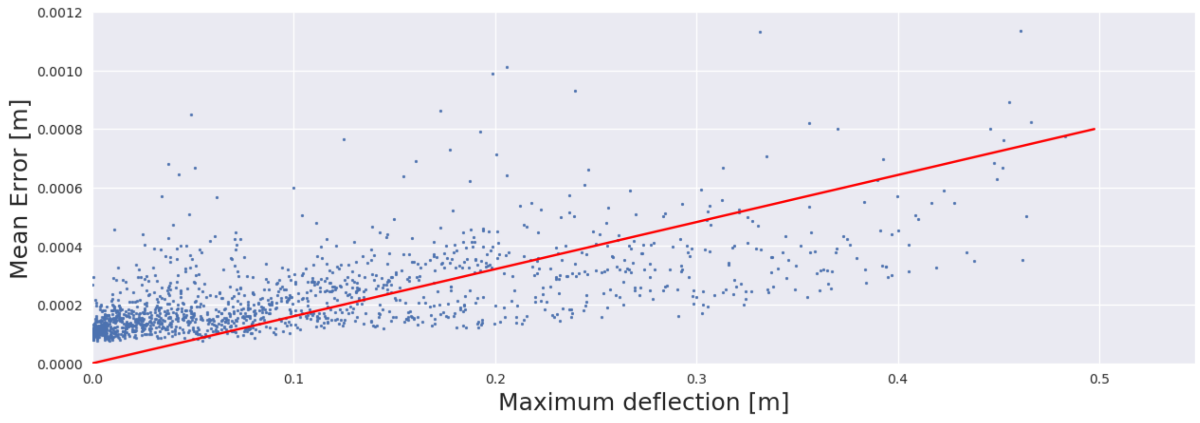


(a) Loss

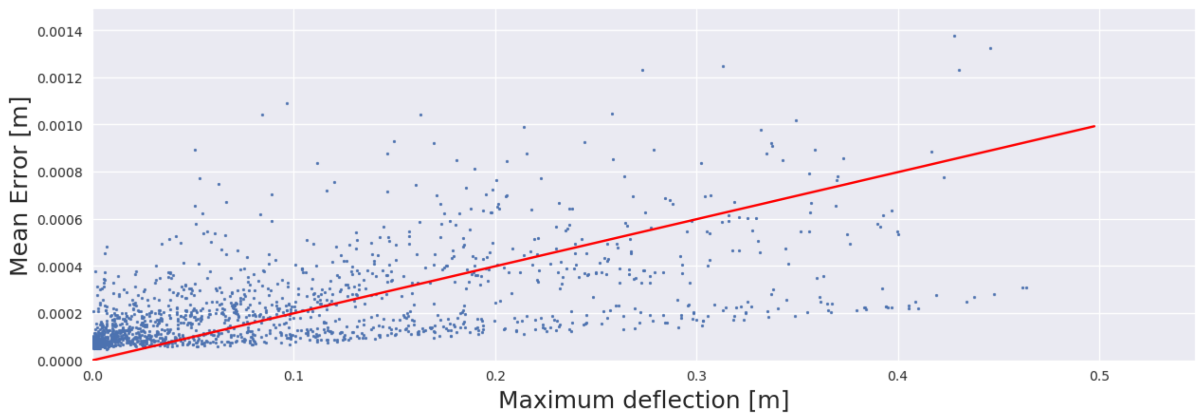


(b) MAMPE

Figure B.4: The course of the training and validation metrics for the experiment with physics-driven regularization with a batch size of 32. All training sessions are performed with the hyperparameters shown in the left column of Table 3.3 and the learning scheme shown in Table 4.1



(a) Sensitivity for batch size 64, $\alpha = 0.00160$



(b) Sensitivity for batch size 32, $\alpha = 0.00199$

Figure B.5: Sensitivity analysis between the mean error and the maximum deflection for the experiment with PDR. Each dot represents a single sample in the test set. The red line represents a least squares linear fit through all points.

