UNIVERSITÉ DU LUXEMBOURG

PhD-FSTM-2023-102
The Faculty of Science, Technology and Medicine

DISSERTATION

Presented on 18/09/2023 in Esch-sur-Alzette

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN Sciences de l'Ingénieur

by

Saurabh DESHPANDE
Born on 17 November 1993 in Latur (India)

DATA DRIVEN SURROGATE FRAMEWORKS FOR COMPUTATIONAL MECHANICS: BAYESIAN AND GEOMETRIC DEEP LEARNING APPROACHES

Dissertation defence committee

Dr. Stéphane Bordas, Supervisor
Professor, Université du Luxembourg

Dr. Andreas Zilian, Chair
Professor, Université du Luxembourg

Dr. Jakub Lengiewicz, Vice Chair
Research Specialist, Polish Academy of Sciences

Dr. Stéphane Cotin, External
Professor, Research Director, INRIA

Dr. Eleni Chatzi, External
Professor, Chair of Structural Mechanics, ETH Zurich

Data Driven Surrogate Frameworks for Computational Mechanics: Bayesian and Geometric Deep Learning Approaches

Saurabh Deshpande

Supervisor: Prof. Stéphane Bordas

Doctoral Programme in Computational Sciences
University of Luxembourg

This dissertation is submitted for the degree of
Doctor of Philosophy

I dedicate this thesis to my parents, whose unwavering commitment and countless sacrifices have allowed me to pursue a better life. Their belief in my abilities has been a constant source of motivation, and their unconditional love has been a guiding light through every step of life's journey ...

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Saurabh Deshpande

Acknowledgements

I am immensely grateful to every person and event in my life that has influenced me throughout my PhD journey and life in general. George Adams rightly said, “There is no such thing as a "self-made man". We are made up of thousands of others. Everyone who has ever done a kind deed for us, or spoken one word of encouragement to us, has entered into the makeup of our character and of our thoughts, as well as our success.”

I would like to begin by thanking my supervisor, Stéphane Bordas, for his constant support and guidance, for being a friend more than a supervisor, for giving unparalleled freedom in conducting research, for creating collaboration opportunities with the bests, and, importantly, for constantly inspiring me through your professional and personal achievements. Stéphane, your energy is contagious! A big thank you to Odile for ensuring my PhD journey was incredibly smooth through her excellent administrative support.

If I were to credit one person for the success of this thesis, it would undoubtedly be Jakub. Jakub, you literally appeared as a light at the end of the tunnel in this journey. Thank you for being such an incredible guide. I have always enjoyed our long scientific discussions, joint paper writing (excluding the times I had to revise figures over 10 times ;)), and our running sessions (where I admittedly slowed you down). Over our 2.5 years of collaboration, I’ve directly witnessed your curiosity, scientific rigor, and humility. I hold deep admiration for you as both a scientist and a person. Thank you for consistently uplifting me during both professional and personal challenges. My best wishes always go out to you, Tomek, Tymek, and Anja. Also, thanks to my other collaborators, Ian, Camilo, Stéphane Urcun, Hussein, and more. It is a pleasure working with you. I truly appreciate your expertise and support.

I would like to express my gratitude to my PhD committee members. Stéphane Cotin, whose work has been a main source of inspiration for my thesis (I can’t forget Alband & Andrea). Stéphane, I have huge appreciation for Team MIMESIS’s work. Also, thanks to you and the entire team for being so warm and welcoming during my month long visit. Also, I would like to thank Andreas Zilian, for providing crucial feedback and helpful insights during my CET meetings. A special thanks to Eleni Chatzi, for agreeing to be a part of my defense committee.

This PhD wouldn't have been possible without the generous funding from the ITN Rainbow network. I would like to thank Kenny, Christine, Ehsan, Faezeh, Thomas, Torkan, Patricia, José, Hamed, Christos, Kate, Zhongyun, Antoine, and all the mentors. A special thanks goes to Vasilis for his kindness and assistance during the initial uncertain phase of my PhD.

I have been extremely fortunate to meet amazing people during my stay in Luxembourg. To being with, I wouldn't have asked for better office mates. Arnaud, Milad, Aflah, Geremy, Marie, Mudhafar and Mahdi, thanks a ton for many interesting stories and a wonderful time. You guys were kind enough to tolerate my unconscious humming. From our scientific discussions, squeezing sessions (chess, table tennis, foosball), music breaks, to our mindless conversations, everything has a special place in my heart.

Coming to the 4th floor, seniors first, I extend my gratitude to Olivier, Jack, Lars, Pratik, Anas, Anina, Eleni, Sajad, Asif, Vikram, and Onkar for their timely guidance and insightful discussions. Immense thanks to Stéphane Urcun for his musical suggestions that added beauty to my new song. Olivier, I miss our post-lunch coffee breaks.

This journey wouldn't have been as enriching without my remarkable fellow PhDs. Thank you, Soumi, for your assistance during onboarding; Chintan, for the wonderful meals and discussions. Thomas, for your help with my papers and encouragement for bike rides. Vu, your aid in thesis writing and organizing my Vietnam trip is deeply appreciated. To Zhaoxiang, Aravind, Paris, and Damian (crème de la crème) - our foosball and swimming sessions, not to mention the friendly trash-talking lessons. Sofia's delicious tiramisus remain unforgettable. A shoutout to Julian for his assistance with the Bayesian deep learning component. Gratitude to my fitness coaches, Diego and Paulina, for calisthenics and climbing sessions, and Meryem, for our engaging conversations. Also, heartfelt thanks to Andre, Chrys, Jiajia, Lee Chen, Natasha, Raquel, Renaldo, Jinyuan, Sona, and Parisa.

I'm thankful to Prasad, Fatemeh, Navid, Sinha, and all the others on the floor. I can't forget those who left some time ago: Thank you Anita (for your musical inspiration and our photography walks), Paul, Marco, Raphael, and Hugues. Heartfelt thanks also go to Benjamin, Aline, Susanne, and the entire DSSE administration; it was an honor serving as a student representative.

To my neighbors from UNIVAL-I, who consistently made me feel at home even when far from home - Anshika, Sam, Apurva, Richa - my sincere gratitude. And to the entire Indian community for their warmth and the enjoyable game and dinner nights. I can never fail to mention my beloved instimates, for their unwavering mental support, constant presence whenever needed, and their perpetual inspiration through their accomplishments.

Lastly, I would like to acknowledge my constants - Aai, Baba, Sayali, Achyut, and the entire family - for their unwavering support. To conclude, I want to thank myself for my persistence, composure, hardwork during this journey, and unyielding passion for various aspects of life.

'Onwards and upwards!'

Scientific Contributions

Journal articles

- **Saurabh Deshpande**, Jakub Lengiewicz, Stéphane P.A. Bordas. 'Probabilistic Deep Learning for Real-Time Large Deformation Simulations'. Computer Methods in Applied Mechanics and Engineering (CMAME), 2022. https://doi.org/10.1016/j.cma.2022.115307.
- **Saurabh Deshpande**, Raúl I. Sosa, Stéphane P.A. Bordas, Jakub Lengiewicz. 'Convolution, aggregation and attention based deep neural networks for accelerating simulations in mechanics'. Frontiers in Materials, 2023. https://doi.org/10.3389/fmats.2023.1128954.

Preprints

- **Saurabh Deshpande**, Stéphane P.A. Bordas, Jakub Lengiewicz. 'MAgNET: A Graph U-Net Architecture for Mesh-Based Simulations'. arXiv, 2023. Under review at Engineering Applications of Artificial Intelligence. https://doi.org/10.48550/arXiv.2211.00713.
- **Saurabh Deshpande**, Hussein Rappel, Stéphane P.A. Bordas, Jakub Lengiewicz. 'A probabilistic reduced-order emulation framework for nonlinear solid mechanics'. To be submitted (chapter 7).

In preparation

- **Saurabh Deshpande***, Stéphane Urcun*, Stéphane P.A. Bordas et al. 'Keloid nuclei counting with deep learning based object detection tools'.
- Camilo S., **Saurabh Deshpande**, Stéphane P.A. Bordas et al. 'Surrogate models for estimating macroscopic thermoviscoelastic behavior of 3d printed composite polymers'.

Conferences

- **Saurabh Deshpande**, , Stéphane Bordas, Lars Beex, Stéphane Cotin and Anina Glumac. 'Data Driven Surgical Simulations'. 14th World Congress on Computational Mechanics (WCCM), 2020. https://orbilu.uni.lu/handle/10993/42677.
- **Saurabh Deshpande**, Jakub Lengiewicz, Stéphane P.A. Bordas. 'Real-time large deformations: A probabilistic deep learning approach'. 18th European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS), 2022. http://hdl.handle.net/10993/52344.

- Arnaud Mazier, Thomas Lavigne, Jakub Lengiewicz, **Saurabh Deshpande**, Stéphane Urcun and Stéphane Bordas. 'Towards realtime patient-specific breast simulations: from full-field information to surrogate model'. 9th World Congress of Biomechanics (WCB), 2022. https://orbilu.uni.lu/handle/10993/51163.
- **Saurabh Deshpande**, Jakub Lengiewicz, Stéphane P.A. Bordas. 'Real-Time Large Deformation Simulations Using Probabilistic Deep Learning Framework'. The Platform for Advanced Scientific Computing (PASC), 2022. http://hdl.handle.net/10993/52829.
- **Saurabh Deshpande**, Jakub Lengiewicz, Stéphane P.A. Bordas. 'Real Time Hyperelastic Simulations with Probabilistic Deep Learning'. 15th World Congress on Computational Mechanics (WCCM), 2022. https://orbilu.uni.lu/handle/10993/52345.
- **Saurabh Deshpande**, Jakub Lengiewicz, Stéphane P.A. Bordas. 'Novel Geometric Deep Learning Surrogate Framework for Non-Linear Finite Element Simulations'. The Platform for Advanced Scientific Computing (PASC), 2023.
- **Saurabh Deshpande**, Raúl I. Sosa, Stéphane P.A. Bordas, Jakub Lengiewicz. 'Novel deep learning approaches for learning scientific simulations'. The 14th International Conference of Computational Methods (ICCM), 2023.

Outreach/Seminars

- **Saurabh Deshpande**, Arnaud Mazier. 'Digital twinning for real-time simulation'. European Investment Bank (EIB) Tech fair, Luxembourg, 2019.
- Machine Learning Seminars, Team Legato, University of Luxembourg. (presented in three seminars)
- **Saurabh Deshpande**, 'Probabilistic Deep Learning for Real-Time Large Deformation Simulations'. Team MEMESIS seminar, INRIA Strasbourg, 2022.

Software

- **Saurabh Deshpande**. Source codes for Multi-channel Aggregation Network (MAgNET).
  Github repository: https://github.com/saurabhdeshpande93/MAgNET
  Dataset repository: https://doi.org/10.5281/zenodo.7784804
- **Saurabh Deshpande**, Raul I. Sosa. Implementation of convolutional-aggregation and attention based neural networks for mechanics simulations.

Github repository: https://github.com/saurabhdeshpande93/convolution-aggregation-attention

Dataset repository: https://doi.org/10.5281/zenodo.7585319

Abstract

In modern applications, high-fidelity computational models are often impractical due to their slow performance and also lack information about the certainty of their predictions. Deep learning techniques have recently emerged as a powerful tool for accelerating such predictions. However, these techniques can be inefficient when confronted with larger and more complex problems. This thesis introduces innovative deep learning surrogate frameworks that are scalable, robust, require minimum hyper-parameter tuning, are fast at the inference stage, and are accurate in forecasting non-linear deformation responses of solid objects. These surrogate frameworks are constructed using various deep learning techniques under deterministic as well as Bayesian settings. Bayesian frameworks enable us to capture uncertainties and provide a means to trust the predictions of the neural network approaches.

This thesis introduces a new geometric deep learning framework, called MAgNET (Multi-channel Aggregation Network). MAgNET is designed to handle large-dimensional graph-structured data using an encoder-decoder architecture. MAgNET is built upon the novel MAg (Multichannel Aggregation) operation, which generalises the concept of multi-channel local operations found in convolutional neural networks to arbitrary non-grid inputs. The MAg layers are combined with the novel graph pooling/unpooling operations to form a powerful graph U-Net architecture capable of efficiently performing supervised learning on large-dimensional graph-structured data, like complex meshes. Additionally, the thesis demonstrates the use of state-of-the-art attention-based networks, which have revolutionized various engineering fields but have remained unexplored for their uses in the field of computational mechanics.

We demonstrate the efficiency and versatility of the proposed frameworks by applying them to surrogate modeling for non-linear finite element simulations. Our suggested methods, particularly the MAgNET architecture, possess broad applicability, enabling researchers and practitioners to explore novel modeling scenarios and applications. Through the open sharing of the source codes and datasets employed, this thesis not only makes a significant contribution to the field of surrogate modeling in mechanics but also paves the way for numerous research opportunities for their utilisation in various engineering and scientific applications.

Table of contents

List of figures

List of tables

Nomenclature

Roman Symbols

$\mathbf{A}$	Adjacency matrix
$\tilde{\mathbf{A}}$	Pooled adjacency matrix
$\mathbf{B}_h$	nodal body forces
$\boldsymbol{B}$	left Cauchy-Green strain tensor
$\boldsymbol{b}$	external body forces
$\boldsymbol{b}_h$	external body forces in discretised representation
$\boldsymbol{C}$	right Cauchy-Green strain tensor
$\mathbf{d}^l$	l-th neural network layer
$\boldsymbol{u}$	displacement in continuous representation
$\boldsymbol{u}_h$	displacement in discretised representation
L	Total number of layers in neural network
$\boldsymbol{E}$	Green Lagrange strain tensor
E	Elasticity modulus
$\boldsymbol{f}_{\text{ext}}$	External forces
$\boldsymbol{f}_{\text{int}}$	Internal forces
$\boldsymbol{F}$	Deformation gradient
$\mathbf{G}/\mathbf{G}_i$	Subgraph/i-th subgraph
I_c	Invariant in terms of deformation gradient

$\boldsymbol{J}$ Jacobian, to map volume element from undeformed to deformed configuration

$\mathbf{K}$ stiffness matrix

$\boldsymbol{k}^l$ learnable weights for l-th neural network layer

$\boldsymbol{P}$ 1st Piola-Kirchhoff stress tensor

$\mathbf{R}(\boldsymbol{u}_h)$ Residual in terms of discretised displacements (balance of internal and external forces)

$\mathbf{S}/\mathbf{S}_i$ Set of nodes in a graph/i-th subgraph

$\tilde{\mathbf{G}}$ Pooled graph

$W(\boldsymbol{F})$ Strain-energy density potential

$\mathbf{T}_h$ nodal traction forces

$\boldsymbol{t}$ traction forces

$\boldsymbol{t}_h$ traction forces in discretised representation

$\mathbf{U}_h$ nodal displacement

$\boldsymbol{b}^l$ learnable biases for l-th neural network layer

$\boldsymbol{X}$ Position vector in undeformed configuration reference

$\boldsymbol{x}$ Position vector in deformed configuration reference

Greek Symbols

α Learning rate

$\mathcal{G}$ MAgNET (graph U-net) network

$\mathcal{U}$ CNN U-net network

$\mathcal{D}$ Dataset

Γ_N Neumann's boundary

Γ_u Dirichlet boundary

$\boldsymbol{\epsilon}_{\text{linear}}$ Linear strain tensor

$\boldsymbol{\theta}_{\text{det/MLE/VB}}$ parameters of deterministic/MLE/variational Bayes neural networks

$\mathcal{A}$ Activation function

$\mathcal{F}$ Degrees of freedom

$\mathcal{L}_{\text{det/MLE/VB}}$ Loss for deterministic/MLE/variational Bayes neural networks

Ω Computational domain in the deformed configuration

Ω_0 Computational domain in the undeformed configuration

ϕ One to one mapping from deformed to undeformed configuration

ρ density

μ, λ Lame's parameters

$\bar{e}$ Average NN prediction error for the entire test set

$\mathcal{M}$ N. of monte carlo samples in ELBO calculation

$\mathcal{U}_\mu$ Mean prediction of Baysian U-Net

$\mathcal{U}_\sigma$ Standard deviation of Baysian U-Net prediction

e_m Mean NN prediction error of m-th test example

ν Poisson's ratio

$\boldsymbol{\epsilon}$ Strain tensor

$\boldsymbol{\sigma}$ Cauchy stress tensor

Other Symbols

$\nabla \cdot (\bullet)$ Divergence operation

det( ) Determinant of a matrix

$\nabla(\bullet)$ Gradient operation

$(X, Y) \sim \mathcal{D}$ (X, Y) is a sample from $\mathcal{D}$

$\mathbb{E}$ Expectation of random variables

$\|(\cdot)\|_2$ L-2 norm

$\mathbb{N}_i$ finite element shape functions

Acronyms / Abbreviations

ANN Artificial Neural Network

BDL Bayesian Deep Learning

CNN Convolutional Neural Network

CPU Central processing unit

DL Deep Learning

GDL Geometric Deep Learning

GNN Graph Neural Network

DNN Deep Neural Network

ELBO Evidence lower bound

FEM Finite Element Method

FETI Finite Element Tearing and Interconnecting

GP Gaussian Process

GPU Graphical processing unit

HPC High performance computing

MAg Multichannel Aggregation

MAgNET Multichannel Aggregation Network

MCMC Markov Chain Monte Carlo

MLE Maximum Likelihood Estimation

ML Machine Learning

MLP Multilayer perceptron

MOR Model order reduction

PDE Partial Differential Equation

PGD Proper Generalised Decomposition

PINN Physics Informed Neural Network

POD Proper Orthogonal Decomposition

RBF Radial Basis Function

ReLU Rectified Linear Unit

VB Variational Bayes

VI Variational Inference

Chapter 1

Introduction

Scientific simulations play a significant role in advancing our understanding of complex systems. They rely on mathematical models and computer-based techniques, that not only help us to gain insights into real-world scenarios but also make predictions. Traditionally, these mathematical models are built and tested for hypotheses, which are based on the existing knowledge of the system being studied. These models are then used to simulate the behaviour of a system under different conditions. On the other hand, the technological advances in the Industry 4.0 era have given birth to an immense amount of data, and so the data-driven models. These models rely on the analysis of data to identify the underlying relationships that can be used to make predictions and optimize processes.

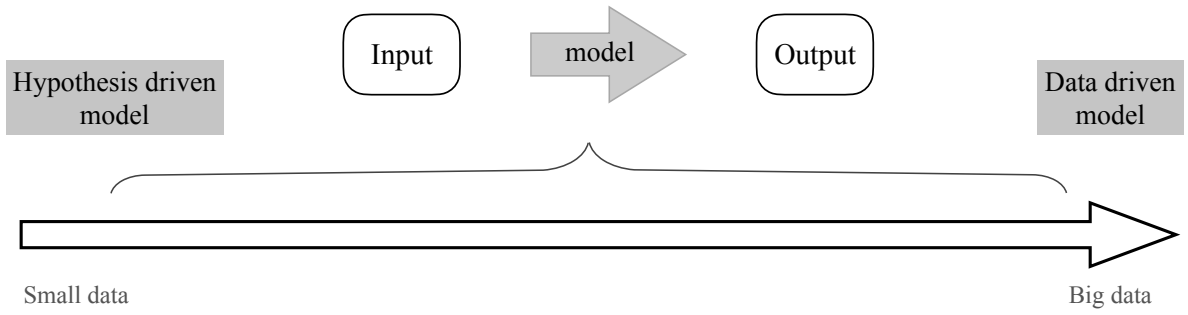


Fig. 1.1 A mathematical model is used to identify the relationships between different quantities of interest in the system being studied. A model can fall somewhere in the paradigm of hypothesis to a data-driven model. Hypothesis-driven models require less data to validate them, whereas data-driven models rely on large amounts of data. A hybrid approach combines hypothesis-driven modeling with data-driven modeling. This thesis majorly focuses on developing data-driven approaches.

Data-driven, hybrid, and pure hypothesis-driven approaches represent distinct methodologies in scientific inquiry. In a data-driven approach, the emphasis is on extracting patterns and insights directly from available data, often without a preconceived hypothesis [Kirchdoerfer and Ortiz, 2016; Montáns et al., 2019; Stainier et al., 2019]. This method leverages the power of computational techniques and statistical analysis to uncover hidden relationships and trends within the data, allowing new hypotheses to emerge organically. On the other hand, hybrid approaches combine elements of both data-driven and hypothesis-driven methods [Chatzi et al., 2010; Law et al., 2015; Raissi et al., 2019]. These approaches start with initial hypotheses but also incorporate data exploration to refine and validate these hypotheses. In contrast, the pure hypothesis-driven approach begins with a well-defined hypothesis based on existing theories or prior knowledge. Researchers conduct targeted experiments or observations to test and validate these hypotheses rigorously. Each of these approaches offers unique advantages: data-driven methods open the door to new discoveries, hybrid methods harness the strengths of both approaches, and pure hypothesis-driven methods offer a structured and systematic means of validating existing theories [Liu et al., 2019]. The choice of approach depends on the research goals, available resources, and the nature of the scientific question at hand.

Data-driven, hybrid, and pure hypothesis-driven approaches represent distinct methodologies in scientific inquiry. In a data-driven approach, the emphasis is on extracting patterns and insights directly from available data, often without a preconceived hypothesis. Another advantage of data-driven modeling is that it can be used to identify complex relationships and patterns that may not have been apparent through the traditional hypothesis-driven modeling approach. It can also be used to optimize processes in real time based on the data generated by sensors and other sources. However, data-driven modeling also requires large volumes of high-quality data and sophisticated algorithms to analyze the data. This thesis focuses on building such sophisticated data-driven frameworks for performing simulations in mechanics.

Traditional methods are computationally expensive

Mathematical models are commonly expressed using partial differential equations (PDEs). They can be solved analytically or numerically using a variety of techniques, depending on the complexity of the problem and the desired level of accuracy and efficiency. Complex and real-world problems are dealt with numerical methods. In the particular context of mechanics, the finite element method (FEM)[Zienkiewicz and Taylor, 1991] is one of the most commonly used numerical approaches. FEM simulations are accurate, but this accuracy comes at the cost of significant computational efforts, especially for non-linear problems. While hardware developments and software optimizations have facilitated faster FEM computations to some extent, they still fall short of meeting real-time constraints, which are critical in certain

application domains such as robotics [Choi et al., 2021; Rus and Tolley, 2015] and biomedical simulations [Cotin et al., 1999; Courtecuisse et al., 2014a; Mazier and Bordas, 2023].

In short, the need for computationally efficient simulation tools in various fields, including engineering, science, and medicine, has become increasingly crucial as the complexity and size of problems continue to grow. It is essential to find computational inexpensive solutions that can handle the scale and complexity of the problem within a reasonable amount of time. Thus speeding up such computational models whilst maintaining the desired accuracy is an active area of research and one of the main motivations of this thesis work.