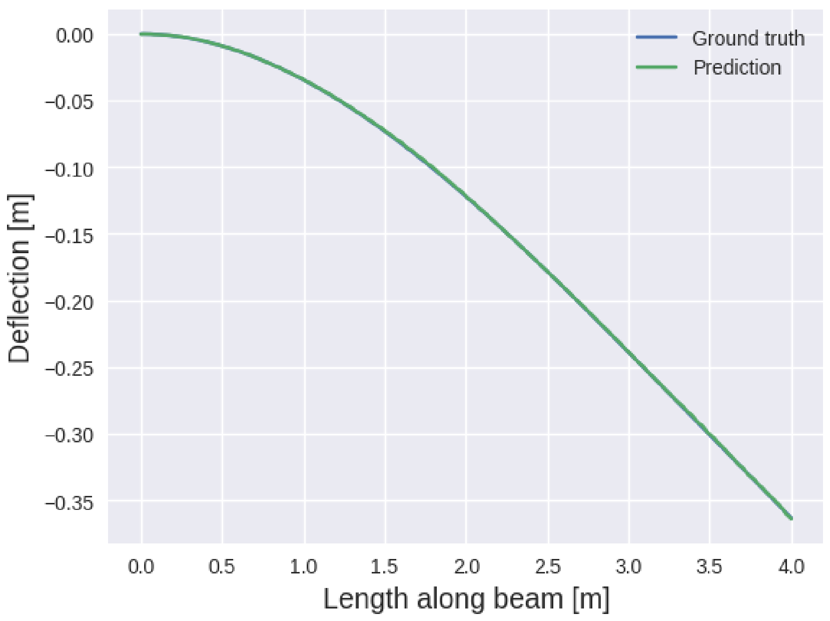


Figure B.6: Prediction of the deflection $\hat{u}$, compared to the ground truth u for a single example in the test set for the experiment with physics-driven regularization and batch size 32.

B.3 Experiment: Boundary conditions

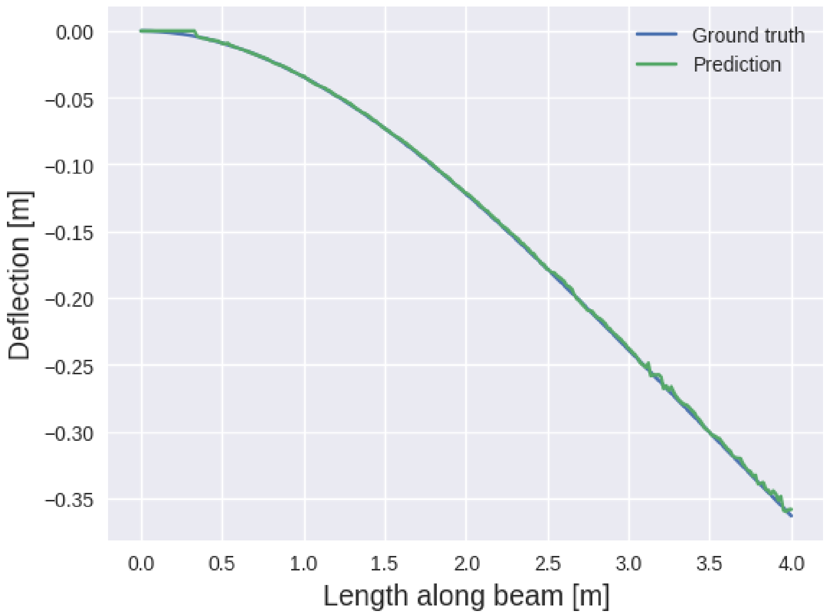


Figure B.7: Prediction of the deflection $\hat{u}$, compared to the ground truth u for a single example in the test set for the experiment with a boundary condition regularizer