Machine/deep learning techniques and their uses as surrogate models

With exponential growth in data generation and storage capabilities, data-driven approaches have emerged as a powerful tool for analyzing and understanding complex systems. Machine learning (ML) techniques are a key component of these data-driven approaches. ML algorithms can be used to analyze and learn from large data sets, which can then be used to develop predictive models.

Scientific simulations have already begun to undergo a revolution, thanks to recent developments in machine learning. Without any mathematical explanation of the problem, machine learning algorithms may transfer the input of a function to its output given enough ground truth data. These approaches appear promising for learning behaviour of systems without relying on empirical models because they are driven directly by data [Bomarito et al., 2021; Rudy et al., 2017]. Many applications where prediction speed is very important can benefit from the high inferring speed of these methods [Mendizabal et al., 2019b; Sánchez-Sánchez and Izzo, 2018]. Within the class of ML methods for data-driven modeling, deep learning (DL) approaches have seen great success due to their ability to efficiently extract complex relationships present in the underlying data. DL models have proven to be accurate and efficient in predicting non-trivial nonlinear relationships in data, but they need it in sufficiently high amounts.

Lately, DL models have also been successfully utilised as surrogate models in mechanics [Haghighat et al., 2021; Krokos et al., 2022a; Mianroodi et al., 2021]. In the ideal case scenario, these models would be trained on real-world datasets, which is not always feasible. Obtaining the necessary amount of training data is often difficult when it originates from physical experiments. This can be due to multiple factors, such as high costs, risks & difficulties associated with the experiments, or data privacy clauses. In such cases, data can be generated synthetically through high-fidelity simulations done *in silico* [Kim et al., 2022; Le et al., 2017; Pfeiffer et al., 2019]. This thesis follows this approach and in particular focuses on deep learning surrogate models trained on synthetic data generated from finite element simulations in non-linear elasticity.

Scalability of DL techniques

One of the main objectives of this thesis is to investigate high-dimensional relationships characterized by large input and/or output sizes. Such relationships can be observed in various domains, including medical imaging, as in the case of full-field measurement data [Lavigne et al., 2022], or in the generation of synthetic mesh data from finite element simulations, as shown in studies by Lorente et al. [2017] and Pellicer-Valero et al. [2020].

Many of the existing DL-based surrogate frameworks are based on fully connected networks, which process the entire input at once, which can be problematic when dealing with high-dimensional data [Shrestha and Mahmood, 2019]. Fully connected layers treat the input as a flat vector and do not take into account the spatial structure present in data, for example, the quantity of interest of closely located nodes can be co-related, and this correlation is not taken into account by FC networks. As a result, they may not efficiently capture spatial patterns, leading to sub-optimal performance in building surrogate models for mechanics and other applications. To address this challenge, modern solutions have emerged. Convolutional neural networks have gained prominence [Obiols-Sales et al., 2020; Zhao et al., 2019], along with the utilization of graph neural networks [Sanchez-Gonzalez et al., 2020; Vlassis et al., 2020], and techniques based on transformer networks [Geneva and Zabaras, 2022; Hassanian et al., 2023]. These methods are experiencing growing adoption due to their effectiveness in scientific simulations. The upcoming chapters of this thesis provide an in-depth exploration of each of these approaches and present novel deep-learning frameworks built upon these foundations.

Chapter 4 of the thesis introduces a Convolutional Neural Network (CNN) U-Net architecture framework that is capable of efficiently predicting non-linear deformations of soft bodies in real-time [Deshpande et al., 2022]. This framework scales remarkably well with input size, making it suitable for grid-structured inputs. However, its limitation lies in the fact that it can only be applied to structured meshes, whereas real-world applications are often represented with unstructured meshes.

To address this limitation, Chapter 5 proposes a novel geometric deep learning framework called MAgNET [Deshpande et al., 2023a,b]. MAgNET extends the concept of localized operations in CNNs to arbitrary graph-structured data, efficiently leveraging mesh topology to construct neural network architectures. MAgNET comprises of two novel deep-learning layers specifically designed for graph-structured data applications. As a result, MAgNET can be utilised in a wide range of engineering and scientific applications.

In recent times, attention-based deep neural networks have brought significant advancements to various engineering disciplines. However, their potential applications in computational mechanics remain relatively unexplored. This thesis takes a pivotal step in Chapter 6 by forging

a connection between computational mechanics and cutting-edge transformer models. In this context, we introduce a novel surrogate framework based on the Perceiver IO architecture, designed to perform accurate non-linear finite element simulations [Deshpande et al., 2023c,d].

Uncertainty is inherent to real world systems

Real-world systems are intricate and exhibit nonlinear behavior, making it difficult to completely analyze and predict their outcome. Uncertainty is an inherent part of such systems, which is the absence of comprehensive knowledge or information about its constituent parts. It can arise for several reasons, including changes in the environment, variability in the system's beginning circumstances or boundary conditions, or stochastic processes. Hence it is crucial to produce reliable uncertainty estimates in addition to predictions since uncertainty can affect the reliability and accuracy of the prediction. Particularly, it is very important to know the impact of uncertainty in crucial applications such as surgical simulations [Bui et al., 2018; Mazier et al., 2022] or autonomous driving [Feng et al., 2018; Shafaei et al., 2018]. Otherwise, model predictions can lead to harmful consequences in these critical tasks. Thus, there is a need to develop methods for quantifying and managing uncertainties in scientific simulations.

Uncertainties in engineering systems come from different sources. These can be split into two main types: first, the random variations we see in the data (called aleatoric uncertainty or data uncertainty); and second, uncertainties due to how well our model represents reality (known as epistemic uncertainty or model uncertainty) [Kendall and Gal, 2017b; Kiureghian and Ditlevsen, 2009]. Data uncertainty is there because data naturally has some randomness, and we can't get rid of it. On the other hand, model uncertainty can be reduced by adding more training data [Hüllermeier and Waegeman, 2021]. A significant aspect of model uncertainty lies in its correlation with the volume of data available — as the dataset expands, our confidence in predictions proportionally increases, i.e., model uncertainty decreases. It is important to develop stochastic/Bayesian computational frameworks that can track both types of uncertainties.

Stochastic approaches have been already considered in the context of FEM models [Matthies, 2008; Stefanou, 2009]. In general, they are computationally costly, have convergence issues, and scale badly with input dimensions. One key technique in stochastic modeling called the Monte Carlo method, involves running numerous simulations for each parameter value [Hauseux et al., 2018; Rappel et al., 2020]. Essentially, the Monte Carlo approach turns the integrals needed for statistical expectations into discrete sums, usually large ones. Since material model parameters can span significant ranges, proper sampling across their variations is crucial for accurate outcomes. However, this need for extensive computation makes these simulations much more computationally expensive compared to deterministic ones [Biehler et al., 2014;

Hauseux et al., 2017b]. Hence these approaches are not suitable for applications requiring fast solutions.

In order to effectively handle both types of uncertainties, this thesis introduces two Bayesian approaches in the context of ML/DL techniques, for their applications to high dimensional problems in mechanics:

1. Bayesian Deep Learning (BDL) framework: This approach is presented in Chapter 4 and is based on the approximate Bayesian inference—Variational Inference (VI), used in the context of deep neural networks [Deshpande et al., 2022; Graves, 2011]. VI enables straightforward use of the framework for high-dimensional problems. By incorporating uncertainty estimates into the model, it provides a more robust and reliable solution.

2. Using Gaussian processes regression (GPR) with autoencoder networks: Chapter 7 of the thesis presents this novel approach based on the Gaussian processes (GP), a well-regarded machine learning technique [Rasmussen et al., 2006]. GPs face challenges when dealing with a large number of data points and high dimensions in a problem. To address this, we employ GPs on data that has been dimensionally compressed using autoencoder neural networks [Hinton and Salakhutdinov, 2006]. This integration of autoencoder networks significantly lessens the computational load and enhances the capacity to efficiently apply GPs to complex, high-dimensional problems.

By employing these two approaches, this thesis effectively addresses uncertainties, providing more accurate and reliable results for high-dimensional problems. Both surrogate frameworks are specifically developed for simulating mechanical deformation responses of soft bodies along with their associated uncertainties.

Potential applications to biomedical simulations

This research work is a part of the European Innovative Training Network (ITN) named Rapid Biomechanics Simulation for Personalized Clinical Design (RAINBOW). The program received financial support from the European Union's Horizon 2020 Research and Innovation initiative through a Marie Skłodowska-Curie grant. As part of this program, 15 Ph.D. students worked collaboratively to investigate and devise innovative techniques in biomedical simulations.

Ever since the inception of computer technology, experts have envisioned its potential applications in the field of medicine. While surgery is a life-saving procedure, complications or unfamiliar techniques can lead to an increased risk of incidents [Sarker and Vincent, 2005]. As such, it is vital to train competent and skilled surgeons who can handle any pathology

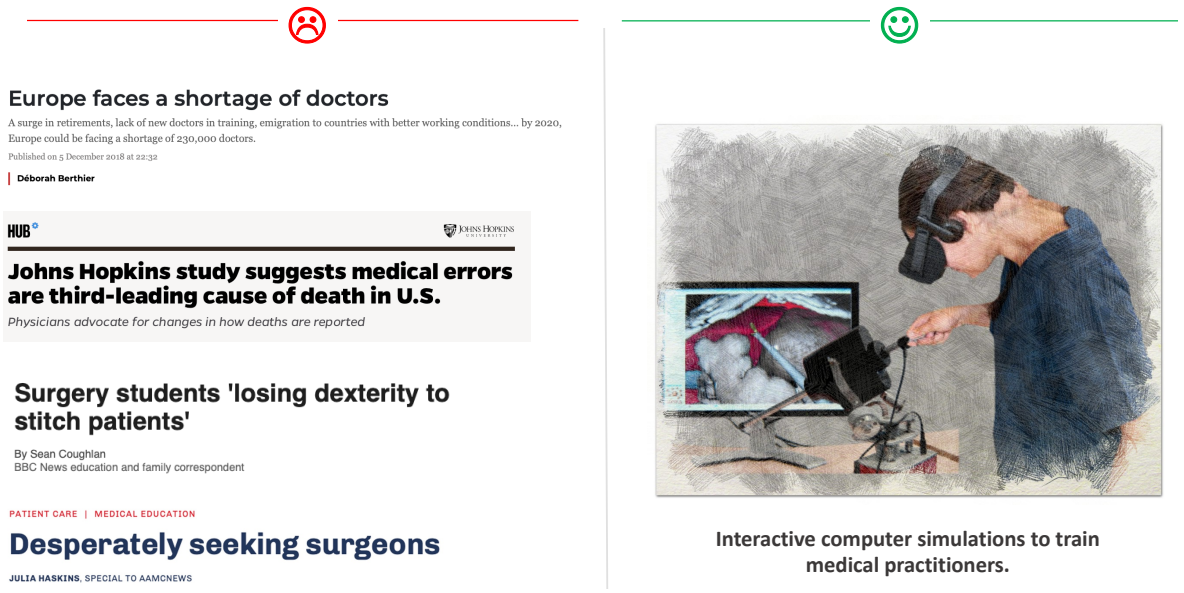


Fig. 1.2 Various scientific studies and surveys have shed light on the seriousness of medical errors and anticipate a shortage of surgeons in the coming decade (cutouts of media coverage on the left). Simulation-based medical training has been proven to have many advantages which help improve medical practitioners' competencies, and in return, improve patient safety and reduce health care costs.

[Rosser Jr et al., 2000]. Over the years, computer simulations have emerged as a preeminent tool to accomplish this task. Biomedical simulations are being widely used to assist in various medical procedures [Adagolodjo et al., 2018; Plantefeve et al., 2016], for training purposes [Cotin et al., 2000; Talbot et al., 2017], and for preoperative planning [Alcañiz et al., 2022; Mazier et al., 2021]. It can be said that the future of surgical education lies in the development of realistic bio-mechanical simulators. In such training systems, the surgical tools are deformable objects and the vessels are generally considered to be rigid. Such models need to be interactive, therefore the simulations must be computed in real-time. One of the aspects of this thesis is to conduct real-time deformation simulations of soft bodies, like human organs. These simulations are crucial because they serve as the foundational components for the development of advanced surgical simulators.

Objective of the thesis

The primary objective of this thesis is to create advanced and original deterministic as well as Bayesian deep learning frameworks that can model non-linear finite element solutions, to tackle problems in solid mechanics. The frameworks emphasize scalability, robustness, require minimal hyperparameter tuning, and can efficiently deal with high-dimensional problems. Moreover, a

crucial aspect of proposed frameworks is their ability to track associated uncertainties, thereby providing means to establish confidence in the neural network surrogate modeling.

Outline of the thesis

In this chapter, we provided a general introduction and the objective of the thesis, which is to develop deterministic and Bayesian deep learning surrogate techniques to solve problems in solid mechanics. In Chapter 2 we first introduce the fundamentals of mechanics and provide an overview of finite element formulation. Then in Chapter 3, we introduce the basics of deep learning techniques, which is a rapidly evolving field of artificial intelligence. The subsequent chapters provide an in-depth study of DL surrogate methods, these chapters comprise novel work developed in this thesis. Chapter 4 proposes a probabilistic deep learning framework to solve real-time large deformation simulations along with their uncertainties. Chapter 5 presents a novel geometric deep learning framework in which we introduce two deep learning layers for efficiently learning on graph-structured data. Chapter 6 presents attention-based deep neural networks for accelerating simulations in mechanics and also provides a comparative study of the frameworks proposed in earlier chapters. In Chapter 7, we propose a probabilistic framework that combines multi-output Gaussian processes with autoencoder networks and show its uses for surrogate modeling in mechanics alongside uncertainty quantification. Finally, Chapter 8 provides a general conclusion of the thesis and motivates future research directions.

Chapter 2

Finite element formulation

Numerical methods are used in a wide range of fields, including a multitude of engineering and natural science domains [Jansari et al., 2022; ki Choi et al., 2023; Papavasileiou et al., 2022; Suchde and Kuhnert, 2019]. They are particularly useful when exact solutions to mathematical problems are not possible due to the computational complexity of the problem.

The finite element method (FEM) is one of the most common numerical methods for solving differential equations arising in engineering and mathematical modeling. It gives a numerical approximation of a partial derivatives equation discretizing the object using nodes connected by elements. It is extensively used in fields such as solid mechanics, structural analysis, and fluid dynamics [Farina et al., 2021; Piranda et al., 2021; Urcun et al., 2021]. The main objective of this thesis is to calculate the non-linear deformations of solid bodies when subjected to external forces. Therefore, this chapter will begin by providing a concise overview of elasticity theory, followed by the solution strategy using the finite element method.

2.1 Fundamentals of elasticity in solid mechanics

This thesis focuses on the mechanical modeling of solid, continuous materials, where the solid material is viewed as a collection of countless infinitesimal pieces that can displace, deform, and rotate in response to external stimuli. The assumption made here is that the connectivity between these infinitesimal pieces remains constant, such that they cannot detach from each other or penetrate one another. If one wants to include detachment, one needs to regard fracture mechanics which is not in the scope of this thesis. The fact that the connectivity between all infinitesimal pieces of material remains the same thus means that the solid body deforms in a continuous manner.

The material models described in this thesis only focus on the elastic constitutive description. That is the models are characterised by the fact that no energy is lost during the deformation process, these models are also known as hyperelastic models.

2.1.1 Kinematics

As depicted in Figure 2.1, we consider a deformable body with an initial configuration Ω_0. The position of a particle within Ω_0 is represented by $\boldsymbol{X}$, while the deformed configuration is represented by Ω and the position of a particle within Ω is represented by $\boldsymbol{x}$. A one-to-one mapping, denoted by ϕ, is used to associate the position of a particle $\boldsymbol{X}$ in Ω_0 with the position of the same particle $\boldsymbol{x}$ in Ω, such that $\boldsymbol{x} = \phi(\boldsymbol{X})$.

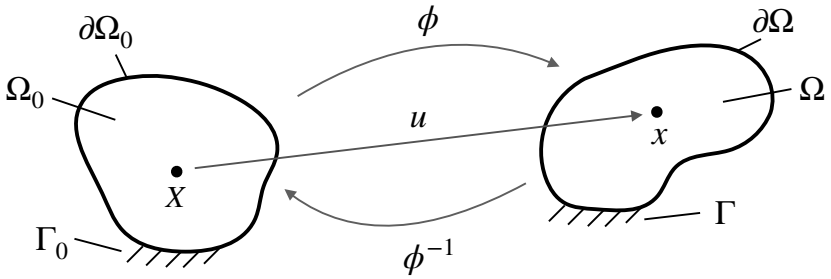


Fig. 2.1 A continuous solid body in the initial configuration on the left is mapped to the deformed configuration through the mapping ϕ (ϕ^{-1} denotes the inverse mapping, which is not in the scope of this thesis). Γ_0 and Γ denote the boundary conditions denoted in Ω_0 and Ω respectively.

Now let us introduce the deformation gradient $\boldsymbol{F}$ that maps an infinitesimal small element $\mathrm{d}\boldsymbol{X}$ in Ω_0 to $\mathrm{d}\boldsymbol{x}$ in Ω as follows:

$$\mathrm{d}\boldsymbol{x} = \boldsymbol{F} \cdot \mathrm{d}\boldsymbol{X}, \tag{2.1}$$

For each point we denote the displacement between the deformed and undeformed position as $\boldsymbol{u}$.

$$\boldsymbol{u} = \boldsymbol{x} - \boldsymbol{X} = \phi(\boldsymbol{X}) - \boldsymbol{X}, \tag{2.2}$$

so the deformation gradient ($\boldsymbol{F}$) can be written as:

$$\boldsymbol{F} = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{X}} = \frac{\partial \phi(\boldsymbol{X})}{\partial \boldsymbol{X}} = \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{X}} + \boldsymbol{I} = \nabla_0 \boldsymbol{u} + \boldsymbol{I}, \tag{2.3}$$

where $\nabla_0(\bullet)$ is the gradient operation with respect to the undeformed configuration Ω_0. The volume element in initial configuration $\mathrm{d}\Omega_0$ is mapped to the $\mathrm{d}\Omega$ in Ω using the Jacobian (J) as:

$$\begin{aligned} \mathrm{d}\Omega &= \boldsymbol{J} \cdot \mathrm{d}\Omega_0, \\ J &= \det(\boldsymbol{F}) \end{aligned} \tag{2.4}$$

Using the deformation gradient $\boldsymbol{F}$, we obtain the right Cauchy-Green strain tensor $\boldsymbol{C}$, and the left Cauchy-Green strain tensor $\boldsymbol{B}$ as follows:

$$\begin{aligned} \boldsymbol{C} &= \boldsymbol{F}^T \boldsymbol{F}, \\ \boldsymbol{B} &= \boldsymbol{F} \boldsymbol{F}^T \end{aligned} \tag{2.5}$$

Both Cauchy-Green strain tensors contain information about the strain, i.e. change of length of a vector. They are both symmetric and positive definite. They do not contain information about the rigid body rotation, i.e. rotation of a vector without change of length. Finally the Green-Lagrange strain tensor $\boldsymbol{E}$ is defined as:

$$\begin{aligned} \boldsymbol{E} &= \frac{1}{2}(\boldsymbol{C} - \boldsymbol{I}), \\ &= \frac{1}{2}(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T) + \frac{1}{2}(\nabla \boldsymbol{u}^T \nabla \boldsymbol{u}) \end{aligned} \tag{2.6}$$

The opening bracket of the equation indicates the linear component of the strain tensor, while the subsequent bracket denotes the non-linear component. In cases where the displacement is sufficiently small, the non-linear component can be disregarded, resulting in a simplified equation.

$$\boldsymbol{\epsilon}_{\text{linear}} \approx \frac{1}{2}(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T) \tag{2.7}$$

2.1.2 Strong and weak form

In the context of elasticity, the strong form and the weak form are two different ways of representing the governing equations of the problem. The strong form comprises differential equations that relate stresses and strains in a solid, derived from the principles of mass, momentum, and energy conservation. It must be satisfied at every point in the domain, along with any boundary conditions.

The weak form, on the other hand, is obtained by multiplying the strong form by a suitable test function and integrating over the domain of interest. It is well-suited for numerical methods,

such as the finite element method, which requires the solution to be represented in a discretized form. The weak form also provides greater flexibility in choosing the approximation space for the solution, allowing for the use of piecewise polynomial functions of varying degrees.

Strong form

At the equilibrium in the deformed configuration the strong form for the governing equation of elasticity can be written through the balance of the momentum as follows:

$$\nabla \cdot \boldsymbol{\sigma} + \rho \boldsymbol{f}_{\text{ext}} = 0, \tag{2.8}$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\nabla \cdot (\bullet)$ is the divergence operation. ρ represents the density of the body and $\boldsymbol{f}_{\text{ext}}$ are the external forces in the deformed configuration.

Different types of stress tensors:

In linear analysis, the distinction between the deformed and undeformed areas is not necessary due to the infinitesimal deformation. However, in the case of large deformation, it becomes crucial to specify the reference frame (deformed or undeformed) used to define the stress tensor. And for this reason, we define two main stress tensors that are relevant to this thesis, which can be intuitively understood as:

1. **Cauchy stress tensor ($\boldsymbol{\sigma}$)** : forces and areas in the deformed configuration.
2. **1$^{\text{st}}$ Piola-Kirchhoff stress tensor ($\boldsymbol{P}$)** : forces in the deformed configuration and areas in the undeformed configuration.

It should be noted that Eq.(2.9) is written in the deformed configuration. Now the strong form can be written in the undeformed configuration through the Piola-Kirchhoff stress tensor as

$$\nabla \cdot \boldsymbol{P} + \rho_0 \boldsymbol{f}_{\text{ext}} = 0, \tag{2.9}$$

First and second Piola-Kirchhoff are expressed in terms of the Cauchy stress tensor as follows:

$$\boldsymbol{P} = J\boldsymbol{\sigma}\boldsymbol{F}^{-T} \tag{2.10}$$

Weak form

Solving problems using the strong form as presented in Eq.(2.9) often involves solving partial differential equations (PDEs) directly, which can be challenging for complex geometries or material behaviors. Thus the main idea of converting the strong form into a weak form is to turn the differential equation into an integral equation, to lessen the burden on the numerical algorithm in evaluating derivatives.