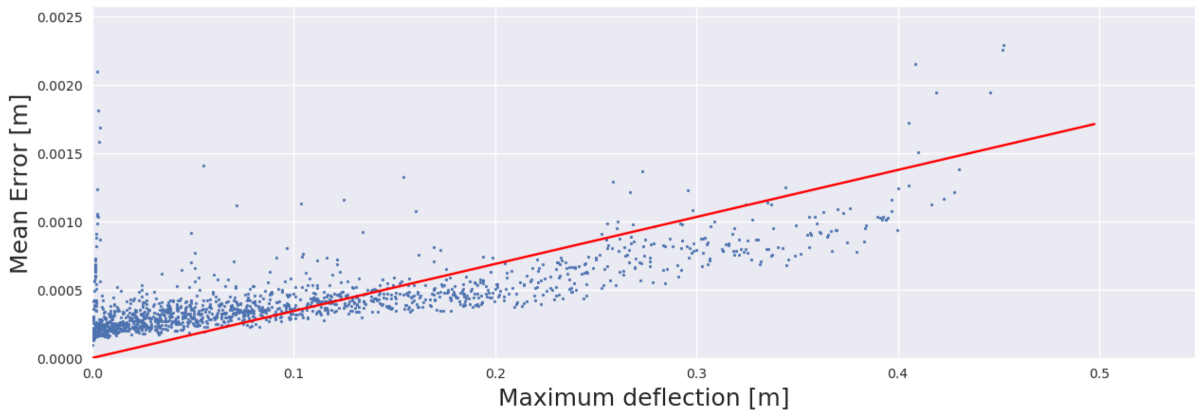
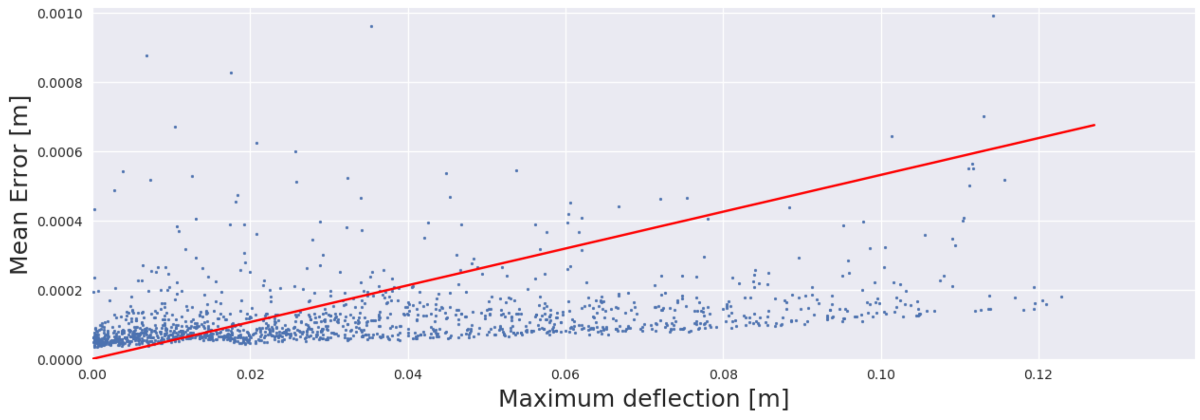
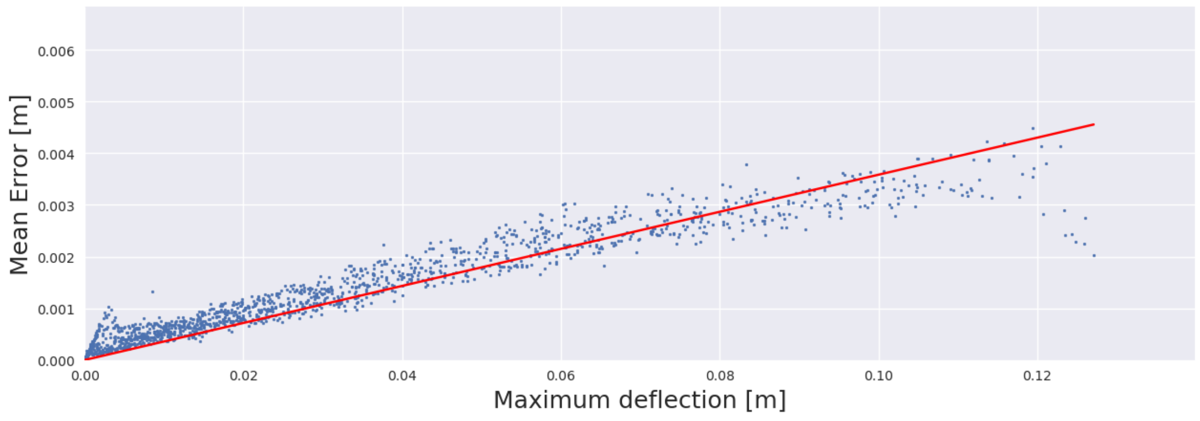


Figure B.8: Sensitivity analysis between the mean error and the maximum deflection for the experiment withboundary conditions as regularizer. Each dot represents a single sample in the test set. The red line represents a least squares linear fit through all points, with sensitivity $\alpha = 0.00344$

B.4 Experiment: Guided Cantilever beam



(a) Sensitivity for MAE loss, $\alpha = 0.00531$



(b) Sensitivity for PDR loss, $\alpha = 0.00358$

Figure B.9: Sensitivity analysis between the mean error and the maximum deflection for the GCB experiment. Each dot represents a single sample in the test set. The red line represents a least squares linear fit through all points.