The weak form is obtained by multiplying the strong form by appropriate test functions and integrating over the whole domain Ω as follows:

$$\int_{\Omega} \boldsymbol{P}(\boldsymbol{F}(\boldsymbol{u})) \cdot \nabla \delta \boldsymbol{u} \, dV - \int_{\Omega} \rho \, \boldsymbol{b} \cdot \delta \boldsymbol{u} \, dV - \int_{\Gamma_t} \boldsymbol{t} \cdot \delta \boldsymbol{u} \, dS = 0 \qquad \forall \delta \boldsymbol{u}, \tag{2.11}$$

where $\boldsymbol{b}$ are prescribed body forces, $\boldsymbol{t}$ are prescribed tractions on the Neumann's boundary Γ_N, while the solution $\boldsymbol{u}$ and the variation $\delta \boldsymbol{u}$ belong to appropriate functional spaces, with $\boldsymbol{u} = \bar{\boldsymbol{u}}$ and $\delta \boldsymbol{u} = \boldsymbol{0}$ on the Dirichlet boundary Γ_u.

2.1.3 Constitutive laws: stress-strain relationships

The relationship between the strain tensor $\boldsymbol{\epsilon}$, and the stress tensor $\boldsymbol{\sigma}$, is governed by the mechanical properties of the material and is described through a constitutive law. Constitutive law helps us define how the applied forces on the body are linked to its deformation.

The simplest relation between stress-strain tensors is given by Hooke's law, i.e., which describes a linear relationship. i.e., body displacements are directly proportional to the forces. This linear constitutive equation relates two symmetric tensors through the constitutive matrix $\mathbf{C}$:

$$\boldsymbol{\sigma} = \mathbf{C} \boldsymbol{\epsilon} \tag{2.12}$$

More generally, hyperelastic laws are used to model the mechanical behavior of materials that exhibit nonlinear elasticity. Hyperelasticity deals with the behavior of materials that can undergo significant deformations without undergoing plastic deformation or permanent changes in shape.

The hyperelastic constitutive relationship is expressed through the strain-energy density potential $W(\boldsymbol{F})$ as

$$\boldsymbol{P}(\boldsymbol{F}) = \frac{\partial W(\boldsymbol{F})}{\partial \boldsymbol{F}}, \tag{2.13}$$

All the implementations presented in this thesis are performed using Neo-Hookean hyperelastic law. A particular form of strain energy expression that is used in this work is as follows:

$$W(\boldsymbol{F}) = \frac{\mu}{2}(I_c - 3 - 2\ln J) + \frac{\lambda}{4}(J^2 - 1 - 2\ln J), \tag{2.14}$$

where the invariants J and I_c are given in terms of deformation gradient, with $I_c = \mathrm{tr}(\boldsymbol{F}^T\boldsymbol{F})$. While μ and λ are Lame's parameters, which can be expressed in terms of Young's modulus, E, and Poisson's ratio, ν, as

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}, \quad \mu = \frac{E}{2(1+\nu)}. \tag{2.15}$$

The Neo-Hookean is one of the simple hyperelastic models. It should be noted that the surrogate modeling strategies presented in this work are straightforwardly applicable to more demanding hyperelastic models, such as Mooney-Rivlin and Ogden [Holzapfel, 2002].

2.1.4 Finite element method

The displacement solution ($\boldsymbol{u}$) for a hyperelastic deformation is computed by solving Eq.(2.11). In an ideal scenario, this equation needs to be integrated an infinite number of times which is practically infeasible and hence it is solved by numerical approximation with a finite number of equations. The FEM is a numerical technique that provides a numerical estimate of partial derivative equations by dividing an object into small pieces called elements. These elements are connected by nodes and solutions inside the elements are interpolated from the nodal solutions [Lavigne et al., 2023].

A continuous domain can be discretisation into finite elements in different ways, with triangular and tetrahedron being the most common choices. These elements are suitable to discretise irregular domains and are easier to generate than quadrilateral or hexahedral elements. The CNN-based framework proposed in Chapter 4 of the thesis is only compatible to be used with structured meshes, such as quadrilateral and hexahedral. While Chapter 5 introduces a new framework that can efficiently handle any type of finite element discretisation.

Let us denote the discretised representation for continuous solution $\boldsymbol{u}$ as $\boldsymbol{u_h}$. Then at any location p in the local coordinate system

$$\boldsymbol{u_h}(p) = \sum_{i=1}^{N} \mathbb{N}_i(p)\boldsymbol{u_{h}}_i, \tag{2.16}$$

where $\boldsymbol{u_{hi}}$ is the displacement of i^{th} node (which can be multi-dimensional) and N is the total number of nodes in the mesh (excluding those on the Dirichlet boundary). There is one shape function $\mathbb{N}_i$ for each node in the mesh. Shape function must have local support and be piece-wise continuous.

Similarly, discretised forms of body forces and tractions are expressed with shape functions as follows:

$$\begin{aligned} \boldsymbol{b_h}(p) &= \sum_{i=1}^{N} \mathbb{N}_i(p) \boldsymbol{b_{hi}}, \\ \boldsymbol{t_h}(p) &= \sum_{i=1}^{N} \mathbb{N}_i(p) \boldsymbol{t_{hi}}, \end{aligned} \tag{2.17}$$

After substituting these terms, the problem expressed by Eq. (2.11) will take the matrix form representing a system of non-linear equations

$$\mathbf{K}\mathbf{U}_h = \mathbf{B}_h + \mathbf{T}_h, \tag{2.18}$$

Where $\mathbf{K}$ is the global stiffness matrix and $\mathbf{U}_h, \mathbf{B}_h, \mathbf{T}_h$ are global displacement, volume, and surface force vectors given as follows:

$$\mathbf{U}_h = \begin{pmatrix} \boldsymbol{u_{h1}} \\ \vdots \\ \boldsymbol{u_{hN}} \end{pmatrix}, \mathbf{B}_h = \begin{pmatrix} \boldsymbol{b_{h1}} \\ \vdots \\ \boldsymbol{b_{hN}} \end{pmatrix} \text{ and } \mathbf{T}_h = \begin{pmatrix} \boldsymbol{t_{h1}} \\ \vdots \\ \boldsymbol{t_{hN}}, \end{pmatrix} \tag{2.19}$$

The above equations are in fact representing the balance of internal and external forces and can be also seen as follows:

$$\mathbf{R}(\boldsymbol{u_h}) = \mathbf{f}_{\text{int}}(\boldsymbol{u_h}) - \mathbf{f}_{\text{ext}} = \mathbf{0}, \tag{2.20}$$

It is important to note that the stiffness matrix in Eq.(2.18) is valid for small displacements. However, in many cases, the stiffness matrix can't be explicitly defined due to material nonlinearity (the relationship between stress and strain) or geometrical nonlinearity (the connection between strain and displacement). As a result, the weak formulation is addressed

iteratively through the Newton-Raphson algorithm. In this process, $\mathbf{K}$ is substituted with the corresponding tangent stiffness matrix.

2.1.5 Conclusion of the chapter

In this chapter, we introduced the fundamentals of solid mechanics and provided the formulation for non-linear elasticity. We also introduced the finite element method, a well-regarded numerical approach for solving partial differential equations. We generate several non-linear finite element datasets to train the deep learning surrogate models proposed in this thesis. In the subsequent chapter, we will shift our focus toward the understanding of deep learning techniques, thereby forging a bridge between the realms of traditional numerical modeling and the cutting-edge world of artificial intelligence.

Chapter 3

The Vast Field of Deep learning

Deep learning (DL) falls under the umbrella of machine learning methods, which have revolutionised several fields including scientific computing, computer vision, robotics, and many more. In particular, DL techniques have outperformed other machine learning methods when facing large amounts of data. DL also makes problem-solving much easier by providing end-to-end pipelines, because it completely automates feature engineering, which was previously a manual and time-consuming process in other machine-learning approaches.

DL is based on the so-called neural networks (NNs), whose roots can be traced to the late 1950s [Rosenblatt, 1958]. At that time, the available computational power was limited, which hindered the development of NNs. Later, Rumelhart et al. [1986] proposed the backpropagation algorithm to train multi-layer neural networks, which laid the foundation for the current state-of-the-art deep learning methodologies. In the 21st century, the exponential growth of technological advancements has led to an unprecedented increase in the amount of data. That, combined with the rapid growth in computational resources has made DL an important technique to tackle problems in a wide variety of domains.

3.1 Artificial Neural Networks

Artificial neural networks (ANNs) are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another. In simple terms, ANNs are a sequence of transformations applied on the inputs, that are capable of finding highly nonlinear underlying relationships in a given data set. According to the universal approximation theorem [Hornik, 1991], a feed-forward

neural network composed of artificial neurons has the capability to approximate any continuous real-valued function on subsets of $\mathbb{R}^n$.

ANNs consist of layers of nodes, which include an input layer, one or more hidden layers, and an output layer, see Figure 3.1. Each node, which can also be referred to as an artificial neuron, is linked to another and has an associated weight. When a node is activated, it undergoes an activation function ($\mathcal{A}$), and the resulting output data is transmitted to the next layer of the network. In this way, ANNs are capable of learning and processing complex data sets.

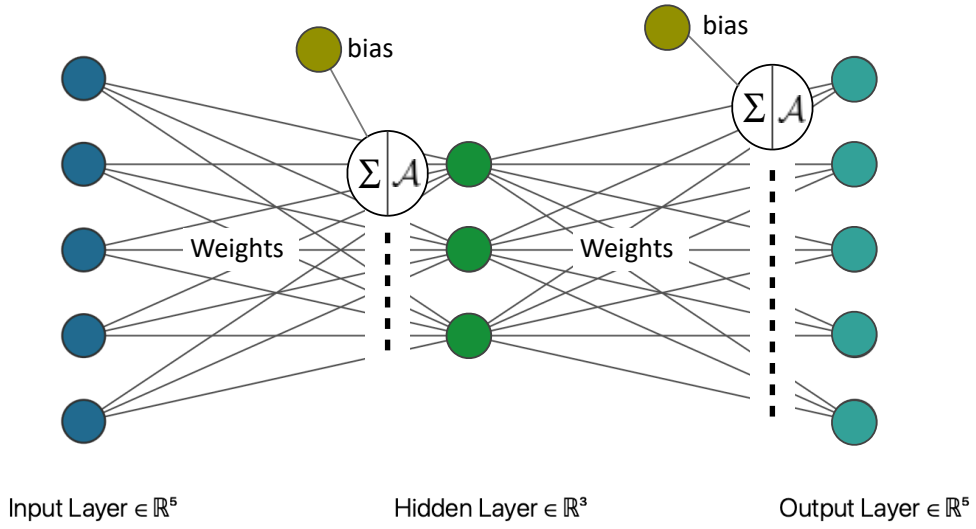


Fig. 3.1 A feed forward neural network with a single hidden layer. Weights and bias together constitute the parameters of the network.

As depicted in the diagram presented in Figure 3.1, when a specific input is fed into the artificial neural network, a weighted sum is calculated. This sum is then combined with bias values and processed through an activation function ($\mathcal{A}$), resulting in an output. The activation function, ($\mathcal{A}$), is responsible for incorporating non-linearity during the transformation of a layer to the next layer. Table 3.1 summarises some important activation functions used for ANNs. We will formally define the transformation expression in the following section.

Choosing activation function:
The choice of activation function in a neural network depends on the specific problem being addressed and the architecture of the network. This thesis mainly focuses on regression problems on the datasets relevant to the mechanics of materials, hence we will focus on activation functions relevant to regression only. In the early days, the sigmoid function was a common choice for the hidden layers in different types of NNs. However, it has a slower convergence rate thus making it computationally expensive. Sigmoid also suffers from the vanishing gradient problem during the backpropagation step. Similar issues occur for the hyperbolic tangent activation function.

ReLU has emerged as a great alternative to both sigmoid and tanh activation functions. It is computationally inexpensive because the ReLU function has a fixed derivative (slope) for one linear component and a zero derivative for the other linear component (it is considered that the ReLU converges 6 times faster than sigmoid and tanh functions). However, as the ReLU function and its derivative are equal to 0 for negative values, it deactivates certain neurons, causing them to output a 0 value always. This limitation is overcome by the Leaky ReLU activation function, i.e., if the input is less than 0, the leaky ReLU function outputs a small negative value defined by a constant negative slope 'a'. Both ReLU and LeakyReLU can have exploding gradient issues because of large positive activation values. This issue is overcome by the ReLU6 activation function, which restricts the activation value to 6 on the positive side.

Note: For the output layer, always the linear activation function is used, which allows to keep the output values unbounded. Thus making it suitable to be used for regressions tasks.

Activation function	Formula
Linear	x
Rectified Linear Unit (ReLU)	$\max(0, x)$
Leaky ReLU	$\begin{cases} x & x \geq 0 \\ ax & x < 0 \text{ (default } a = 0.01) \end{cases}$
Sigmoid	$\frac{1}{1+e^{-x}}$
Hyperbolic tangent (tanh)	$\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

Table 3.1 Neural network activation functions relevant for the scope of this thesis.

3.2 Types of neural networks

This section will introduce different types of neural networks relevant to this thesis.

3.2.1 Fully connected networks

A fully connected neural network is a type of artificial neural network (ANN) in which all neurons in one layer are connected to the neurons in the next layer Rumelhart et al. [1986]. This architecture is illustrated in Figure 3.1. A fully connected network with one or more hidden layers is commonly known as a multilayer perceptron (MLP). Formally, fully connected layer can be seen as a parametric transformation between the input ($\mathbf{d}^{l}$) and output ($\mathbf{d}^{l+1}$) layers as:

$$\mathbf{d}^{l+1} = \mathcal{A}(\boldsymbol{b}^{l+1} + \boldsymbol{k}^{l+1}\mathbf{d}^{l}) \tag{3.1}$$

where $\boldsymbol{k}^{l+1}$ and $\boldsymbol{b}^{l+1}$ are trainable weights and biases and $\mathcal{A}$ is the activation function. For each layer, $\boldsymbol{k}^{l+1}$ is the matrix of size (# of neurons in $\mathbf{d}^{l}$ $\times$ # of neurons in $\mathbf{d}^{l+1}$). Where as $\boldsymbol{b}^{l+1}$ is a one dimensional vector of size = no. of neurons in $\mathbf{d}^{l+1}$.

While fully connected networks are straightforward to implement, they come with following limitations that make them less desirable in certain scenarios:

1. They have a large number of trainable parameters, which can lead to overfitting of the network (in such cases regularisation techniques have to be used). Additionally, they require high memory allocation, especially for large-scale networks, which can be a computational bottleneck.
2. They scale very poorly with dimension of the input layer, hence they are not well suited for processing high dimensional data. This limitation has been discussed in detail in chapter 4.4.2.
3. They do not take into account the spatial co-relations between features in the input data, which can be crucial for certain tasks in mechanics, computer vision, and other fields. This is because they treat the input features as independent entities and do not consider their position or context within the input space.
4. Fully connected networks with large number of hidden layers can suffer from vanishing gradient problem. I.e., gradients propagated backward through the network can become extremely small and fail to update the weights efficiently.

Above mentioned limitations are significantly overcome by the next type of ANNs, convolutional neural networks.

Remark: Major part of this thesis focuses on developing DL frameworks that overcome above listed limitations. We focus on the development of DL techniques to tackle high dimensional problems in solid mechanics.

3.2.2 Convolutional neural networks

Convolutional Neural Networks (CNNs) are a class of ANNs, that is most commonly used for image analysis applications. Convolutional layers overcome the limitations of fully connected layers, especially when dealing with high-dimensional inputs. They require fewer parameters, as the same set of filter weights is applied at every spatial location in the input. This shared weight scheme results in weight sharing and translation invariance, which allows the network to recognize patterns in different parts of the image regardless of their location. CNNs are composed of two types of layers.

Convolution layer: The convolution layer operates by computing the convolution between a kernel and a patch of the grid input, see Figure 3.2. The kernel then slides by a hop defined by the stride 's', until it passes over the entire input. The identical size of output and input can be achieved by adding zero-padding around the input grid (which creates a margin of zeros). The padding ensures that the corner nodes are adequately processed by the convolution kernels, thus improving the accuracy and robustness of feature detection.

The formal expression for the convolution layer is described in Eq.(4.4).

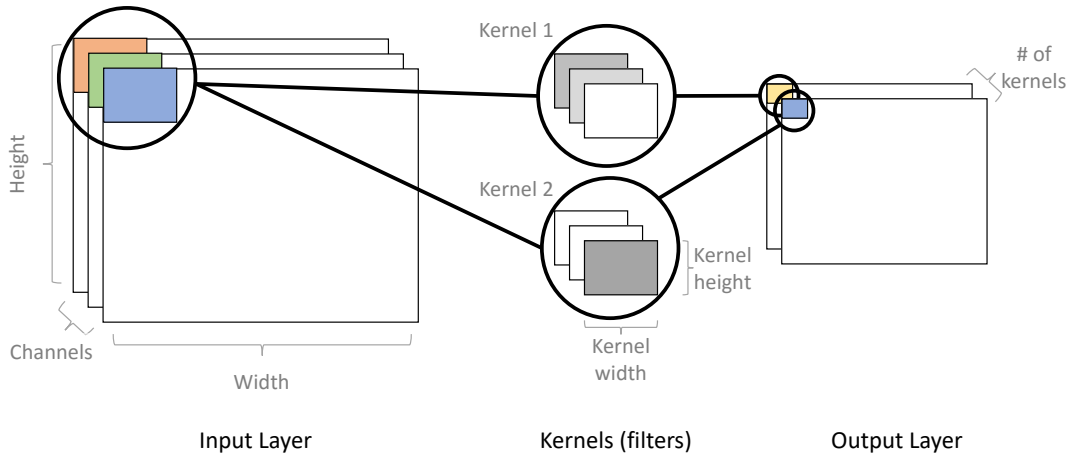


Fig. 3.2 CNN layer application on a 3-channel input, such as a RGB image. Convolution kernels (filters) are applied locally to transform the input layer into the output layer. Source: Gal [2016].

Pooling/unpooling layer: In CNNs, pooling layers are used to reduce the spatial dimensions of the input. The pooling operation involves partitioning the input (feature map) into non-overlapping regions. Pooling layers are used along with convolution layers, this is done to decrease the computational cost of subsequent layers and to prevent overfitting by extracting

the most important information from the input. The formal expression for the pooling layer is described in Eq.(4.5).

Unpooling operation is the exact opposite of pooling. Unpooling layers are used to recover the original spatial resolution after the pooling operation. The goal of the unpooling layer is to increase the spatial resolution of the feature maps while preserving their learned features. This is achieved by repeating the values of each element in the pooled feature map in a fixed neighborhood, in order to generate a higher-resolution feature map. The formal expression for the unpooling layer is described in Eq.(4.6). Both pooling and unpooling operations can be visualised from Figure 3.3.

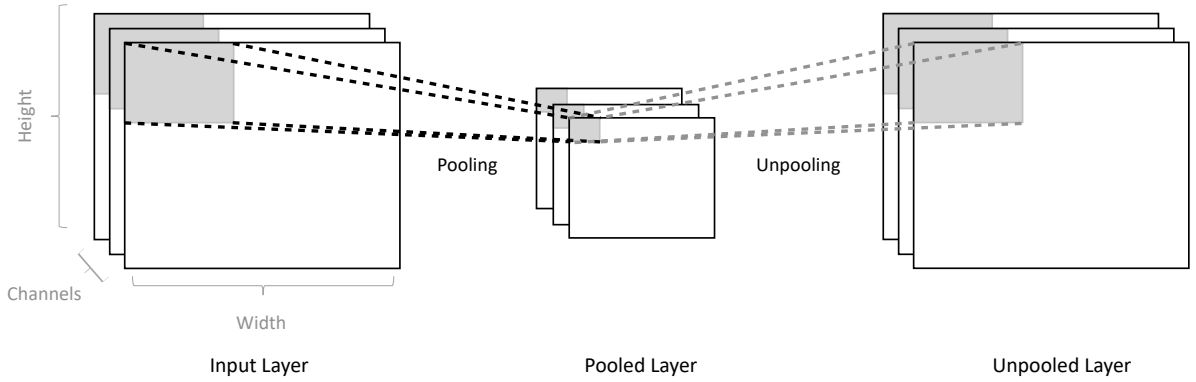


Fig. 3.3 Pooling over a 3-channel input. The pooling operation is performed channel-wise, hence the number of channels remains the same and only the width and height change for the pooled layer. Unpooling operation retrieves the original width and height dimensions.

Remark: CNN architectures work only for image-like grid inputs as they are originally developed to deal with image datasets. In recent years, they have been successfully used in various domains by transforming non-image data to image-like data [Sharma et al., 2019]. In particular, in this thesis we use CNNs to compute mesh deformations, hence they straightforwardly only work for structured meshes. To accommodate unstructured meshes for CNN architectures, strategies have been developed, as discussed in the introduction section of Chapter 5. However, these strategies have associated preprocessing costs and fail to perform efficiently for complex meshes. Also, these strategies do not provide an end-to-end deep learning framework. This motivates us to use graph neural networks and graph convolution networks, which are discussed in detail in Chapter 5.

3.3 Training of neural networks

Once the dataset is prepared and the type of neural network is chosen, the general training process of a neural network involves the following steps:

1. Parameter initialisation: Initialise weights and biases of the network.
2. Forward propagation: Pass the input data through the neural network to obtain a prediction.
3. Loss computation: Compute the difference between the predicted solution and expected output using the chosen loss function.
4. Backward propagation: Calculate Gradients of loss with respect to parameters of every layer.
5. Parameter update: Parameters are updated using gradient descent algorithms.
6. Repeat steps: Repeat 2-5 steps until desired convergence is achieved.

3.3.1 Initialisation of network parameters

Parameter initialization is an important step in training neural networks, as it can have a significant impact on the performance and convergence of the model. In general, biases are initialized with zero values, and weights are initialized with random numbers. Initializing all weights to zero would result in a constant derivative of the loss function with respect to weights, making the values of the weights unchanged in successive iterations and preventing the network from learning. Assigning low or high values to weights can also cause the problem of vanishing or exploding gradients. For preventing this issue, we usually stick to following the thumb of rules.

1. Mean of the activations should be zero for the layer.
2. The variance of the activations should stay the same across the layer

These two conditions ensure that neither activations nor gradients vanish/explode, thus ensuring efficient forward and backward propagation steps. All the implementations in this thesis are performed with Glorot initializer [Glorot and Bengio, 2010], which ensures both of the above conditions are met. The weights for each layer are randomly selected from a normal distribution

with a mean of zero and a small variance (which is inversely proportional to the number of units in the previous layer). This approach to weight initialization is widely used in deep learning and has been shown to be effective in promoting efficient and stable training of neural networks.

3.3.2 Forward propagation

Forward propagation, also known as forward pass, is the process of computing the output of a neural network for a given input. As the name suggests, the input data is fed in the forward direction through the network, whose parameters are initialised as described in the earlier section. Each hidden layer accepts the activations from the previous layer, processes it by applying an activation function to that layer, and passes it to the next layer.

To decrease memory requirements, input data is fed in small batches. Other advantages of feeding inputs in batches are discussed in detail in Section 3.3.5.

3.3.3 Loss computation

After the forward pass, the predicted output is compared with the expected output to compute the prediction error using a loss function. Depending on the considered problem, there are several options for the choice of the loss function to compute this prediction error. This thesis mainly focuses on regression types of problems and solves them in deterministic as well as probabilistic settings. All the deterministic networks implemented in this thesis are trained with a mean square loss function. The loss function used for probabilistic neural networks is described in Chapter 4.

For input dataset of feature-label pairs, $\mathcal{D} = (X, Y)$, the prediction loss '$\mathcal{L}_{\text{sq}}$', of neural network 'h' with $\boldsymbol{\theta} = \{\boldsymbol{k}, \boldsymbol{b}\}$ as its parameters, is computed as:

$$\mathcal{L}_{\text{sq}} = \mathbb{E}_{(X,Y)\sim\mathcal{D}}(\|h(X, \boldsymbol{\theta}) - Y\|_2^2) \tag{3.2}$$

Note: Enhancing loss function with physics information:

Since this thesis focuses on DL techniques for scientific simulations, there is an opportunity to improve the loss function using domain-specific knowledge. This approach is popularly termed Physics Informed Neural Networks (PINNs) [Raissi et al., 2019]. PINNs make use of differential equations in their loss function by taking higher-order derivatives of the output with respect to the input. These derivatives can be computed using automatic differentiation capabilities of popular DL frameworks such as TensorFlow, PyTorch [Abadi et al., 2015][Paszke et al., 2017]. They are then used to construct the residual and boundary conditions that should be approximated. Hence the loss function represented by Eq. (3.2) can be enhanced by adding $\mathcal{L}_{\text{residual}}$ and $\mathcal{L}_{\text{BC}}$, to get the loss function of PINNs as follows:

$$\mathcal{L}_{\text{PINN}} = \mathcal{L}_{\text{sq}} + \mathcal{L}_{\text{residual}} + \mathcal{L}_{\text{BC}} \tag{3.3}$$

However, this thesis does not place a significant emphasis on PINNs, and therefore we will not provide an in-depth description of them.

3.3.4 Backpropagation

Backpropagation, or backward propagation of errors, is an algorithm to effectively train a neural network through a method of chain rule [Rumelhart et al., 1986]. This method is useful for calculating the gradient of a loss function with respect to parameters of the neural network.

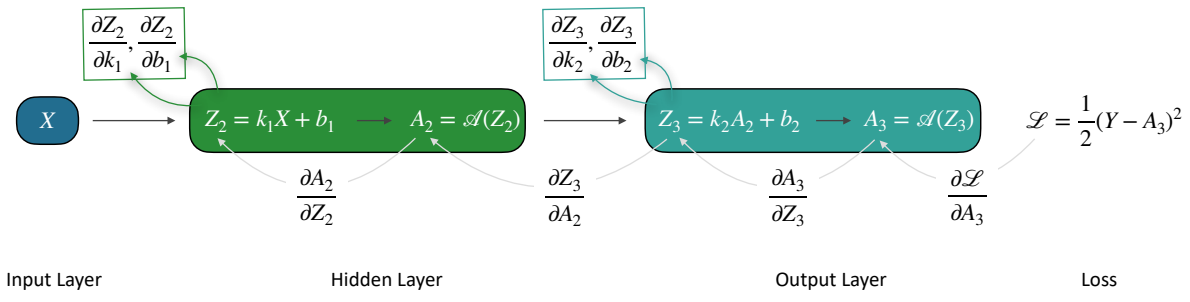


Fig. 3.4 Schematic of the backprop algorithm for a single hidden layer neural network with one neuron in each layer. The goal is to calculate gradients of loss with respect to the parameters of the network.

As illustrated in Figure 3.4, during the forward pass, the input data is fed through the network, and the output is computed. The loss function is then calculated based on the difference

between the predicted output and the actual output. During backpropagation, the error is propagated backward through the layers of the neural network, and the gradient of the loss function with respect to the parameters is calculated using the chain rule of calculus.

For the single neural network shown in Figure 3.4, gradients of loss with respect to weights (and similarly biases) of the layer can be computed as:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial k_2} &= \frac{\partial \mathcal{L}}{\partial A_3} \cdot \frac{\partial A_3}{\partial Z_3} \cdot \frac{\partial Z_3}{\partial k_2} \\ \frac{\partial \mathcal{L}}{\partial k_1} &= \frac{\partial \mathcal{L}}{\partial A_3} \cdot \frac{\partial A_3}{\partial Z_2} \cdot \frac{\partial Z_3}{\partial A_2} \cdot \frac{\partial A_2}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial k_1} \end{aligned} \tag{3.4}$$

After computing the gradients, the network's parameters are modified by utilizing variations of gradient descent algorithms.

3.3.5 Parameter optimization

Utilizing gradient descent to update parameters is a widely employed optimization strategy in the field of deep learning (Ruder, 2016).The loss function of the neural network is minimized with respect to the parameters (weights and biases) of the network. As described in Section 3.3.3, we represent parameters of the network as $\boldsymbol{\theta} = \{\boldsymbol{k}, \boldsymbol{b}\}$. The simplest form of gradient descent update for the parameters reads as:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha \cdot \nabla \mathcal{L}(\boldsymbol{\theta}) \tag{3.5}$$

where α is the learning rate and $\nabla \mathcal{L}(\boldsymbol{\theta})$ is the gradient of the loss function with respect to parameters.

The learning rate, α, is a hyperparameter that plays a crucial role in the minimization process. Its value determines the step size taken at each iteration while approaching the minimum, thereby influencing the speed of learning and convergence, refer Figure 3.5 . If the learning rate is too large, the optimization may diverge due to oscillations around the optimum. Conversely, if the learning rate is too small, the convergence speed may slow down, and the model may converge to a local minimum, resulting in overfitting to the training data. Typically,

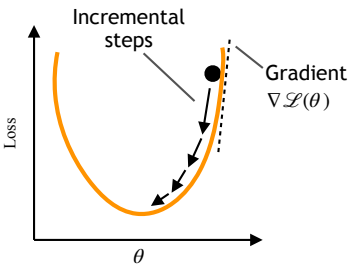


Fig. 3.5 Gradient descent update of a parameter.

it is recommended to start with a relatively large value and gradually decrease it during training iterations. However, there is no universal rule for selecting the learning rate.

Mini-batch gradient descent

Computing gradient over the entire training set at once is often termed batch gradient descent. This approach can be memory intensive, especially for large datasets. While batch gradient descent is effective for convex optimization problems, it may not be suitable for neural network training since the loss function often follows a non-convex pattern. Hence, parameters are susceptible to getting stuck in local minima and saddle points. To avoid the above-mentioned issues, the parameters of the network are updated using the gradient of the loss function computed over a small subset (batch) of the training data. This strategy is called mini-batch gradient descent. Mini-batch gradient descent can be parallelized across multiple processors or GPUs, making it faster than batch gradient descent.

Stochastic gradient descent

A particular case of mini-batch gradient descent, where network parameters are updated after each training example is called stochastic gradient descent (SGD). SGD is memory efficient when compared to batch gradient descent and can converge faster for nonconvex loss functions. SGD adds more fluctuations to the parameter update step, which reduces the chances of being stuck in a non-optimal local minimum. But at the same time, these fluctuations can make the optimization procedure unstable.

Overall, mini-batch gradient descent is a good choice for training deep learning models, especially when dealing with large datasets. It strikes a balance between the memory efficiency of stochastic gradient descent and the convergence guarantees of batch gradient descent.

Adam: Adaptive Moment Estimation

Adam [Kingma and Ba, 2014] combines two stochastic gradient descent approaches, Adaptive Gradients (AdaGrad, [Duchi et al., 2011]), and Root Mean Square Propagation (RMSProp, [Tieleman et al., 2012]). All the implementations presented in this thesis are carried out using Adam optimizer.

Adam facilitates the computation of learning rates for each parameter using the first and second moments of the gradient. The key idea behind Adam is to adapt the learning rate for each weight based on the history of gradients for that weight. This is done by maintaining an exponentially decaying average of past gradients and past squared gradients and using these averages to compute an adaptive learning rate for each weight. Additionally, Adam also includes a bias correction term to account for the fact that the estimates of the mean and variance of the gradients are biased towards zero at the beginning of training.

3.4 Conclusion

In this chapter, we introduced the basics of deep learning and various types of neural networks that are utilized in this thesis. Hence this chapter will be useful in better understanding advanced and novel deep learning frameworks presented in the upcoming chapters. The proposed deep learning frameworks are trained on synthetic non-linear finite element datasets, as explained in Chapter 2 and are implemented using Tensorflow [Abadi et al., 2015] and PyTorch [Paszke et al., 2019] libraries.

Chapter 4

Probabilistic Deep Learning for Real-Time Large Deformation Simulations

Abstract

For many novel applications, such as patient-specific computer-aided surgery, conventional solution techniques of the underlying nonlinear problems are usually computationally too expensive and are lacking information about how certain can we be about their predictions. In the present work, we propose a highly efficient deep-learning surrogate framework that is able to accurately predict the response of bodies undergoing large deformations in real-time. The surrogate model has a convolutional neural network architecture, called U-Net, which is trained with force-displacement data obtained with the finite element method. We propose deterministic and probabilistic versions of the framework. The probabilistic framework utilizes the Variational Bayes Inference approach and is able to capture all the uncertainties present in the data as well as in the deep-learning model. Based on several benchmark examples, we show the predictive capabilities of the framework and discuss its possible limitations.

This chapter is reproduced from: S. Deshpande, J. Lengiewicz, S.P.A. Bordas, *Probabilistic deep learning for real-time large deformation simulations.* Computer Methods in Applied Mechanics and Engineering, 2022, Volume 398, 115307, ISSN 0045-7825, https://doi.org/10.1016/j.cma.2022.115307

4.1 Introduction

Reliable and computationally efficient models are crucial in the design, optimization, or control for various application domains, including aerospace engineering, robotics, or bio-medicine. For instance the increasing interest in biomedical simulations [Cotin et al., 1999], [Delingette et al., 1999], [Courtecuisse et al., 2014a], [Wu et al., 2015], [Bui et al., 2018] may require having real-time responses. Finding convenient trade-offs between the accuracy and response time of such computational models is currently an active area of research in the context of digital twins, and is also one of the motivations for the research presented in this work.

When accuracy is important, the most general and widely used methodology in engineering for solving boundary value problems is the finite element method (FEM) [Zienkiewicz and Taylor, 1991]. This accuracy, especially when it comes to highly non-linear or history-dependent problems, may require a significant computational effort. The advancements in hardware development and software optimization enabled to some extent speeding up FEM computations, which involves specialized solution strategies to take advantage of high-performance computing architectures. A notable example is the class of Finite Element Tearing and Interconnecting (FETI) methods [Farhat and Roux, 1991]. In these methods, the global domain is partitioned into a set of disconnected sub-domains, which are computed in parallel on different processors/nodes. However, in many applications, it is not possible to meet real-time responses on the hardware available for industrial consumers. This is due to a limited number of available cores and a significant communication burden that deteriorates the overall time performance of such solution strategies.

There are various specialized FEM-based approaches to cut down the solution time at the cost of sacrificing the accuracy, see, e.g., [Marinkovic and Zehn, 2019]. An important class of such approaches is the model order reduction (MOR) methods, with Proper Orthogonal Decomposition (POD) being one of the notable examples. The general concept of POD, e.g., applied to a discretized FEM formulation, is to find a low dimension subspace in order to approximate the full space at an acceptable loss of accuracy. This potentially enables controlling the trade-off between accuracy and computation time. POD was adapted to work within the large-deformation regime, see, e.g., [Kerfriden et al., 2011a], which was for instance applied to simulate and control soft robotic arms [Goury and Duriez, 2018a] or to reduce computational costs in nonlinear fracture mechanics problems [Kerfriden et al., 2012]. However, the efficiency of POD deteriorates in high non-linear regimes since it relies on a linear combination of few basis vectors and thus oversimplifies the model [Bhattacharjee and Matouš, 2016]. Proper Generalised Decomposition (PGD) is another MOR technique, in which the solution of the complete problem is computed as a finite sum of separable functions. The compact solution, though not optimal, in general, provides a very light format to store the solution in the form of a meta-model,

thereby speeding up the solution times. [Niroomandi et al., 2009] implemented PGD based approach for computationally efficient simulations of hyper-elastic responses. However, the accuracy of PGD methods decreases when the separation of variables assumption cannot respect the problem to solve [Allier et al., 2015].

Importantly for the present work, we distinguish yet another family of FEM-based approaches, in which the expected speedups and approximation capabilities originate from underlying Deep Neural Networks (DNNs) with Deep Learning (DL) techniques used to train these networks. Generally, DL approaches make an important part of machine learning techniques and have allowed to solving highly complex problems that had eluded scientists for decades. In particular, DL-based methods have also been developed to efficiently solve problems in engineering [I. Goodfellow, 2016]. One of the popular approaches utilises the idea of the so-called Physics Informed Neural Networks (PINNs), in which both the data (either synthetic or experimental) and the assumed governing Partial Differential Equations (PDEs) are incorporated in the training phase; for early traces of these see, e.g., [Lagaris et al., 1998], [Lagaris et al., 2000], [McFall and Mahan, 2009], and for a recent study see, e.g., [Raissi et al., 2019],[Samaniego et al., 2020]. One of the benefits of this approach is that possibly much less data is needed for training, which can be an important factor for many data-driven applications. Note, however, that even if the physics is not explicitly enforced in the training phase (non-PINN case), it can still be recovered in the trained model by implicitly following a large amount of training data (synthetic or experimental). In the case when all training data are synthetically provided from FEM simulations, see, e.g., [Lorente et al., 2017], [Mendizabal et al., 2019b][Krokos et al., 2022a], we will refer to it as the *direct FEM-based approach*.

In this work, we propose a framework that falls into the above-mentioned class of direct FEM-based DNN approaches. The framework is based on a particular DNN architecture—the U-Net architecture [Ronneberger et al., 2015]—which in turn can be viewed as a type of Convolutional Neural Networks (CNNs); further explanations are provided later in this work. Originally, the U-Net architecture has been developed for the purpose of biomedical image segmentation, however, it turned out to be also suitable for other applications. In particular, the present work is inspired by the recent results of [Mendizabal et al., 2019b], in which the authors demonstrate quite accurate real-time non-linear force-displacement predictions done by U-Nets trained on FEM-based data. It has been noticed, see [He and Xu, 2019], [Wang et al., 2020b], that this good accuracy is not accidental but can be possibly linked to the strong resemblance of U-Net architectures and multi grid solution schemes [Brenner and Scott, 2008]. Such a point of view makes the U-Net approach less of a brute-force black-box approximation and more of a suited solution scheme, which makes this line of research very promising.

An equally important aspect that is studied in the present work is the capabilities of DNNs to quantify uncertainties. This is motivated by the fact that in many real-life applications, such

as surgical simulations [Bui et al., 2018] or autonomous driving [McAllister et al., 2017], it is crucial to produce reliable uncertainty estimates in addition to the predictions. Otherwise, model predictions can lead to harmful consequences in these critical tasks. Deterministic neural networks are usually certain about their predictions, and this overconfidence is especially evident when facing data far from the training set. Generally, uncertainties can be categorised as those associated with the misfit of neural network models (epistemic uncertainties) and those that refer to the noise in training data (aleatoric uncertainties)[Gawlikowski et al., 2021][Kendall and Gal, 2017b]. Uncertainties can either fall within or outside the training data region (interpolated and extrapolated regions, respectively). Within the data region, the prediction uncertainties are caused both by noisy data and by the model misfit, which can be captured and quantified in many ways, the Maximum Likelihood Estimation (MLE) method is one of the most straightforward [Kendall and Gal, 2017a] to do so. However, for the extrapolated region we have no data support, therefore, no direct quantification can be done there. What we can only reasonably assume is that the uncertainty should generally increase when moving away from the data region. To achieve this, in this work, we will extend the idea proposed in [Blundell et al., 2015][Gal, 2016][Duerr et al., 2020] which relies on converting a neural network to its stochastic counterpart by replacing discrete parameters with probability distributions and using a special training technique that is based on Bayesian Inference.

Bayesian approaches have been already considered in the context of FEM models; for instance in [Hauseux et al., 2018] uncertainties are quantified for hyperelastic soft tissues by incorporating stochastic parameters in the FEM model, while in [Rappel et al., 2020][Zeraatpisheh et al., 2021] a thorough tutorial on using Bayesian Inference to solid mechanics problems is provided. In the present work, however, we focus on Bayesian Inference and related Bayesian Neural Networks (BNNs). Here, in the context of deep networks with millions of parameters, the application of widely used Markov Chain Monte Carlo (MCMC) type of methods would be computationally intractable. For that reason, in this work we use one of the well-known methods for approximate Bayesian inference—Variational Inference (VI) [Graves, 2011]—in which an approximate distribution is used instead of the true Bayesian posterior over model parameters.

To sum up, the scope of the present work is to study the applicability of U-Net deep learning architectures to performing real-time predictions for large-deformation problems with uncertainties, where synthetic training data is provided by FEM simulations. The organisation of the paper is the following. In Section 5.2 we present the general methodology applied to a deterministic version of U-Net. In Section 4.3 we introduce the extension of the framework to the Variational Bayesian Inference case. Then, in Section 5.3, an extensive study of the proposed framework is performed, which is based on several 2D and 3D benchmark examples. The conclusions and future research directions are outlined in Section 4.5.

4.2 General FEM-based U-Net Methodology

The proposed approach can be divided into two main phases. The first phase involves finite element simulations to prepare necessary datasets. The second phase consists of building and training deterministic/probabilistic U-Net deep neural networks, using the training datasets generated in the first phase. The trained U-Nets are then used as surrogate models.

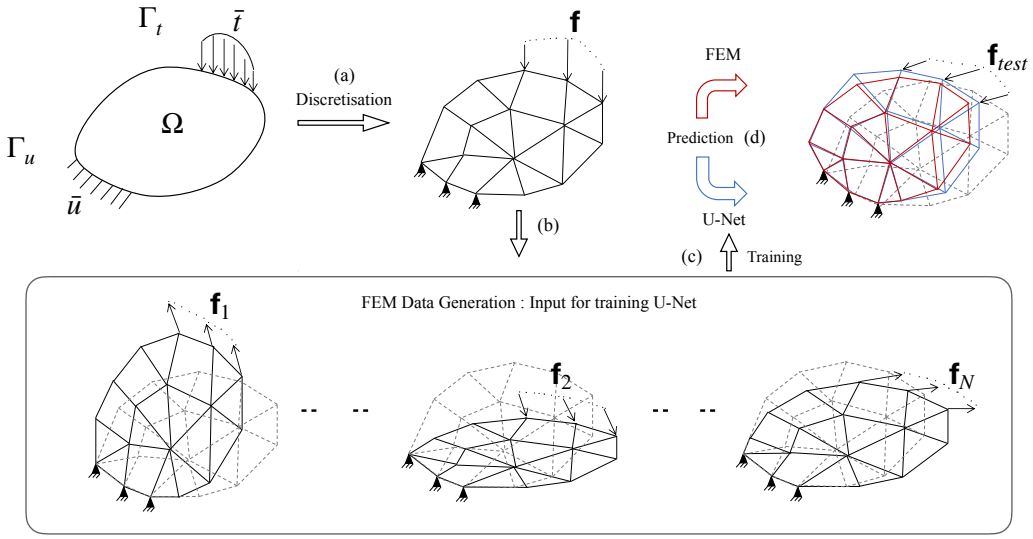


Fig. 4.1 Schematic of the framework (a) Continuum problem is discretized by FEM mesh (b) Training/testing examples are generated by applying random point forces on Neumann boundary. (c) The U-Net is trained on the generated dataset (d) Trained U-Net predicts the deformation for a test force (blue mesh). FEM solution (red) is used for cross-validation. Gray dashed meshes indicate undeformed configurations.

4.2.1 FEM-Based Deep Learning Approach

As a problem to be solved, we consider the boundary value problem of a hyperelastic solid, with a constant prescribed Dirichlet BC. Large deformations exhibit non-linear stress-strain behavior when applied with external forces, hence one needs to consider hyper-elastic constitutive laws for simulating such systems.

Consider a boundary value problem in continuum mechanics in the domain Ω, Dirichlet and Neumann boundary conditions are applied on Γ_D, Γ_N respectively. Neglecting the body forces, the virtual work principle for nonlinear elastostatic equation reads

$$\int_{\Omega} \mathbf{P}(\boldsymbol{u}) \cdot \nabla \delta \boldsymbol{u} \, dV - \int_{\Gamma_t} \bar{\boldsymbol{t}} \cdot \delta \boldsymbol{u} \, dS = 0 \qquad \forall \delta \boldsymbol{u}, \tag{4.1}$$

where $\boldsymbol{u}$ and $\delta\boldsymbol{u}$ belong to appropriate functional spaces, $\boldsymbol{u} = \bar{\boldsymbol{u}}$ and $\delta\boldsymbol{u} = \mathbf{0}$ on Γ_u, and $\mathbf{P}(\boldsymbol{u})$ is the first Piola-Kirchhoff stress tensor. The required constitutive relationship will be defined through the (hyper-)elastic strain energy potential $W(\boldsymbol{F})$ as

$$\mathbf{P}(\boldsymbol{F}) = \frac{\partial W(\boldsymbol{F})}{\partial \boldsymbol{F}} \tag{4.2}$$

where $\boldsymbol{F} = \boldsymbol{I} + \nabla\boldsymbol{u}$ is the deformation gradient tensor. (The particular form of W, used in this work, will be presented in Section 4.4.1.)

After standard FE discretization, the problem expressed by Eq. (7.14) will take the form of system of non-linear equations

$$\mathbf{R}(\mathbf{u}) = \mathbf{f}_{\text{int}}(\mathbf{u}) - \mathbf{f}_{\text{ext}} = \mathbf{0}, \tag{4.3}$$

which expresses the balance between external and internal nodal forces. By solving the system of equations (5.16) (e.g., with the Newton-Raphson method) for a given external force vector $\mathbf{f}_{\text{ext}} = \mathbf{f}$ we obtain a solution in the form of nodal displacements $\mathbf{u}$.