C. Implementation of PDR

C.1 Derivation of u_{fit}

To derive a fit over the predicted deflection, once again the knowledge of the problem is used. It is known that the deflection of the beam could be described with a piecewise continuous function of a 3rd order and 1st order polynomial.

$$u_{fit} = \begin{cases} C_1 + C_2 x + C_3 x^2 + C_4 x^3, & x \leq a \\ C_5 + C_6 (x - a), & a \leq x \end{cases} \tag{C.1}$$

Following the diagram presented in Figure 2.4, some boundary conditions of this problem could be derived. The fist boundary conditions is at the clamp in the wall, which is defined as no deflection at x=0. From this boundary condition it logically follows that the first constant C_1 must be equal to zero.

$$u_{fit}(0) = 0 \Rightarrow C_1 = 0 \tag{C.2}$$

The second known boundary condition is that the slope of the beam is also equal to zero at the wall due to clamping. Filling this out in the first derivative of the piecewise function results in C_2 being equal to zero as well.

$$u'_{fit} = \begin{cases} C_2 + 2C_3 x + 3C_4 x^2, & x \leq a \\ C_6, & a \leq x \end{cases} \tag{C.3}$$

$$u'_{fit}(0) = 0 \Rightarrow C_2 = 0 \tag{C.4}$$

It is also known that the deflection and the slope described by both polynomials should be equal at the point load, resulting in

$$u_{fit}(a) = C_3 a^2 + C_4 a^3 = C_5 \tag{C.5}$$

and

$$u'_{fit}(a) = 2C_3 a + 3C_4 a^2 = C_6 \tag{C.6}$$

Finally it is known that the bending moment is zero from the point load to the tip of the beam, which is also confirmed by the second derivative of the piecewise function. From this knowledge, an expression for C_3 could be made in terms of C_4

$$u''_{fit} = \begin{cases} 2C_3 + 6C_4 x, & x \leq a \\ 0, & a \leq x \end{cases} \tag{C.7}$$

$$u''_{fit}(a) = 2C_3 + 6C_4 a = 0 \Rightarrow C = -3C_4 a \tag{C.8}$$

Substituting this result in Equations C.5 and C.6 results in expressions for C_5 and C_6 in terms of C_4

$$C_5 = -2C_4 a^3 \tag{C.9}$$

$$C_6 = -3C_4 a^2 \tag{C.10}$$

The function could thus be described with only one parameter C_4

$$u_{fit} = \begin{cases} C_4 x^2 \left(x - 3a\right), & x \leq a \\ C_4 a^2 \left(a - 3x\right), & a \leq x \end{cases} \tag{C.11}$$

The next step is to find the value for C_4 so that the function best fits to the noisy prediction. A least squares method is used to define the error R between the prediction of the deflection and the smoothed prediction.

$$R = \sum_{i=1}^{F} \left(u_{fit,i} - \hat{u}_i\right)^2 + \sum_{i=F}^{N} \left(u_{fit,i} - \hat{u}_i\right)^2 \tag{C.12}$$

Substituting Equation C.11 for the smoothed prediction yields

$$R = \sum_{i=1}^{F} \left(C_4 x_i^2 (x_i - 3a) - \hat{u}_i\right)^2 + \sum_{i=F}^{N} \left(C_4 a^2 (a - 3x_i) - \hat{u}_i\right)^2 \tag{C.13}$$