External forces can be applied to a selected region on the surface described by Γ_t. For the current framework, we consider a single FE discretization of a given domain Ω. As described in Figure 4.1, we apply a prescribed family of load distributions, given by vectors of force $\mathbf{f}_i$ on the nodes present in Γ_t to generate nodal displacements $\mathbf{u}_i$. This creates a dataset $\mathcal{D} = (\mathbf{f}_i, \mathbf{u}_i)_{i=1}^N$ of corresponding pairs of nodal force and displacement vectors, which is then used as input to train DNNs. Thus the input of the neural network is a vector of nodal forces, and the predicted output is a vector of nodal displacements.

4.2.2 U-Net deep neural network architecture

As motivated in the introduction, we use a specific family of CNN, U-Nets [Ronneberger et al., 2015]. They owe their name to a specific U-shape of the architecture diagram, e.g., see Figure 4.2, which is an effect of applying cascades of max pooling operations, followed by cascades of upsampling operations.

At its input, the layer $\boldsymbol{d}^0$, the U-Net network, $\mathcal{U}$, accepts the vector of external forces, $\mathbf{f}$, in the original mesh format $n_x \times n_y \times n_z \times 3$, where n_x, n_y, n_z are the dimensions of the structured 3D mesh (for 2D problems the format is $n_x \times n_y \times 2$). The output displacements, in the layer $\boldsymbol{d}^L$, are in the analogous mesh format. The model parameters $\boldsymbol{\theta}_{\text{det}}$ are all weights (kernel $\boldsymbol{k}$ and biases $\boldsymbol{b}$) of the neural network (see below for details).

For the sake of clarity, below we will introduce the idea for a 2D case only, which can be straightforwardly transformed into a 3D case. For 2D problems, we use architecture as described in Figure 4.2. To a given input mesh, we first add double zero padding in each spatial direction (this is done to avoid the loss of information of corner nodes). There are two layers in each encoding and decoding phase. To the padded input, we apply two convolutions with batch normalisation followed by rectified linear unit activation (ReLU), see Eq. (4.4). In the encoding phase, at each step, 2×2 Max Pooling layers are applied which decrease spatial dimensions by half, and we multiply the number of channels by 2 (c=128 for the first level). In decoding steps, 2×2 Up-sampling layers are applied which increases the spatial dimension by two and the number of channels is halved. At each level, encoding and decoding outputs are concatenated together. In the end, 1×1 convolution is applied with linear activation to get the output.

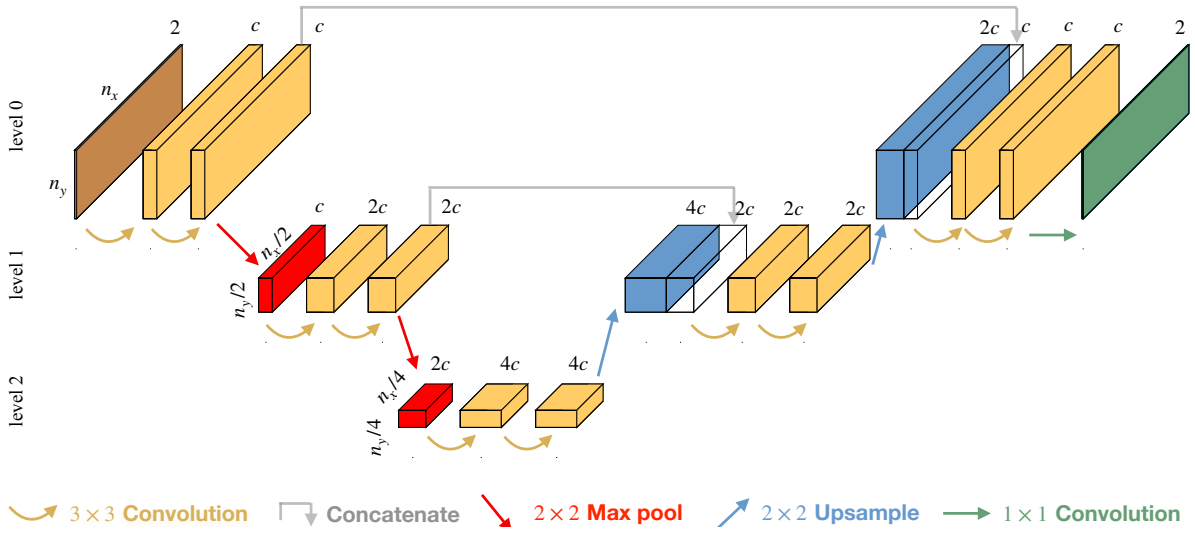


Fig. 4.2 A schematic of exemplary U-Net architecture for 2D domains, (n_x, n_y) stand for number of nodes in x, y direction of 2D domain. Boxes indicate U-Net layers (colors indicate different types of layers), first step contains 'c' channels.

In order to better understand the idea of convolution operator, the $3 \times 3 \times c$ operator in 2D on the first U-Net level can be imagined as a filter window that is applied to a $n_x \times n_y \times c$ mesh. For a two-dimensional domain, the input tensor is of the dimension $n_x \times n_y \times 2$, which is identical to the FEM mesh. Here 2 stands for the number of channels of input convolutions, n_x, n_y stands for the number of nodes in the x, y direction, respectively. To best leverage the CNNs, we keep x & y-dofs in separate channels. We operate a convolutional filter on a local region and then is slid along spatial directions x, y with a stride of 1 as illustrated in Figure 4.3.

An example of a non-restrictive convolution operation (in 2D), between subsequent U-Net layers l and $l+1$, for the filter size $3 \times 3 \times c^n$ reads

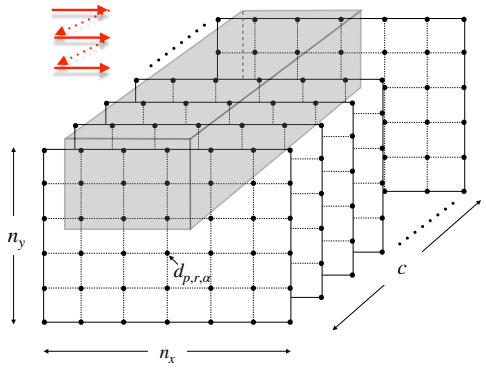


Fig. 4.3 x and y dofs are stored in different channels, 3×3 convolutional filter (gray) acts locally along the channel direction and it slides along with the step of 1 in both horizontal and vertical directions (red).

$$d^{l+1}_{p,r,\beta} = \mathcal{A}\left(b^{l+1}_{\beta} + \sum_{i=1,}^{3} \sum_{j=1,}^{3} \sum_{\alpha=1}^{c^l} d^{l}_{p+i-2,r+j-2,\alpha} k^{l+1}_{i,j,\alpha,\beta} \right), \tag{4.4}$$

where $d^{l+1}_{p,r,\beta}$ are neural network nodes at layer $l+1$, the weights $k^{l+1}_{i,j,\alpha,\beta}$ are parameters of the convolution operator, the weights b^{l+1}_{β} are biases at a layer $l+1$, and $\mathcal{A}(\cdot)$ is an activation function (ReLU). Indices i, j stand for the components of the covolutional filter (3×3 in our case) and the indices p, r are related to nodes in a 2D grid of the output layers. They directly refer to the underlying structured FEM mesh. In our case, we add zero pad in each dimension of input before applying the convolution, to ensure the same size of the input and output. Indices $1 \leq \alpha \leq c^l$ and $1 \leq \beta \leq c^{l+1}$ represent the channel number. Note that the number of channels in subsequent layers need not be equal, i.e., in general $c^l \neq c^{l+1}$. Note also that in the first and in the last layer, the number of channels correspond to the spatial dimension of the problem (2D or 3D).

Max-pooling operation is responsible for reducing the spatial dimensions of its input, channel dimensions are unaffected by it. It reads as follows

$$d^{l+1}_{p,r,\alpha} = \max_{\substack{2p-1 \leqslant i \leqslant 2p \\ 2r-1 \leqslant j \leqslant 2r}} d^{l}_{i,j,\alpha} \tag{4.5}$$

Upsampling operator can be seen as the reverse of max pooling, it increases the spatial dimensions of the input without affecting the channel dimension. As showed in Figure 4.2, in the decoder phase, outputs of up-sampling are concatenated with respective layers from the encoder phase of the U-Net (in case of symmetric U-Nets, the l-th layer is concatenated with

the $(L-l-2)$-th layer, where L is the index of output layer in a U-Net). The up-sampling with concatenation read

$$d^{l+1}_{p,r,\alpha} = \begin{cases} d^{l}_{\lfloor p/2 \rfloor+1,\lfloor p/2 \rfloor+1,\alpha} & , \quad 1 \leqslant \alpha \leqslant c^l \\ d^{L-l-2}_{p,r,(\alpha-c^l)} & \quad c^l+1 \leqslant \alpha \leqslant c^l + c^{L-l-2} \end{cases} \tag{4.6}$$

The final 1×1 convolution operation reads

$$d^{L}_{p,r,\beta} = b^L_\beta + \sum_{\alpha=1}^{c^{L-1}} d^{L-1}_{p,r,\alpha} k^L_{\alpha,\beta}, \qquad \beta = 1, 2. \tag{4.7}$$

For the 3D version of U-Nets, the operations given by Equations (4.4)-(4.7) are straightforwardly extended by one additional dimension. This results in adding one index to nodes' and biases' specifications, d and b, respectively, and two indexes to 3×3 convolution weights, k. For 2D/3D cases, trainable parameters of the deterministic U-Net are

$$\boldsymbol{\theta}_{\text{det}} = \bigcup_{l=1}^{L} \{\mathbf{k}^l, \mathbf{b}^l\}. \tag{4.8}$$

$\mathcal{U}(\mathbf{f}, \boldsymbol{\theta}^{\text{det}})$ is defined recursively (the forward propagation), starting from the input layer $\boldsymbol{d}^0 = \mathbf{f}$, then subsequently applying appropriate transformations given by one of Eqs. (4.4)-(4.6), and finally applying the transformation given by Eq. (4.7), see also Figure 4.2. Finally the prediction of the deterministic U-Net is

$$\mathcal{U}(\mathbf{f}, \boldsymbol{\theta}_{\text{det}}) = \mathbf{d}^L \tag{4.9}$$

For a given training dataset $\mathcal{D} = \{(\mathbf{f}_1, \mathbf{u}_1), ..., (\mathbf{f}_N, \mathbf{u}_N)\}$, the deterministic U-Net is trained by minimizing the following mean squared error loss function

$$\mathcal{L}_{\text{det}}(\mathcal{D}, \boldsymbol{\theta}_{\text{det}}) = \frac{1}{N} \sum_{i=1}^{N} \|\mathcal{U}(\mathbf{f}_i, \boldsymbol{\theta}_{\text{det}}) - \mathbf{u}_i\|_2^2 \tag{4.10}$$

which gives the optimal parameters

$$\boldsymbol{\theta}^*_{\text{det}} = \arg\min_{\boldsymbol{\theta}_{\text{det}}} \mathcal{L}_{\text{det}}(\mathcal{D}, \boldsymbol{\theta}_{\text{det}}). \tag{4.11}$$

A particular training strategy, used in this work, is introduced in Section 4.4.1.

Remark: The current framework (as well as all other neural-network based approaches mentioned in the introduction) requires retraining a network when changing the FE discretization. Recently proposed operator-based learning approaches, called neural operators [Lu et al., 2021][Li et al., 2021][Wen et al., 2022], promise to overcome this disadvantage.

4.3 Probabilistic U-Net framework

There are various sources of uncertainties linked to engineering systems. These can be broadly categorised as noises in the observation (aleatoric uncertainty or data uncertainty) and uncertainty in the assumption of our model (epistemic uncertainty or model uncertainty) [Kendall and Gal, 2017b]. Aleatoric uncertainty is inherent to the data and it can't be reduced, whereas epistemic uncertainty can be reduced by providing more training data. An important part of epistemic uncertainty is being able to tell that the more data we have, the more certain we are about the predictions. This uncertainty is expected to be high while doing predictions on inputs away from the training region. Deterministic U-Nets explained in the Section 4.2.2 fail to account for these uncertainties. In order to capture these effects, in this work, we propose the Bayesian approach, which is introduced in this section.

4.3.1 Variational Bayesian Inference

Bayesian methods provide an approach to quantify the uncertainty of prediction in deep neural networks. To do so, in this framework, we replace the deterministic parameters with probability distributions [Blundell et al., 2015]. Originally this idea was only used to prevent overfitting, but was observed to also increase the variability of outputs in the extrapolated region. In order to suitably control the level of introduced perturbations to parameters, we use Bayesian Inference. This gives us a formal theoretical framework, allowing us to apply suitable computational techniques (VI) to efficiently train networks and predict results. The input of a network remains the same, but some of the model parameters become probability distributions (stochastic), and for that reason also the output of the network becomes a distribution over possible outputs. As per the standard Bayesian approach, we specify a prior distribution $P(\boldsymbol{w})$ over parameters, we consider $\mathcal{D} = \{(\mathbf{f}_1, \mathbf{u}_1), ..., (\mathbf{f}_N, \mathbf{u}_N)\}$ as the given training dataset, then for a new vector $\mathbf{f}^{\text{test}}$, prediction $\mathbf{u}^{\text{test}}$ is given by

$$P(\mathbf{u}^{\text{test}}|\mathbf{f}^{\text{test}}, \mathcal{D}) = \int P(\mathbf{u}^{\text{test}}|\mathbf{f}^{\text{test}}, \boldsymbol{w}) P(\boldsymbol{w}|\mathcal{D}) \mathrm{d}\boldsymbol{w} \tag{4.12}$$

The Bayesian inference involves the calculation of true parameter posterior $P(\boldsymbol{w}|\mathcal{D})$ conditioned over the training data. As mentioned in the introduction, variational inference (VI) is used to approximate true posterior densities in bayesian neural networks [Graves, 2011], i.e. true posterior is approximated by a variational posterior $q(\boldsymbol{w}|\boldsymbol{\theta})$, parametrized by $\boldsymbol{\theta}$. Variational learning finds optimal parameters $\boldsymbol{\theta}^*$ by minimizing the Kullback-Leibler divergence (KL-divergence) between true and variational posteriors, as per the following equation:

$$\begin{aligned}\boldsymbol{\theta}^* &= \arg\min_{\boldsymbol{\theta}} \mathrm{KL}[q(\boldsymbol{w}|\boldsymbol{\theta})||P(\boldsymbol{w}|\mathcal{D})] \\ &= \arg\min_{\boldsymbol{\theta}} \int q(\boldsymbol{w}|\boldsymbol{\theta}) \log \frac{q(\boldsymbol{w}|\boldsymbol{\theta})}{P(\boldsymbol{w})P(\mathcal{D}|\boldsymbol{w})} dw \\ &= \arg\min_{\boldsymbol{\theta}} \mathrm{KL}[q(\boldsymbol{w}|\boldsymbol{\theta})||P(\boldsymbol{w})] - \mathbb{E}_{q(\boldsymbol{w}|\boldsymbol{\theta})}[\log P(\mathcal{D}|\boldsymbol{w})].\end{aligned} \tag{4.13}$$

The resulting cost function is the loss function for training the neural network. It consists of two parts, first is the prior dependent part represented by the KL-divergence term, it can be referred to as model complexity cost; it tells how close approximate posteriors are to priors. And later is the data-dependent part which can be referred to as likelihood cost, it tells how well the network fits the data. Bayesian neural networks with prior distributions are well known to induce regularisation effect [Neal, 1996]; in particular, using Gaussian priors is equivalent to weight decay (L_2 regularization) [Vladimirova et al., 2019].

During the forward pass, weights are sampled from the variational posterior $q(\boldsymbol{w}|\boldsymbol{\theta})$.Now, during the backpropagation, the issue is that one cannot get a gradient of the sampled points, because the sampling operation cannot be differentiated. To avoid this issue, the following reparameterization trick is used [Kingma et al., 2015]. A sampled weight, w, is obtained by sampling a parameter-free distribution (the unit Gaussian), which is then scaled by a standard deviation $\boldsymbol{\sigma}$ and shifted by a mean $\boldsymbol{\mu}$. We parameterise the standard deviation point-wise as $\boldsymbol{\sigma} = \log(1 + \exp(\boldsymbol{\rho}))$ to have $\boldsymbol{\sigma}$ always non-negative. Thus the sample is $\boldsymbol{w} = \boldsymbol{\mu} + \log(1 + \exp(\boldsymbol{\rho})) \odot \boldsymbol{\epsilon}$, where $\odot$ is point-wise multiplication, $\boldsymbol{\epsilon}$ is drawn from $\boldsymbol{\mathcal{N}}(\mathbf{0}, \boldsymbol{I})$. Hence the variational posterior parameters are $\boldsymbol{\theta} = (\boldsymbol{\mu}, \boldsymbol{\rho})$ [Blundell et al., 2015]. In our framework we use Gaussian priors with its parameters being $(\boldsymbol{\mu_p}, \boldsymbol{\sigma_p})$. For the reasons mentioned in Section 4.3.3, we include prior means $(\boldsymbol{\mu_p})$ in training procedure.

4.3.2 Maximum likelihood estimation

In addition to the Bayesian approach, we also introduce the Maximum Likelihood Estimation (MLE) method—a popular frequentist approach. We do it to compare both methods in their capabilities to quantify uncertainties. Parameters of the MLE model are deterministic, but we take the double number of outputs compared to the deterministic counterpart. They stand

for means and non-constant (heteroscedastic) standard deviations [Duerr et al., 2020], thus yielding distributions as the outputs. These non-constant standard deviations can only capture the noises in the data, MLE inherently fails to account for uncertainties in the extrapolated region. The loss function for MLE can be recovered from Equation (4.13) by removing the KL divergence part (since we don't have distributions on parameters of the MLE model), and the MLE model is trained on the Gaussian negative log-likelihood loss:

$$\boldsymbol{\theta}^*_{\text{MLE}} = \arg\min_{\boldsymbol{\theta}_{\text{MLE}}} -\log P(\mathcal{D}|\boldsymbol{\theta}_{\text{MLE}}). \tag{4.14}$$

4.3.3 Trainable priors: Use of Empirical Bayes

Since NN parameters are latent variables of the model, it is very difficult to make a proper choice of priors. If one sets the priors far from their true values, then the posterior may be unduly affected by such choice. To overcome this, we incorporate Empirical Bayes (EB) approach [Carlin and Louis, 1997], a method that uses the observed data to estimate the prior hyperparameters. In our approach, in the training phase, we update the prior means, keeping the prior standard deviation constant. Hence we minimize the loss function by also considering gradients with respect to the prior means. This treatment enables us to obtain a good fit to the data, while at the same time giving high prediction uncertainties in the region where little or no data is available.

4.3.4 Loss functions for probabilistic U-Net

We modify the deterministic U-Net architectures by replacing their layers with probabilistic layers, as a result, the output of the network is a probability distribution itself. We choose Gaussian distributions to represent priors and approximate posteriors of probabilistic layers. For the Bayesian U-Net, we use loss function as given in Eq. (4.13). Expectations of the Eq. (4.13) are approximated by $\mathcal{M}$ Monte Carlo samples drawn from the approximate posterior $q(\boldsymbol{w}|\boldsymbol{\theta})$ as referred below

$$\begin{aligned} \mathcal{L}_{\text{VB}} = &\ \text{KL}[q(\boldsymbol{w}|\boldsymbol{\theta})||P(\boldsymbol{w})] - \mathbb{E}_{q(\boldsymbol{w}|\boldsymbol{\theta})}[\log P(\mathcal{D}|\boldsymbol{w})] \\ \approx &\ \sum_{i=1}^{\mathcal{M}} \log q(\boldsymbol{w}^{(i)}|\boldsymbol{\theta}) - \log P(\boldsymbol{w}^{(i)}) - \log P(\mathcal{D}|\boldsymbol{w}^{(i)}) \end{aligned} \tag{4.15}$$

If we substitute Gaussian probability density functions, the expression in the RHS of Eq.(4.15) turns out to be as given in Eq. (4.16). We consider '$\mathcal{G}$' probabilistic parameters (Gaussian distributions) for our Bayesian U-Net, where every distribution is parameterised by its mean and standard deviation values. Since the standard deviations $\boldsymbol{\sigma}$ must be positive, we first train the network on untransformed standard deviations $\boldsymbol{\rho}$ which are later transformed to $\boldsymbol{\sigma}$ through soft-plus function. Also, for the reasons discussed in Section 4.3.3, we involve prior means, $\boldsymbol{\mu_p}$, in the training procedure as well. Hence parameters to be learned in the training procedure are $\boldsymbol{\theta}_{\mathrm{VB}} = (\boldsymbol{\theta}, \boldsymbol{\mu_p})$. Finally, the loss function for the Variational Bayes is given as follows

$$\begin{aligned}\mathcal{L}_{\mathrm{VB}}(\mathcal{D}, \boldsymbol{\theta}_{\mathrm{VB}}) \approx \sum_{i=1}^{\mathcal{M}} \Bigg[\sum_{j=1}^{\mathcal{G}} \Bigg(&- \log\left(\sqrt{2\pi}\ \sigma_j^{(i)}\right) - \frac{\left(w_j^{(i)} - \mu_j^{(i)}\right)^2}{2\left(\sigma_j^{(i)}\right)^2} + \log\left(\sqrt{2\pi}\ \sigma_p\right) + \frac{\left(w_j^{(i)} - (\mu_p)_j^{(i)}\right)^2}{2\sigma_p^2} \Bigg) \\ &- \sum_{k=1}^{N} \sum_{l=1}^{\mathcal{F}} \left(- \log\left(\sqrt{2\pi}\ d_\sigma^{(l)}(\mathbf{f}^{(k)}, \boldsymbol{w}^{(i)})\right) - \frac{\left(u_l^{(k)} - d_\mu^{(l)}(\mathbf{f}^{(k)}, \boldsymbol{w}^{(i)})\right)^2}{2\left(d_\sigma^{(l)}(\mathbf{f}^{(k)}, \boldsymbol{w}^{(i)})\right)^2} \right) \Bigg]\end{aligned} \tag{4.16}$$

where

$$\begin{aligned} d_\sigma^{(l)}(\mathbf{f}^{(k)}, \boldsymbol{w}^{(i)}) &= \log(1 + \exp(d_\rho^{(l)}(\mathbf{f}^{(k)}, \boldsymbol{w}^{(i)}))), \\ w_j^{(i)} &= \mu_j^{(i)} + \sigma_j^{(i)} \epsilon_j^{(i)}, \qquad \epsilon_j^{(i)} \ \sim\ \mathcal{N}(0,1), \\ \sigma_j^{(i)} &= \log(1 + \exp(\rho_j^{(i)})). \end{aligned} \tag{4.17}$$

$(\boldsymbol{d_\mu}(\mathbf{f}, \boldsymbol{w}),\ \boldsymbol{d_\rho}(\mathbf{f}, \boldsymbol{w}))$ are the outputs at the penultimate layer of the Bayesian U-Net, which stand for means and heteroscadetic (non-constant) standard deviations. And the last output layer is a distribution layer with the same parameters. Since $(\boldsymbol{d_\mu}(\mathbf{f}, \boldsymbol{w}),\ \boldsymbol{d_\rho}(\mathbf{f}, \boldsymbol{w}))$ are variables in themselves, in order to get the prediction one needs to sample over this output distribution. $N, \mathcal{F}$ are total number of training examples and dof per problem respectively. σ_p stands for the standard deviation of each the prior, which is kept constant in the training procedure. Optimized parameters, $\boldsymbol{\theta}^*_{\mathrm{VB}} = (\boldsymbol{\theta}^*, \boldsymbol{u}_{\boldsymbol{p}}^*)$, for the Variational bayes case are obtained by minimising the above loss function:

$$\boldsymbol{\theta}^*_{\mathrm{VB}} = \arg\min_{\boldsymbol{\theta}_{\mathrm{VB}}} \mathcal{L}_{\mathrm{VB}}(\mathcal{D}, \boldsymbol{\theta}_{\mathrm{VB}}) \tag{4.18}$$

Once the optimised parameters are computed, we replace the true posterior $P(\boldsymbol{w}|\mathcal{D})$ in Eq. (4.12) with the variational posterior $q(\boldsymbol{w}|\boldsymbol{\theta}^*)$ to get the predictive distribution:

$$P(\mathbf{u}|\mathbf{f},\mathcal{D}) = \int P(\mathbf{u}|\mathbf{f},\boldsymbol{w})P(\boldsymbol{w}|\mathcal{D})\mathrm{d}\boldsymbol{w} \approx \int P(\mathbf{u}|\mathbf{f},\boldsymbol{w})q(\boldsymbol{w}|\boldsymbol{\theta}^*)\mathrm{d}\boldsymbol{w} \tag{4.19}$$

The resultant predictive distribution can be approximated by Monte Carlo integration of Eq. (4.19) by sampling weights over optimised distributions, $\widetilde{\boldsymbol{w}_t} \sim q(\boldsymbol{w}|\boldsymbol{\theta}^*)$. At last for a given input force array, $\boldsymbol{f}$, probabilistic displacement prediction is obtained as an output. We represent this output distribution by the mean $\mathcal{U}_\mu(\mathbf{f},\boldsymbol{w})$ and the standard deviation $\mathcal{U}_\sigma(\mathbf{f},\boldsymbol{w})$ of the prediction, $P(\mathbf{u}|\mathbf{f},\mathcal{D})$. This is done by taking mean and standard deviation of T stochastic forwarded passes for the same input as follows:

$$\begin{aligned}
\mathcal{U}_\mu(\mathbf{f},\boldsymbol{w}) &\approx \frac{1}{T}\sum_{t=1}^{T} P(\mathbf{u}|\mathbf{f},\widetilde{\boldsymbol{w}_t}) \\
\mathcal{U}_\sigma^2(\mathbf{f},\boldsymbol{w}) &\approx \frac{1}{T}\sum_{t=1}^{T} P(\mathbf{u}|\mathbf{f},\widetilde{\boldsymbol{w}_t})^T P(\mathbf{u}|\mathbf{f},\widetilde{\boldsymbol{w}_t}) - \mathcal{U}_\mu(\mathbf{f},\boldsymbol{w})^T \mathcal{U}_\mu(\mathbf{f},\boldsymbol{w})
\end{aligned} \tag{4.20}$$

In case of MLE, we do not place distributions over parameters, and they are discrete like in the case of the deterministic network, as in Eq. (4.8). In the penultimate layer, we take ($\boldsymbol{d_\mu}(\mathbf{f},\boldsymbol{w})$, $\boldsymbol{d_\rho}(\mathbf{f},\boldsymbol{w})$) outputs standing for means and heteroscadetic standard deviations, which are then used to form the final Gaussian distribution output layer. Optimal parameters of the network are computed by minimising the following loss function

$$\mathcal{L}_{\mathrm{MLE}}(\mathcal{D},\boldsymbol{\theta}_{\mathrm{MLE}}) \approx -\sum_{k=1}^{N}\sum_{l=1}^{\mathcal{F}}\left[-\log\left(\sqrt{2\pi}\; d_\sigma^{(l)}(\mathbf{f}^{(k)},\boldsymbol{\theta})\right) - \frac{\left(\hat{u}_l^{(k)} - d_\mu^{(l)}(\mathbf{f}^{(k)},\boldsymbol{\theta}_{\mathrm{MLE}})\right)^2}{2\left(d_\sigma^{(l)}(\mathbf{f}^{(k)},\boldsymbol{\theta}_{\mathrm{MLE}})\right)^2}\right], \tag{4.21}$$

where

$$d_\sigma^{(l)}(\mathbf{f}^{(k)},\boldsymbol{\theta}_{\mathrm{MLE}}) = \log(1+\exp(d_\rho^{(l)}(\mathbf{f}^{(k)},\boldsymbol{\theta}_{\mathrm{MLE}})). \tag{4.22}$$

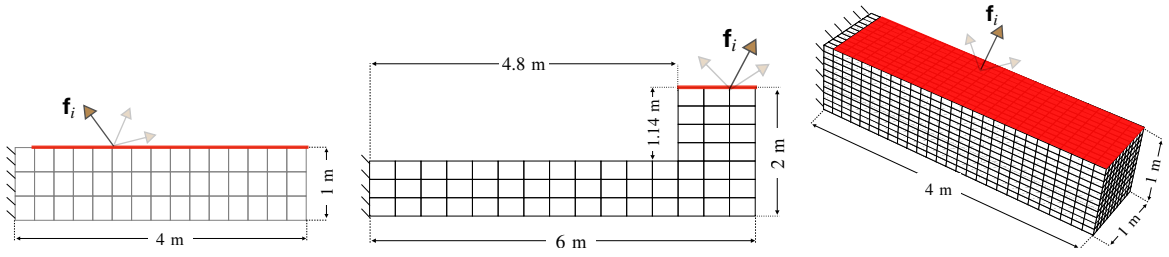
At last, optimal parameters of MLE U-Net models are computed by minimising the loss functions as

$$\boldsymbol{\theta}^*_{\mathrm{MLE}} = \arg\min_{\boldsymbol{\theta}_{\mathrm{MLE}}} \mathcal{L}_{\mathrm{MLE}}(\mathcal{D},\boldsymbol{\theta}_{\mathrm{MLE}}) \tag{4.23}$$

4.4 Results

4.4.1 The numerical experiment procedure

Generation of Training Data from Hyperelastic FEM Simulations



(a) (b) (c)

Fig. 4.4 Schematics of three benchmark examples (a) 2D beam, (b) 2D L-shape and (c) 3D beam. The parts of top surfaces marked in red color indicate the nodes at which random nodal forces are applied to generate training datasets.

Two 2D and one 3D benchmark problems are considered in this work, as schematically shown in Figure 4.4. The Neo-Hookean hyperelastic material model is used, with Young's modulus $E = 0.5$ kPa and the Poisson's ratio $\nu = 0.4$. We use the following version of Neo-Hookean strain energy potential

$$W(\boldsymbol{F}) = \frac{\mu}{2}(I_c - 3 - 2\ln J) + \frac{\lambda}{4}(J^2 - 1 - 2\ln J), \tag{4.24}$$

where the invariants J and I_c are given in terms of deformation gradient $\boldsymbol{F}$ as

$$J = \det(\boldsymbol{F}), \quad I_c = \mathrm{tr}(\boldsymbol{F}^T\boldsymbol{F}), \quad \text{where} \quad \boldsymbol{F} = \boldsymbol{I} + \nabla\boldsymbol{u}, \tag{4.25}$$

while μ and λ are Lame's parameters, which can be expressed in terms of the Young's modulus, E, and the Poisson's ratio, ν, as

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}, \quad \mu = \frac{E}{2(1+\nu)}. \tag{4.26}$$

As introduced in Section 5.2.6, for a given discretized problem, the training/testing dataset is constructed as follows. Within nodes occupying a prescribed region of the boundary (in red color in Figure 4.4), a particular family of external force distribution is considered. Each loading case consists of a single excited node, while for the remaining nodes the external forces are $\mathbf{0}$. For a given training/testing example, a single node is chosen for which the external force vector is generated randomly, component-wise, from a uniform distribution within a given range of

magnitude. The example is then solved with FEM, and the entire vector $\mathbf{f}$ of prescribed nodal external forces (including unloaded nodes) and the vector $\mathbf{u}$ of computed nodal displacements are saved. The procedure is repeated for all $N+M$ examples, which creates the training/testing dataset $D = \{(\mathbf{f}_{(1)}, \mathbf{u}_{(1)}), ..., (\mathbf{f}_{(N+M)}, \mathbf{u}_{(N+M)})\}$.

The finite element simulations have been performed with the AceGen/AceFem framework [Korelc, 2002] (standard library displacement-based Neo-Hoohean finite elements are used). The non-linear FE problems are solved with the Newton-Raphson method, and an adaptive load-stepping scheme is used to avoid convergence issues for large load cases. A single quad/hexahedral FE mesh per problem is only considered.

Remark: For 2D/3D beam examples the structured FE mesh is used, which is compatible with the U-Net architecture introduced in Section 4.2.2. In the L-shaped example, the FE mesh is not structured, which makes it impossible to directly transform it to a compatible node numbering, with a possible consequence of accuracy drop, as explained in Section 4.4.2. To correct this, a special zero-padding operation is applied to each $\mathbf{f}_{(i)}$ and $\mathbf{u}_{(i)}$ before using the dataset for training/testing, see Figure 4.5. Note here that unstructured meshes can be handled in several other ways. One way would be to embed a structured grid on the unstructured mesh and map the unstructured nodal values to the structured nodes. Another promising approach would be to use recently developed graph networks [Hanocka et al., 2019][Pfaff et al., 2021]. These more sophisticated approaches are, however, out of the scope of the present work.

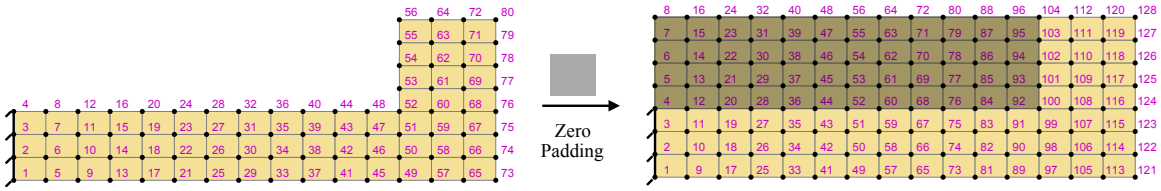


Fig. 4.5 Extra nodes with zero force/displacement are added to convert the data to a structured format. Node numbers are mapped accordingly to follow the assumed order of U-Net architecture.

The datasets are randomly split into training sets, N (95%), and testing sets, M (5%). The characteristics of FE meshes and datasets for all three problems are provided in Table 4.1.

Implementation and training of U-Nets

For the 2D cases, we use 3 level U-Net architectures as in Figure 4.2, at each level, we apply two convolutional operators with 3×3 filters with c=128 channels in the first level. For Bayesian U-Nets, we replace half of the layers with probabilistic layers (one layer out of two at each level is replaced with a probabilistic layer). For the 3D case, we use a 4 level U-Net architecture,

Problem	N.of FEM DOFs ($\mathcal{F}$)	Force component range [N]	dataset size N+M
2D beam	128	-2.5 to 2.5	$5700 + 300$
2D beam$^{\#}$	128	-1 to 1	$3800 + 200$
2D L-shape	160 (256*)	-1 to 1	$3800 + 200$
3D beam	12096	-2 to 2	$33688 + 1782$

Table 4.1 FE datasets. The number of DOFs with the asterisk refers to the zero-padded 2D L-Shaped mesh. 2D beam$^{\#}$ is an additional dataset used for analysing probabilistic U-Nets in Section 4.4.3

we apply two convolutional operators with $3 \times 3 \times 3$ filters with c=128 channels in the first level. Additionally, for both cases, we use batch-normalization on each layer. This technique standardizes the inputs to a layer for each mini-batch [Ioffe and Szegedy, 2015]. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

Training: Network is trained by minimising loss function for the given training dataset, minimisation is performed using Adam optimizer, a well-known adaptive stochastic gradient-descent algorithm. We set the learning rate to 1×10^{-4} and set other optimizer parameters as per recommendations [Kingma and Ba, 2017]. For the Monte Carlo sampling of loss function ($\mathcal{L}_{\text{VB}}$) in Eq. (4.16), we use *Flipout* estimator [Wen et al., 2018] with its recommended parameter values. Trainings of deterministic and probabilistic versions of U-Net are carried out using Keras [Chollet et al., 2015] and Tensorflow-probability [Dillon et al., 2017] libraries respectively. All the implementations are done on Tesla V100-SXM2 GPU, on HPC facilities of the University of Luxembourg [Varrette et al., 2014] using a batch size of 4 and 600/75 epochs for 2D/3D cases. All the experiments in this work are performed using a single-precision arithmetic ('float32'), which is the usual default choice for all the deep learning libraries. The use of double-precision increased the memory requirements and the training time, without any improvement in the accuracy, and hence is unnecessary. (For the 2D beam example, double precision implementation took 4 times the training time that of the single-precision implementation.)

Since the prediction of Bayesian U-Net is a distribution, we take 300 stochastic forward passes for the same input to get the mean and uncertainty of the prediction (T=300 in Eq.(7.13)).

Validation Metrics for the testing phase

For the test set $\{(\mathbf{f}_1, \mathbf{u}_1), ..., (\mathbf{f}_M, \mathbf{u}_M)\}$, we use the following mean absolute error norm as the validation metric:

$$e_m = \frac{1}{\mathcal{F}} \sum_{i=1}^{\mathcal{F}} |\mathcal{U}(\mathbf{f}_m)^i - \mathbf{u}_m^i|. \tag{4.27}$$

$\mathcal{F}$ is the number of dofs of the mesh. For m^{th} test example, $\mathcal{U}(\mathbf{f}_m)$ is the deterministic network prediction and $\mathbf{u}_m$ is the finite element solution. To have a single validation metric over the entire test set, we compute the average mean norm $\bar{e}$ and the corrected sample standard deviation $\sigma(e)$ as follows:

$$\bar{e} = \frac{1}{M} \sum_{m=1}^{M} e_m, \qquad \sigma(e) = \sqrt{\frac{1}{M-1} \sum_{m=1}^{M} (e_m - \bar{e})^2}. \tag{4.28}$$

(Note: It is the standard deviation of averaged errors across the test set, not the standard deviation of all errors.)

In the case of Bayesian U-Nets, the output of the network is a probability distribution, for that reason, we sample over the output distribution by taking multiple forward passes as described in Eq. (7.13). Mean over these samples, $\mathcal{U}_\mu(\boldsymbol{f}_m)$, is taken as the mean prediction of the Bayesian U-Net, while the standard deviation of these samples, $\mathcal{U}_\sigma(\boldsymbol{f}_m)$, gives us the confidence intervals of predictions. Now the error norm for m^{th} test example is given as

$$e(\mathcal{U}_\mu(\mathbf{f}_m), \mathbf{u}_m) = \frac{1}{\mathcal{F}} \sum_{i=1}^{\mathcal{F}} |\mathcal{U}_\mu(\mathbf{f}_m)^i - \mathbf{u}_m^i|. \tag{4.29}$$

Again the average error norm and the corrected sample standard deviation for all test examples is computed as

$$\bar{e} = \frac{1}{M} \sum_{i=1}^{M} e(\mathcal{U}_\mu(\mathbf{f}_m), \mathbf{u_m}), \qquad \sigma(e) = \sqrt{\frac{1}{M-1} \sum_{m=1}^{M} (e(\mathcal{U}_\mu(\mathbf{f}_m), \mathbf{u}_m) - \bar{e})^2}. \tag{4.30}$$

4.4.2 Deterministic U-Nets

Advantages of the U-Net convolutional architecture

U-Nets vs. fully-connected NNs

U-Nets leverage the fact that nearby nodes of the FEM mesh show strong local correlation, and provide computationally efficient topology that is able to capture non-linearities. However,

if we had to use a fully connected neural network to capture these non-linearities, the number of latent parameters of this network would be significantly larger as compared to that of the U-Net.

To show this effect, we consider the simplest fully connected network, with only input and output layers, without no hidden layers nor activation functions, as a surrogate model for the 3D beam example (as in Figure 4.4c). This example has 12096 dof, so the dimension of the input and output layer is 12096 each. Because of the absence of hidden layer/activation functions, this network is only able to capture a linear response of the force-displacement relationship. In order to have the best-linearised approximation, we initialise trainable parameters of the fully connected network with the inverse of the FEM stiffness matrix. Table 4.2 shows that the fully connected network (which is an inaccurate assumption) has 1.5 more parameters than the deterministic U-Net, while the accuracy is greatly reduced. In order to do a better (non-linear) approximation, one would need to use a multi-layer fully connected network, which would require even more parameters, and hence the training time would be significantly higher. Hence, the choice of U-Nets makes complete sense, in particular for complex non-linear problems.

NN type	N. of trainable parameters	$\bar{e}$ [m]	$\sigma(e)$ [m]
Deterministic U-Net	94.1 E+6	0.6 E-3	0.3 E-3
Fully-connected	146.3 E+6	7.0 E-3	8.0 E-3

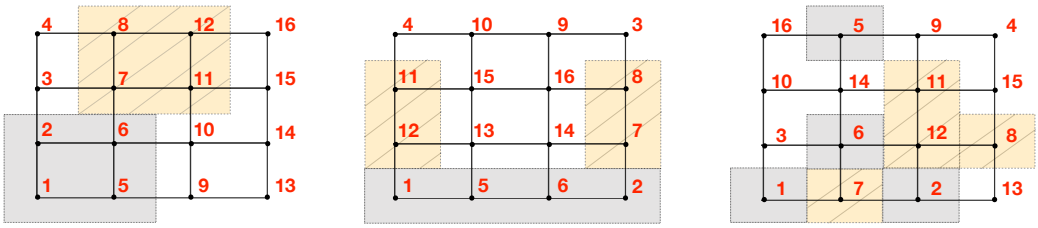
Table 4.2 Comparison of U-Net vs feed forward network for 3D Beam example

Effect of DOF ordering

The topology of input FEM mesh plays a crucial role in the training of U-Nets, and it must be compatible with that of the U-Net architecture topology. However, different FEM pre-processors have different ways of numbering nodes. For instance, Gmsh [Geuzaine and Remacle, 2009], a popular FE mesh generator, first numbers corner nodes, then edge nodes followed by internal nodes, see Figure 4.6b. This is not compatible with the expected U-Net input, which effects deteriorating the predictive capabilities of the U-Net. A completely random ordering, see Figure 4.6c, performs even worse, see Table 4.3. To fully leverage the advantages of U-Nets, care has to be taken to order nodes properly. This is the reason why the zero-padding has been done to the L-shaped case, see the remark in Section 4.4.1, and Figure 4.5.

Ordering strategy	$\bar{e}$ [mm]	$\sigma(e)$ [mm]
Preferred ordering (as in Figure 4.6a)	0.6	0.3
Random ordering (as in Figure 4.6c)	5.4	2.8

Table 4.3 Error metrics for the preferred and randomly ordered case for the 3D beam problem.



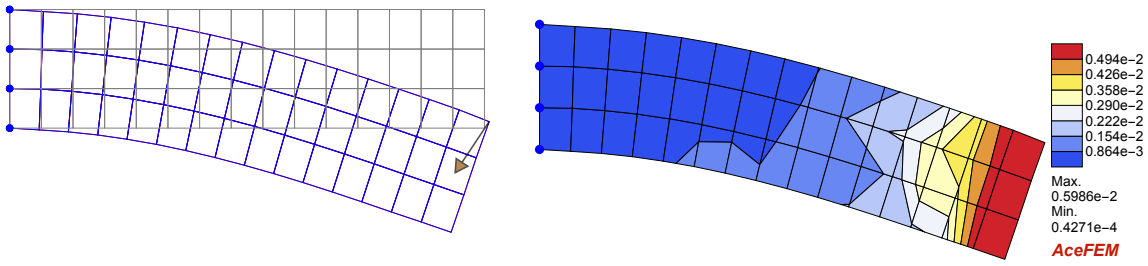
(a) (b) (c)

Fig. 4.6 Different node numbering strategies. (a) Numbering assumed by the TensorFlow (the preferred one; used in this work), (b) Gmsh preprocessor numbering, and (c) random numbering.

Prediction accuracy

Deterministic U-Nets are trained on FEM datasets generated as described in Section 4.4.1. Below, we analyse in a more detail some selected test examples for each case, and compare their FEM and U-Net solutions. For all the examples, we show the overlap of deformed meshes obtained using FEM and U-Net models. In Figures 4.7-4.11 and Figure 4.13, gray, blue and red meshes represent undeformed configuration, U-Net solution and FEM solution, respectively. In addition to that, we also present the interpolated node-wise L_2 norm of the prediction error (the error of the nodal displacement between FEM and U-Net.

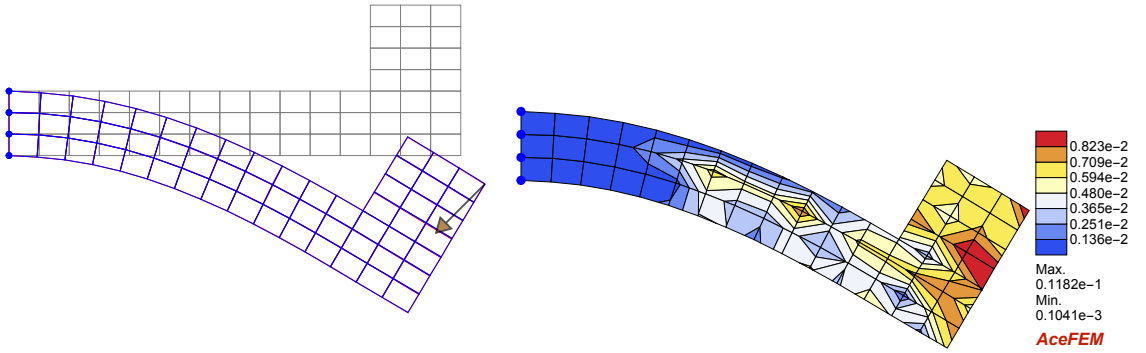
In Figure 4.7, we show a test example of the 2D-beam case. A point force is applied at the corner node of the beam and the deformation of mesh is predicted using the deterministic U-Net. As we can see, the deformed mesh predicted with U-Net is overlapping with the reference FEM solution. As explained above, we also plot the nodal error field on the deformed mesh, one can observe that the error is relatively higher in the high displacement region, i.e, near the free end. The relative error for the tip with respect to its displacement magnitude is only 0.6%.