We know that the minimal value of R must be in a valley, which means that the derivative of R over C_4 must be equal to zero for the best fit

$$\frac{dR}{dC_4} = 0 \tag{C.14}$$

Deriving the derivative gives

$$\frac{dR}{dC_4} = \sum_{i=1}^{F} 2 \left(C_4 x_i^2 (x_i - 3a) - \hat{u}_i\right) \left(x_i^2 (x_i - 3a)\right) + \sum_{i=F}^{N} 2 \left(C_4 a^2 (a - 3x_i) - \hat{u}_i\right) \left(a^2 (a - 3x_i)\right) = 0 \tag{C.15}$$

Isolating C_4 results in

$$C_4 = \frac{\sum_{i=1}^{f} \hat{u}_i \left(x_i^2 \left(x_i - 3a\right)\right) + \sum_{i=f}^{N} \hat{u}_i \left(a^2 \left(a - 3x_i\right)\right)}{\sum_{i=1}^{f} \left(x_i^2 \left(x_i - 3a\right)\right)^2 + \sum_{i=f}^{N} \left(a^2 \left(a - 3x_i\right)\right)^2} \tag{C.16}$$

To conclude: substituting the value of C_4 from Equation **??** into the piecewise polynomial function of Equation C.11 yields the least squares fit of the predicted deflection $\hat{u}$ in the shape of a realistic deflection that is continuously differentiable.

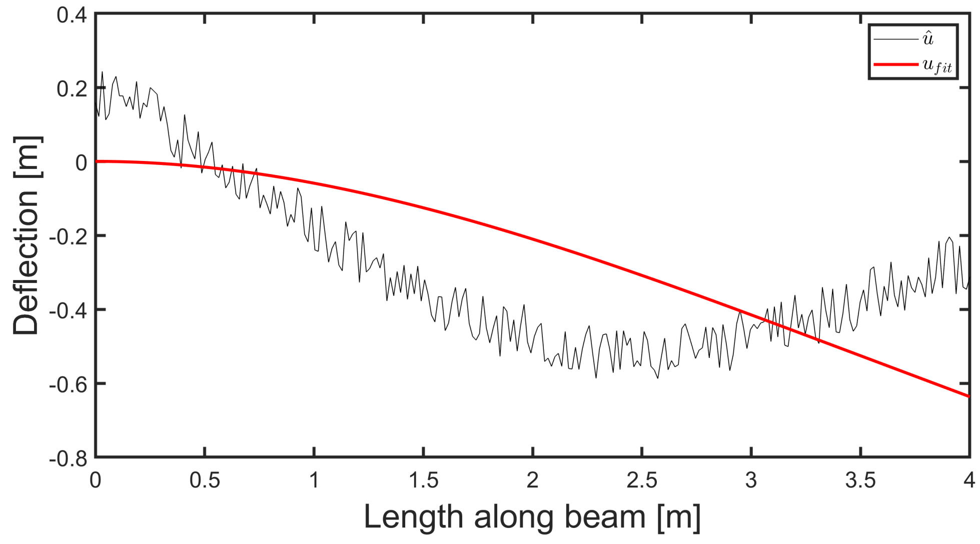


Figure C.1: Example is intermediate prediction

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., and others (2016). Tensorflow: A system for large-scale machine learning. In *12th Symposium on Operating Systems Design and Implementation*, pages 265–283.

Badarinath, P. V., Chierichetti, M., and Kakhki, F. D. (2021). A machine learning approach as a surrogate for a finite element analysis: Status of research and application to one dimensional systems. *Sensors*, 21(5):1–18.

Bhattacharjee, S. and Matouš, K. (2016). A nonlinear manifold-based reduced order model for multiscale analysis of heterogeneous hyperelastic materials. *Journal of Computational Physics*, 313:635–653.

Comas, O., Taylor, Z. A., Allard, J., Ourselin, S., Cotin, S., and Passenger, J. (2008). Efficient Nonlinear FEM for Soft Tissue Modelling and Its GPU Implementation within the Open Source Framework SOFA. In *Biomedical Simulation*, volume 5104 LNCS, pages 28–39. Springer Berlin Heidelberg, Berlin, Heidelberg.

Dassault Systèmes (2011). Abaqus Scripting User's Manual Abaqus 6.11 Scripting User's Manual.

Duval, A., Al-akhras, H., Maurin, F., Elguedj, T., Duval, A., Al-akhras, H., Maurin, F., and Elguedj, T. (2014). Abaqus/CAE 6.14 User's Manual. *Dassault Systémes Inc. Providence, RI, USA*, IV(June):1–6.