(a) (b)

Fig. 4.7 Deformation of 2D beam computed using the deterministic U-Net, for the point force at the free end. (a) Comparison of deformed meshes, both blue mesh (U-Net solution) and red mesh (reference FEM solution) are overlapping. The magnitude of the tip displacement is 0.95 m. (b) L_2 error of nodal displacements between deterministic U-Net and FEM solutions.

Figure 4.8 shows an example of the 2D L-shape case. Again the deterministic U-Net solution is overlapping with the reference FEM solution. L_2 error contour shows that a high error trend is observed near the free end again, the relative error at the top corner node with respect to displacement magnitude is 0.4% only.



(a) (b)

Fig. 4.8 Deformation of 2D-L shape computed using deterministic U-Net. (a) Initial and deformed meshes predicted using deterministic U-Net(blue) and FEM(red), the magnitude of tip displacement is 2.39 m, and (b) L_2 error of nodal displacements between deterministic U-Net and FEM solutions.

We further take a look at two 3D-beam test examples, one with the force applied near the free end and another with the force applied in the middle region of the 3D beam. For both cases, the deterministic U-Net solutions are overlapping with the FEM solutions. Insets in Figure 4.9 show that the U-Net is capable of predicting high local non-linear deformations. For the first example in Figure 4.9a-4.9b, the error field shows high error region near the point of application of the force. The relative error for the tip for this case is only 0.6%. Whereas, Figure 4.9c-4.9d shows an example with the force applied relatively near to the fixed end. In this case, a high error field is observed at the point of application of force as well as near the free end. The relative L_2 error of the tip for this example is 1.6%. From this, we can say that errors are usually higher near the nodes with higher displacement magnitudes.

Till now we looked at the prediction accuracy for individual examples, now we would like to judge the performance of deterministic U-Net over the entire test sets (5% of the generated data is designated for testing purposes). Table 4.4 provides such comparison in a form of averaged error over entire test sets. We can see that, on average, the error is at a reasonably low level. To extend this analysis, in Figure 4.10 we plot the mean error (e) of each test example of the three benchmark problems. We sort these errors as per the increasing displacement magnitude at the point of application of force. To get a relation between displacement and mean error (e),

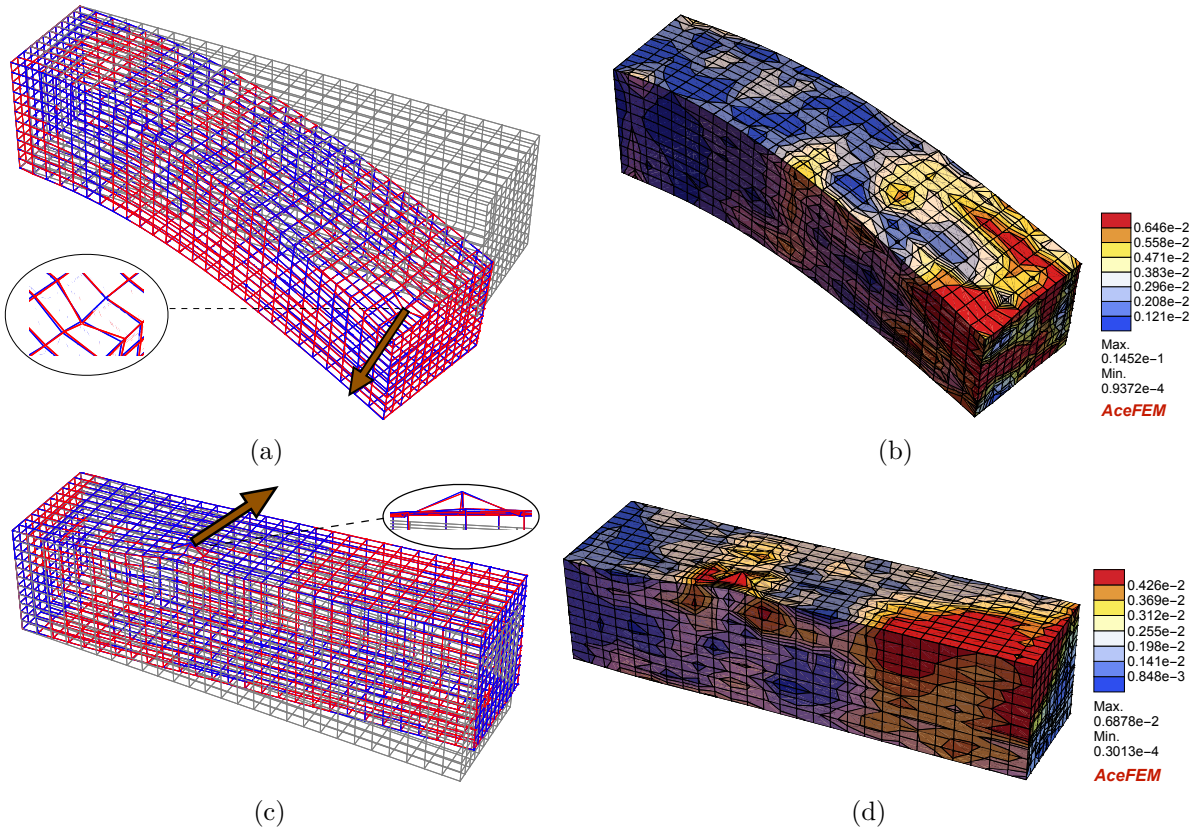


Fig. 4.9 Deformation of 3D beam computed using deterministic U-Net (blue) for two force cases, for comparison FEM solution (red) is presented. (a)&(c) deformed meshes for both examples. The magnitude of tip displacement for the first case (force near the free end) is 1.1 m and for the second case (force in the middle region) is 0.26 m. High localized deformations are shown in the insets. (b)&(d) L_2 error of nodal displacements between deterministic U-Net and FEM solutions.

we do a least square linear fit for all three cases. From Figure 4.10, all the three examples show generally low sensitivity to the increase of displacement magnitude.

Effect of changing the distribution of applied forces

Deterministic U-Net has been trained by using single point load examples only, but we would like to check how it performs when multiple point load input is given for the prediction. Figure 4.11 shows one such example where random multiple forces are applied on the top edge, U-Net is able to closely follow the reference FEM solution. Figure 5.16b shows the L_2 norm of the error across the beam, it shows a different trend for this example. Though the deformation is higher in the free end region and at the point of application of forces, a higher error is observed at a different location also. Solution accuracy of multiple point load cases can be improved by

Example	M	$\bar{e}$ [m]	$\sigma(e)$ [m]
2D Beam	300	0.3 E-3	0.2 E-3
2D L-Shaped	200	0.8 E-3	0.4 E-3
3D Beam	1782	0.6 E-3	0.3 E-3

Table 4.4 Error metrics for 2D and 3D test sets for predictions using deterministic U-Net. M stands for the number of test examples, and $\bar{e}$ and $\sigma(e)$ are error metrics defined in Section 4.4.1.

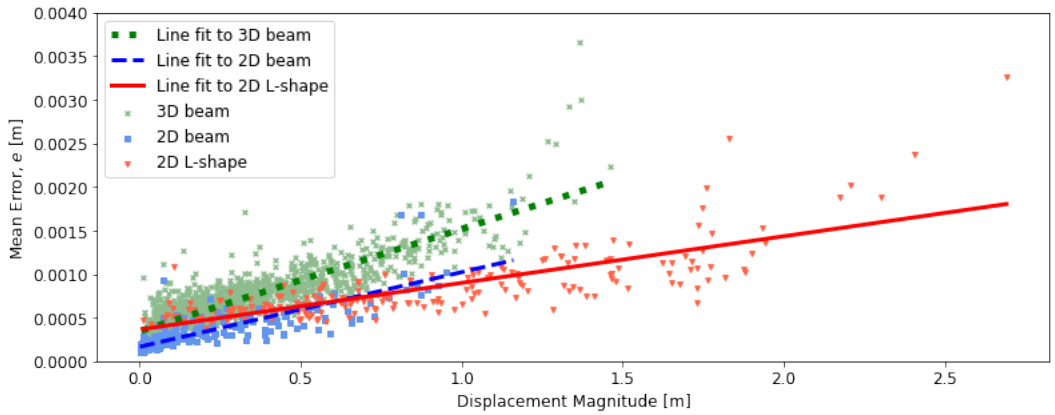
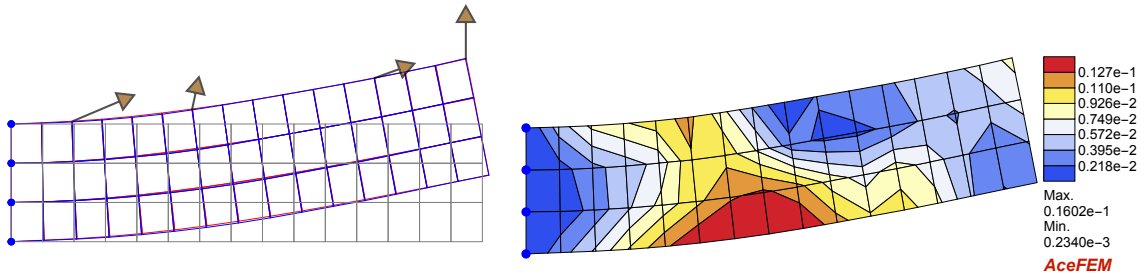


Fig. 4.10 Mean errors (e) for all test examples of three benchmark cases. The regression lines $y \propto 0.0008 \times x$ (2D-beam), $y \propto 0.0005 \times x$ (2D-L shape), $y \propto 0.001 \times x$ (3D beam) show low sensitivity of the deterministic U-Nets to displacement magnitudes.

incorporating multiple point loads in the training phase. Also, the relative error for the tip with respect to its displacement magnitude for this example is 0.6%.

In most engineering applications, we are interested in the cases where force is applied in a given prescribed region of interest (e.g., the Neumann boundary). Here, we would like to check how U-Net performs when this assumption is violated, i.e, we apply forces on the nodes which were not involved in the training procedure. To do so, we use the same 2D beam case with the training set generated by applying point forces on the top edge (indicated by the red line in Figure 4.4 in Section 4.4.1). What we change is the prediction phase, during which we apply forces on nodes located on the vertical free edge of the beam (see schematics in Figure 4.13). In the example, we apply a vertical force of 1.5 N on each of the 4 nodes of the free edge of the beam. Figure 4.13a shows that mesh (blue) predicted with U-Net deviates more and more from the true FEM solution, as we move away from the training line. The U-Net solution is much worse when the force is applied on the 4^{th} node as compared to the 2^{nd}, i.e. when the point of force application is farthest from the training line. In Figure 4.13b, we plot the mean and maximum errors of all 4 examples, and we can observe a significant accuracy drop reaching two orders of magnitude when predicting outside the training region. Also, we can see that the



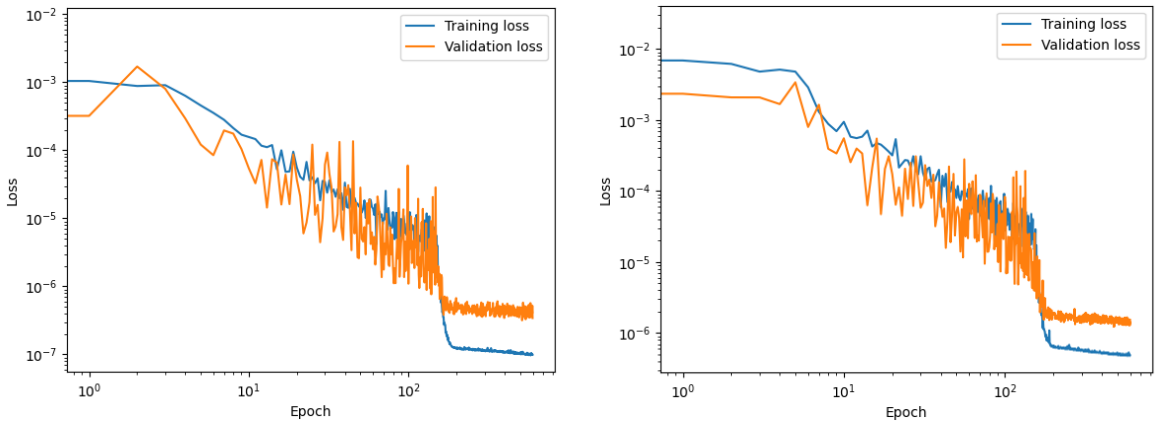
(a) (b)

Fig. 4.11 Deformation of the 2D beam subjected to multiple point loads. (a) Comparison of deformed meshes predicted with deterministic U-Net and FEM, the magnitude of tip displacement is 0.55 m. (b) L_2 error of nodal displacements between deterministic U-Net and FEM solution.

errors are increasing when moving away from the training dataset. This proves that the U-Nets extrapolate predictions poorly when moving away from the training range in spatial directions.

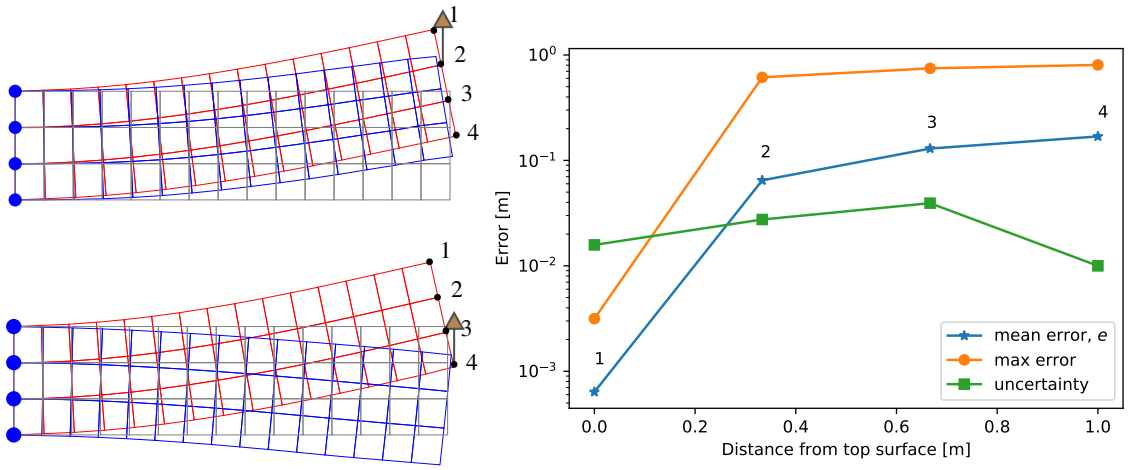
Training convergence

The choice of the amount of training data and the appropriate neural network architecture are two important criteria in the case of neural network surrogate modeling. This is crucial to ensure that neither underfitting nor overfitting is observed. For all the cases in this work, training convergence is ensured by observing loss plots of training and validation errors, i.e., the training error doesn't decrease, and validation error doesn't go higher with the number of epochs. For the reference, the loss plots for 2D cases are shown in Figure 4.12.



(a) (b)

Fig. 4.12 Mean squared error log-loss plots for (a) 2D beam and (b) 2D L-shape case.



(a) (b)

Fig. 4.13 Application of point loads on the nodes away from the training region. (a) Deformed meshes obtained by FEM (red) and with U-Net (blue) when point load is applied on 2^{nd} or 4^{th} node. (b) Mean error (e) and maximum error for each of the 4 point loads cases. Green line shows the uncertainty prediction of Bayesian U-Net, for the node on which the force is applied.

4.4.3 Probabilistic U-Nets

The goal of our probabilistic U-Net framework is to get reliable predictions and uncertainty associated with those predictions. Further in this section, we will check this for the case of data and model uncertainties for selected examples analogous to the deterministic case.

Prediction accuracy

We train the probabilistic U-Net framework on the same datasets as used in deterministic cases. Because the output of the network is a distribution, we make 300 stochastic forward passes to get the mean and uncertainty predictions for a given input. Mean prediction of Bayesian U-Net is treated as the solution of the network, whereas uncertainty predictions give information of credible intervals of predictions. Table 4.5 gives the error metrics for the Bayesian U-Net predictions over the entire test sets, for comparison we have shown the errors of deterministic counterparts as well.

Similar to the deterministic case, we do the analysis of the error metric (e) for all the test examples predicted using Bayesian U-Net this time. Figure 4.14 shows errors sorted as per the increasing displacement magnitudes of the point of application of forces. We perform a least-squares line fit to the error data. Slopes for 2D-beam and 2D L-shape cases are small, proving a little sensitivity of errors to the displacement magnitudes.

Example	M	$\bar{e}$ [m]	$\sigma(e)$ [m]
2D Beam (VB)	300	1.3 E-3	1.3 E-3
2D Beam (D)	300	0.3 E-3	0.2 E-3
2D L-Shaped (VB)	200	5.3 E-3	3.7 E-3
2D L-Shaped (D)	200	0.8 E-3	0.4 E-3

Table 4.5 Error metrics for 2D test sets using Bayesian U-Nets. D = Deterministic, VB = Variational Bayes.

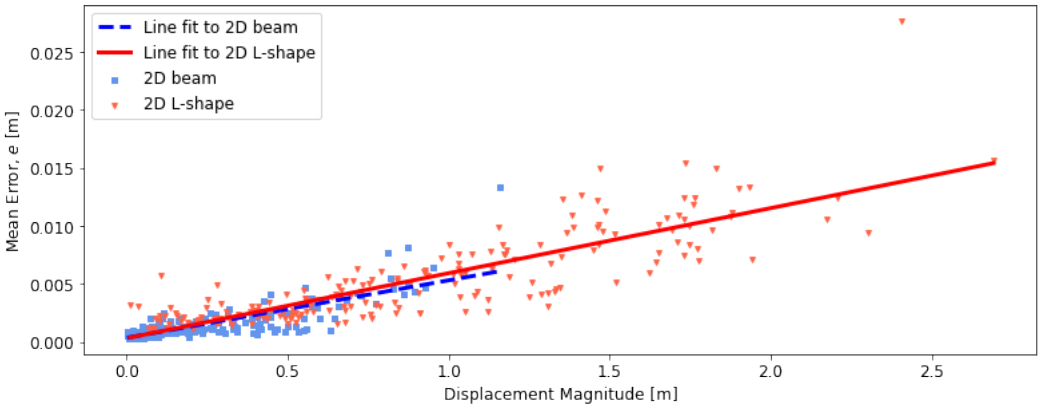
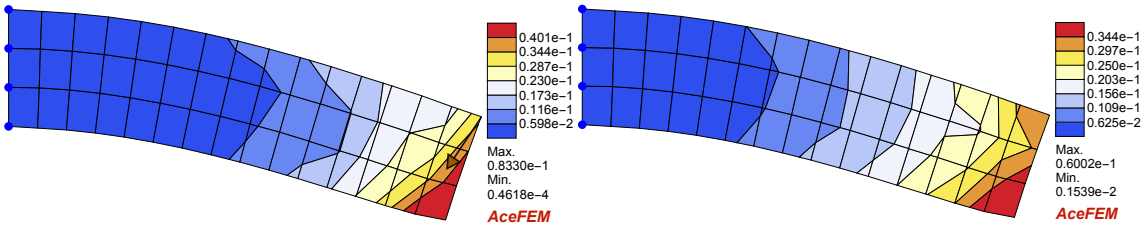


Fig. 4.14 Mean errors (e) for all test examples of 2D cases, for predictions using Bayesian U-Net. The regression lines $y \propto 0.005 \times x$ (2D beam), $y \propto 0.006 \times x$ (2D-L shape) show low sensitivity of Bayesian U-Net errors to displacement magnitudes.

Hereafter we focus on particular examples to get more insights on Bayesian U-Net predictions. Similar to the deterministic cases, we take node-wise L_2 norm of the error (Error of FEM and mean prediction of Bayesian U-Net) and also that of the uncertainty prediction from Bayesian U-Net. Both error and uncertainty values are interpolated within the element to get respective fields, which are plotted on the deformed mesh obtained using Bayesian U-Net.

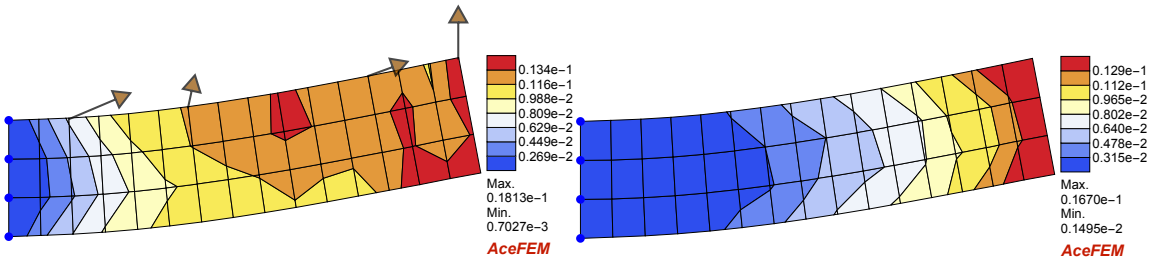
We consider the same 2D-beam test case as in Figure 4.7, (as in deterministic case). This time we make the prediction using Bayesian U-Net. Figure 4.15 shows the comparison of error and uncertainty associated with the prediction (we plot single standard deviation values associated with the prediction of respective dof). One can see that both are strongly co-related spatially.

A similar kind of analysis is done for the multiple point load case Figure, see 4.11, in the deterministic section. Figure 4.16 compares the error and uncertainty fields obtained using the Bayesian U-Net, we can see that they are correlated and closely follow each other as well.



(a) (b)

Fig. 4.15 Deformation of 2D Beam predicted by the Bayesian U-Net, for the same example as in Figure 4.7. (a) The error between Bayesian U-Net and FEM solution plotted on the deformed mesh. (b) Uncertainty of prediction obtained using Bayesian U-Net plotted on the deformed mesh.