Fenner, R. T. and Reddy, J. N. J. N. (2012). Mechanics of solids and structures.

Frank, M., Drikakis, D., and Charissis, V. (2020). Machine-learning methods for computational science and engineering. *Computation*, 8(1):1–35.

Geers, M. G. D. and Schreurs, P. (2015). Lecture notes: Solid mechanics. *Solid Mechanics*, pages 1–223.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M., 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. PMLR.

Goury, O. and Duriez, C. (2018). Fast, Generic, and Reliable Control and Simulation of Soft Robots Using Model Order Reduction. *IEEE Transactions on Robotics*, 34(6):1565–1576.

Kingma, D. P. and Ba, J. L. (2015). Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–15.

Kouznetsova, V. (2020). Lecture notes: Advanced Computational Continuum Mechanics.

Mathworks Inc. (2020). *MATLAB (R2020b).* The MathWorks Inc., Natick, Massachusetts.

Mendizabal, A. (2020). Machine Learning meets real-time Numerical Simulation - Application to surgical training, preoperative planning and surgical assistance.

Miller, K., Joldes, G., Lance, D., and Wittek, A. (2006). Total Lagrangian explicit dynamics finite element algorithm for computing soft tissue deformation. *Communications in Numerical Methods in Engineering*, 23(2):121–134.

Nabian, M. A. and Meidani, H. (2020). Physics-driven regularization of deep neural networks for enhanced engineering design and analysis. *Journal of Computing and Information Science in Engineering*, 20(1):1–23.

Niroomandi, S., Alfaro, I., Cueto, E., and Chinesta, F. (2008). Real-time deformable models of non-linear tissues by model reduction techniques. *Computer Methods and Programs in Biomedicine*, 91(3):223–231.

Niroomandi, S., Gonzalez, D., Alfaro, I., Weldt, F. B., Leygue, A., Cueto, E., and Chinesta, F. (2017). Real Time Simulation of Biological Soft Tissues : A PGD Approach.

Nvidia (2018). Documentation: Nvidia T4 70W low profile PCIe GPU accelerator. (March).

Phellan, R., Hachem, B., Clin, J., Mac-Thiong, J. M., and Duong, L. (2021). Real-time biomechanics using the finite element method and machine learning: Review and perspective. *Medical Physics*, 48(1):7–18.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707.

Roewer-Despres, F., Khan, N., and Stavness, I. (2018). Towards Finite-Element Simulation Using Deep Learning. *15th International Symposium on Computer Methods in Biomechanics and Biomedical Engineering*, page 2018.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *IEEE Access*, 9:16591–16603.

Sluis, O. V. D., Beers, P. V., Lau, K., Baragona, M., Maessen, R., and Lavezzo, V. (2018). Philips internal report on hybrid models. (December).

UPSIM (2022). Healthcare usecases.

van der Sluis, O. (2020). "Digital Twins and Hybrid Intelligence in Healthcare". *EAISI AI Café Second Edition*, Eindhoven,.

van Houten, H. (2018). The rise of the digital twin: how healthcare can benefit, Blog — Philips.

Van Rossum, G. and Drake Jr, F. L. (1995). *Python reference manual.* Centrum voor Wiskunde en Informatica Amsterdam.

Young, W. C. and Budynas, R. G. (2002). *Roark's formulas for stress and strain*, volume 7. McGraw-Hill Professional Publishing,, New York, USA :, 7 edition.

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Declaration concerning the TU/e Code of Scientific Conduct for the Master's thesis

I have read the TU/e Code of Scientific Conduct[i].

I hereby declare that my Master's thesis has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct

Date

22-03-2022

……………………………………………………………………

Name

R.J. Mestriner

……………………………………………………………………

ID-number

0945956

……………………………………………………………………

Signature

[illegible]

……………………………………………………………………

Submit the signed declaration to the student administration of your department.

[i] See: https://www.tue.nl/en/our-university/about-the-university/organization/integrity/scientific-integrity/
The Netherlands Code of Conduct for Scientific Integrity, endorsed by 6 umbrella organizations, including the VSNU, can be found here also. More information about scientific integrity is published on the websites of TU/e and VSNU