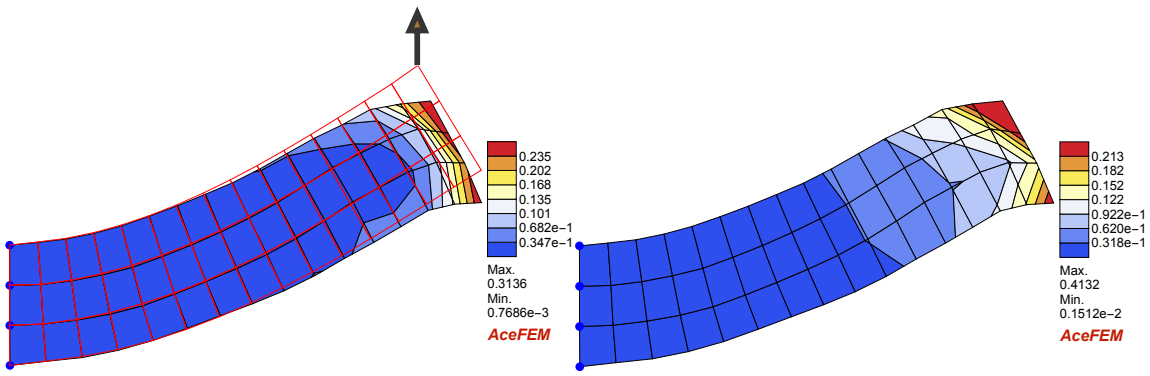


(a) (b)

Fig. 4.16 Deformation of 2D Beam for multiple point forces using Bayesian U-Net, for the same example as in Figure 4.11. (a) The error between Bayesian U-Net and FEM solution plotted on the deformed mesh. (b) Uncertainty of prediction obtained using Bayesian U-Net plotted on the deformed mesh.

Force outside training range: Let us consider a test example in which a force of 5 N is applied on the corner node, which is far away from the training range (which is -2.5 to 2.5 N). Again we compare error and uncertainty associated with the Bayesian U-Net prediction. For reference, the FEM solution is presented (red mesh) with the error contour plot. Both error and uncertainty are plotted on the deformed mesh predicted with the Bayesian U-Net. In Figure 4.17, one can see that both are strongly correlated, rather both values are close to each other across the spatial dimensions of the beam. Thus, the uncertainty predictions can give us an idea about the error of U-Net predictions, irrespective of whether an input is within or outside the training region.

For each of the above examples shown in Figure 14-16, we can see that the U-Net solution is deviating from the true FEM solution, which is given by the error contour, i.e., the U-Net model is not able to fit the data exactly. And uncertainty prediction obtained using the Bayesian U-Net is able to capture this fitting error.



(a) (b)

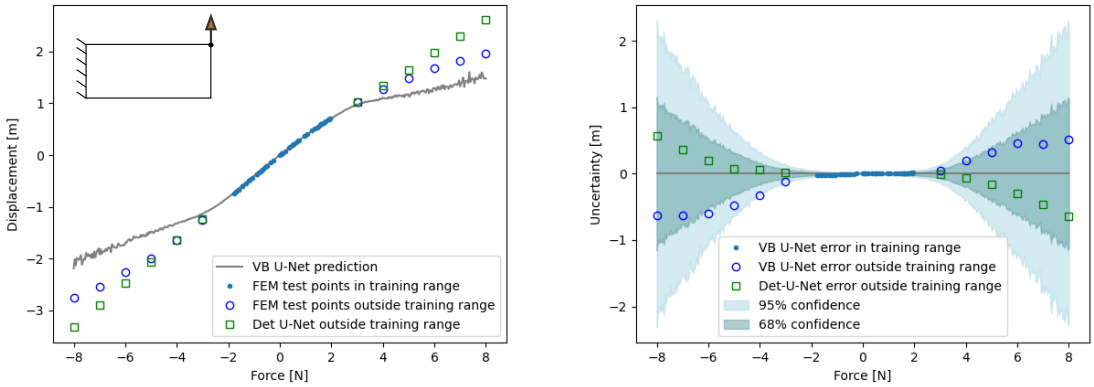
Fig. 4.17 Deformation of 2D Beam using Bayesian U-Net for an input force outside the training range. (a) Error between Bayesian U-Net and FEM solution plotted on the deformed mesh. (b) Uncertainty of prediction obtained using Bayesian U-Net plotted on the deformed mesh.

The example in Figure 4.17 can be considered as a case of extrapolation in the sense of the magnitude of force being outside the training range. One can also think of extrapolation in the sense of applying force on the nodes which were not included in the training, i.e., extrapolation in the spatial dimensions of the geometry. To analyse such cases we consider the same example as shown in Figure 4.13 in the deterministic Section 4.4.2. In Figure 4.13b we have shown the uncertainty of Bayesian U-Net prediction (one standard deviation), for the node on which point load is applied. As one can infer, Bayesian U-Net is not giving reliable uncertainty estimates when we move away from the training region in spatial directions. Intuitively the predicted uncertainty should be more for the case when force is applied on the farthest node from the training line, but on contrary, we observed a low prediction uncertainty for this point. One possible explanation of this limitation is, gradients w.r.t the spatial dimensions are not available neither in the data nor in the U-Net models. Hence there is no natural way of extrapolating information of solutions or uncertainties.

Hereafter we focus on displacement prediction of a single dof with Bayesian U-Net. We do this to see how the associated uncertainty varies with the value of input force, depending on whether the input is within or outside the training range. In Figure 4.18-4.21, we keep a constant direction of the input force, but gradually increase the magnitude and study the displacement prediction using Bayesian U-Net. The output of the Bayesian U-net is the displacement solution and the uncertainty associated with it, in Figure 4.18-4.20 we provide separate plots for both of these outputs. Whereas in Figure 4.21 we only study the prediction uncertainties for noisy data cases.

2D Beam: We apply multiple vertical forces varying from -8 N to 8 N on the corner node of the beam and predict its displacements using the Bayesian U-Net. Figure 4.18a gives the prediction

of displacement magnitude, as one can see prediction matches with test FEM solution within the training region. Outside the training region, Bayesian U-Net prediction deviates from the FEM solution. For reference, we provide deterministic U-Net solutions as well, even they deviate from FEM solutions outside the training range. Whereas Figure 4.18b gives confidence intervals associated with these predictions. One can see that network has very little uncertainty i.e. it is confident in the region of training data (-2.5 to 2.5 N) but as one moves away, the uncertainty of the prediction increases. We can also see that 95% confidence is able to capture the error of Bayesian U-Net predictions outside the training region, for reference, errors of deterministic U-Nets are presented as well.



(a) (b)

Fig. 4.18 Outputs of Bayesian U-Net when a range of forces is applied on the corner node of the 2D Beam (see inset). (a) The magnitude of Y-displacement of the corner node predicted with Variational Bayesian U-Net. (b) The uncertainty associated with the predictions of displacement solutions.

In this paragraph, we compare uncertainty intervals for Bayesian U-Nets trained on two different datasets. In addition to the existing 2D beam dataset (force range: -2.5 to 2.5 N), we consider another training dataset with a lower force range this time (force range: 1 to 1 N). Figure 4.19 shows the comparison of uncertainty intervals for these two cases, as the range of input force in the training set is decreasing, Bayesian U-Net tends to get more uncertain about its predictions in higher force ranges, which follows the common intuition.

2D L-shape: This training dataset was created by applying point forces in the range of -1 N to 1 N as shown in Table 4.1. In order to see how prediction uncertainty varies with the input forces, we apply multiple forces in a horizontal direction varying from -6 N to 6 N on the inner corner of the L-shape and predict its displacements using Bayesian U-Net. Figure **??** shows how displacement magnitude changes with applied force values. As we start to move away from the training region, the Bayesian U-Net solution deviates from the FEM solution. For reference, we have plotted the deterministic U-Net solutions as well. Figure 4.20b gives the uncertainty

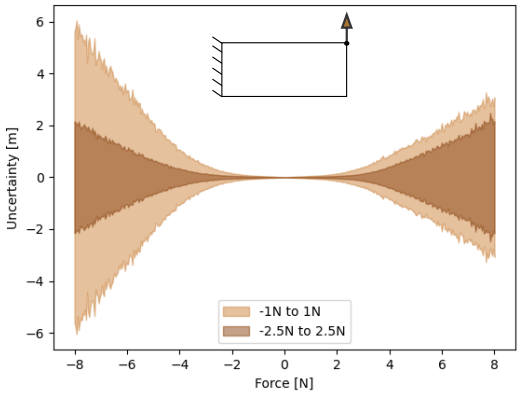
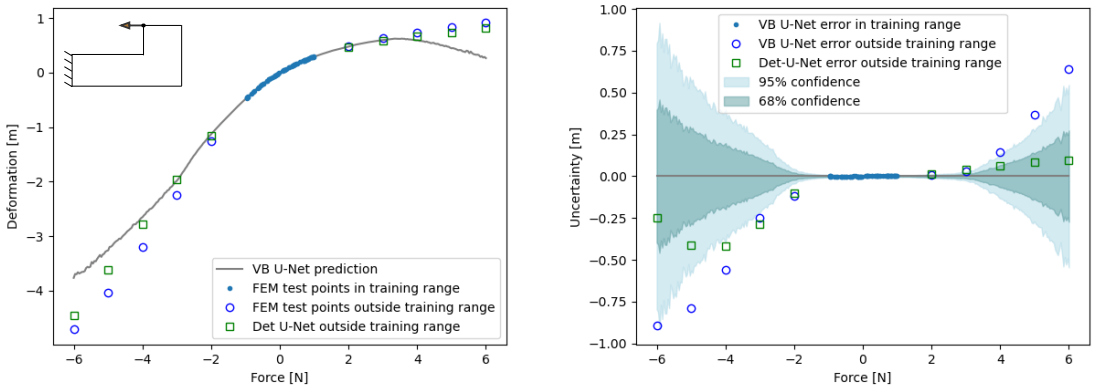


Fig. 4.19 Uncertainty intervals for the Y-displacement of the corner node of the 2D beam (see inset), predicted using the Bayesian U-Nets trained on different training sets. Uncertainty reduces with the increase of force range in training data.

associated with the prediction. Again the network is very confident in the training data region. But as the force value goes outside the training range, uncertainty tends to increase, for the reference, errors of deterministic U-Nets are presented as well.



(a) (b)

Fig. 4.20 Outputs of Bayesian U-Net when a range of forces is applied on the inner corner node of the 2D L-shape (see inset). (a) The magnitude of X-displacement of the inner corner node predicted with Variational Bayesian U-Net. (b) The uncertainty associated with the predictions of displacement solutions.

Noisy Data Case

In all the cases above, the U-Net models have been trained with numerical FEM datasets which can be regarded as noiseless. However, in many practical applications, especially when working

with experimental data, the data noises exist and can originate from various sources, such as measurement errors, errors associated with tools, human errors, etc. In this section, we would like to demonstrate that our framework is capable of capturing these data noises. To show that, we add random noises to our existing FEM datasets, and check how MLE- and Variational Bayes U-Nets perform in capturing these noises in terms of the predicted uncertainties.

For both 2D beam and 2D L-shape cases, we modify the existing datasets (of the input force range -1 N to 1 N as shown Table 4.1) by incorporating random noises to displacement values. When the magnitude of applied force is less than 0.7 N, we add a random noise (from a continuous uniform distribution) within 20% of the real displacement solutions, i.e., when $\|\boldsymbol{f}\|_2 \leqslant 0.7$ we set $\boldsymbol{u} \rightarrow (1+\gamma)\boldsymbol{u}$, where $\gamma \sim \mathrm{U}(-0.2, 0.2)$. Now, the probabilistic networks (MLE and Variational Bayes) are trained using these noisy datasets. In the prediction phase, we apply forces to a single chosen corner node in a single direction, with magnitudes ranging from -4 N to 4 N (see insets in Fig. 4.21). Then we analyse the predicted uncertainties associated with displacements of respective nodes to which the force has been applied, and how they relate to the level of input force noises.

Figures 4.21a and 4.21c show that the MLE approach is able to capture the noises in the training data region, although it fails to produce reliable uncertainty estimates outside that region (extrapolated region). The network is very confident in predictions even though we move away from the training region, and the prediction errors there are clearly visible. Whereas from Figure 4.21b-4.21d, we can see that the Variational Bayes approach is able to capture both effects: the effect of noises in the data, as well as the desired effect of gradually increasing uncertainty as we move away from the training region.

4.4.4 Prediction and training times

Prediction Times

Although networks are trained on Graphical Processing Units (GPU), predictions are computationally inexpensive on user end Central Processing Units (CPU) as well. Also, since recent years, GPU cloud computing is easily accessible, one can leverage GPU support over the internet. All these factors make our framework easily deployable to the user end. Table 4.6 gives the comparison of prediction times for different examples on GPU as well on CPU.

For some of the force values in the testing set, the FEM solution took more than 3s. Hence under identical computational resources, deterministic U-Net gave 31 times speedup. Another important point to mention here is, both deterministic and Bayesian U-Net, individually take the same time for prediction irrespective of the value of the input (applied force). In the case

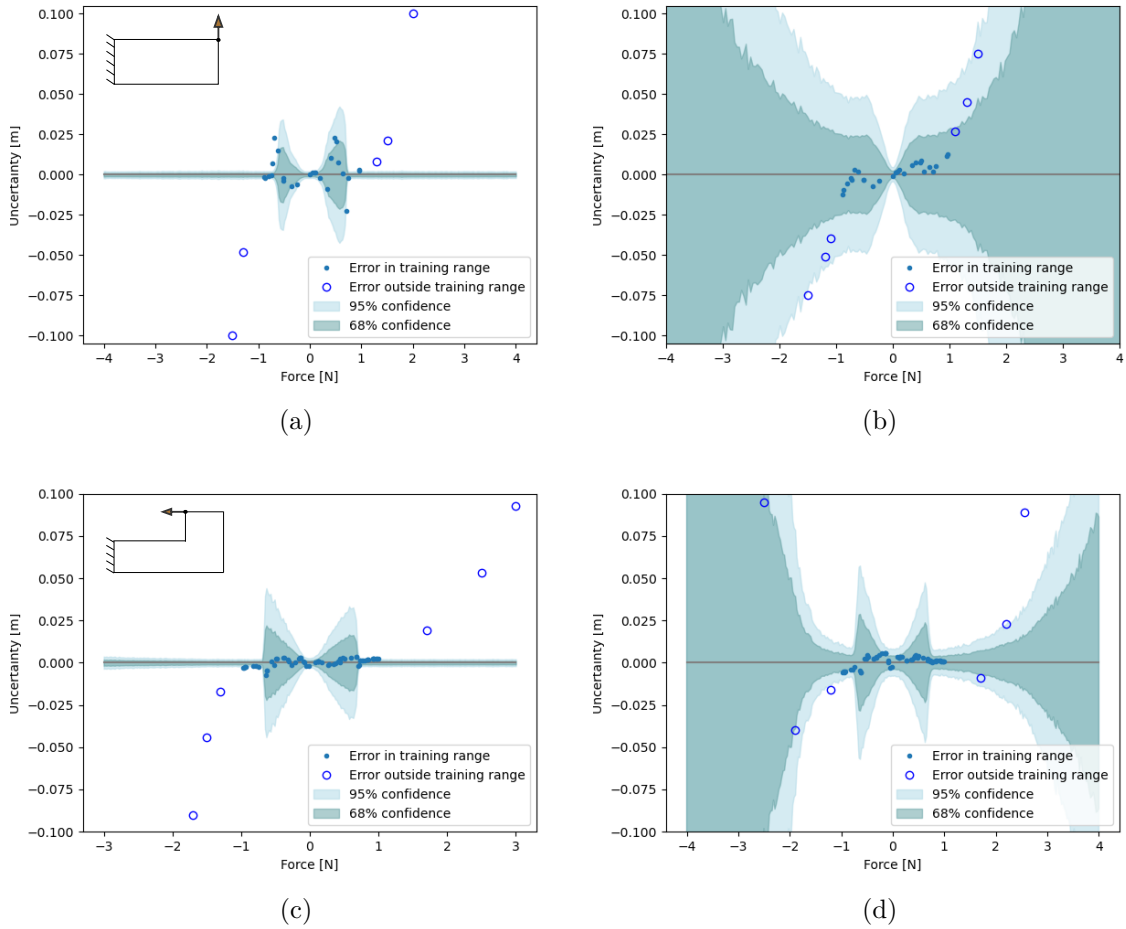


Fig. 4.21 Uncertainty predictions for noisy data cases using probabilistic U-Nets. For the 2D beam case (upper row), prediction is done for the Y-displacement of the corner node. For 2D L-shape (lower row), prediction is done for the X-displacement of the inner corner node (see insets). (a)&(c) Uncertainty predictions using MLE approach. MLE fails to capture the uncertainty outside the training region. (b)&(d) Uncertainty predictions using Variational Bayes approach. VB is capable to capture the uncertainty outside the training region.

of the FEM, solution time evolves with the value of applied force. This is because we use an iterative solver and adaptive load-stepping scheme to avoid convergence issues for large load cases. Hence on local, we can expect much more speedup than 31 times when we go towards the higher input force values. Deterministic U-Nets gave nearly 350 times speed up when predictions were done using GPU. Even with the high dimensional 3D examples, U-Net did not take more than 10 ms, thus satisfying the real-time constraint.

The time of prediction of Bayesian U-Net is the time of sampling over output distribution, which is as long as 300 stochastic forward passes in our case. For the above example (for

Type	dof	$t\_\text{fem}_{\text{CPU}}$ [s]	t_{CPU} [s]	t_{GPU} [s]	$\frac{t\_\text{fem}_{\text{CPU}}}{t_{\text{CPU}}}$	$\frac{t\_fem_{\text{CPU}}}{t_{\text{GPU}}}$
2D Beam	128	0.123	0.005	0.001	25	123
2D L-shape	256	0.120	0.007	0.001	17	120
3D Beam	12096	3.1	0.1	0.009	31	345

Table 4.6 Prediction times of deterministic U-Net on CPU, GPU. Under similar computational resources on CPU, U-Net shows 31 times speedup, which can be even more depending on the boundary conditions. Whereas GPU shows nearly 350 times speedup.

both 2D beam and 2D L-shape), 300 forward passes for a single test example took 0.1 secs. Compared to the deterministic case, the average time for the prediction of single-pass is less (0.3 ms) because of the efficient utilisation of batch prediction. Hence even Bayesian inference takes very little time in the prediction phase.

Training Times

For any neural network, the training phase of the model is the most resource-intensive task. Hence modern machine learning open source libraries such as Tensorflow, Keras, PyTorch are optimized to work with GPUs. GPU has a parallel structure that offers faster computing and increased efficiency compared to the user end computer with its CPU. Table 4.7 gives GPU training times for different datasets for both deterministic and probabilistic U-Nets.

Example	Dataset size, N	t_{train} [min]	N. of trainable parameters
2D Beam (D)	5700	131	7.5 E+6
2D Beam (VB)	5700	226	15.1 E+6
2D L-Shaped (D)	3800	78	7.5 E+6
2D L-Shaped (VB)	3800	143	14.6 E+6
3D Beam (D)	33688	1060	94.1 E+6

Table 4.7 U-Net training times, t_{train}. D = Deterministic, VB = Variational Bayes.

Bayesian U-Nets have more parameters to be trained, additionally, we need to sample over the approximate posterior as described in Section 4.3.4. Hence training times for Bayesian U-Nets are significantly higher than for the deterministic counterparts. As the size of the problem grows, training time proportionally increases as well. Hence the training time for the 3D beam case is higher compared to 2D cases. Note however, that this time can be reduced by opting alternate topologies of U-Nets, and one way of doing so is keeping a constant number of channels in each U-Net level instead of increasing it (which will be analyzed below).

Effect of number of channels

The training time of U-Net can be reduced by decreasing the number of trainable parameters of the model, and one of the ways to achieve this is to decrease the number of channels at each U-Net level. This can have, however, a side effect on prediction accuracy (intuitively, channels are partially responsible for capturing nonlinearities). We analyze these competing effects by performing a case study for the deterministic 3D-Beam case for architectures with different constant (not variable) number of channels, c.

Table 4.8 shows a comparison of training times and prediction errors. As we can see, as compared to the architecture used in Section 4.4.2, the use of 64 channels at each level gave comparable error values, while the training time is about three times lower. We can also observe that an excessive increase in the number of channels ($c = 128$) results in deterioration of not only training time but also the prediction accuracy, which can be interpreted as a well-known effect of overfitting. For reference, we have provided GPU prediction times for these networks as well.

N. of channels, c	$\bar{e}$ [m]	$\sigma(e)$ [m]	t_{train} [min]	t_{GPU} [ms]
16	1.6 E-3	0.9 E-3	272	6
32	1.1 E-3	0.7 E-3	293	6.5
64	**0.8 E-3**	**0.5 E-3**	**348**	7.5
128	3.5 E-3	3.3 E-3	646	9

Table 4.8 Error metrics, and training and predicting times for different deterministic U-Net architectures trained on the 3D dataset. A constant number of channels, c, is used in each level of U-Net. The use of 64 channels is optimum to achieve error and computational time trade-off. Standard 3D U-Net architecture took 1060 mins to train on the identical dataset.

4.5 Conclusions

In this work, we have proposed a deterministic/probabilistic neural network framework that is capable of accurately predicting large deformations in real-time. Although in the present work we only used artificially generated data for training, the framework can naturally assimilate experimental data as well. Because of these factors, our framework has the potential for data-driven applications requiring very fast response rates, such as patient-specific computer-aided surgery of soft human tissues.

In addition to the predictions, the proposed probabilistic framework is also capable of giving reliable uncertainty estimates. Indeed, we showed that the predicted uncertainties correlate with the prediction errors (fitting errors to FEM solution). We also showed that the uncertainties rapidly increase in the extrapolated region, which is the desired property that we expected

to achieve. Additionally, we were able to capture the noises present in the data, which has been validated with two probabilistic approaches (Maximum Likelihood Estimation and Variational Bayes). As such, our framework can be seen as a step towards making real-time large-deformation simulations more trustworthy.

To the best of our knowledge, this is the first time the state-of-the-art Bayesian Neural Networks are used in the context of non-linear body deformations. We believe that this work can serve as a reference for further developments in this emerging area of research. Due to its potentially high efficiency and accuracy, as well as due to its unique probabilistic predictive capabilities, we believe that the presented framework will turn out to be useful in a wide scope of novel engineering applications.

Besides showing promising results, we also demonstrated several important limitations of the current framework. Firstly, the convolution operations that are used in our U-Nets' implementation require structured meshes. We showed in the paper possible methods to extend our framework to unstructured meshes, which can be done with a moderate effort in the future. Secondly, we observed that the proposed novel technique to quantify uncertainties in extrapolated regions does not always give reliable predictions. Bayesian U-Nets failed to give reliable credible intervals of predictions when we applied the force on the nodes which were not part of the training procedure (i.e. extrapolated data in the spatial dimensions). As discussed in the paper, it seems to be a more fundamental and challenging problem that needs a dedicated approach, which is left for future research.

Chapter 5

MAgNET: A Graph U-Net Architecture for Mesh-Based Simulations

Abstract

In many cutting-edge applications, high-fidelity computational models prove too slow to be practical and are thus replaced by much faster surrogate models. Recently, deep learning techniques have become increasingly important in accelerating such predictions. However, they tend to falter when faced with larger and more complex problems. Therefore, this work introduces MAgNET: Multi-channel Aggregation Network, a novel geometric deep learning framework designed to operate on large-dimensional data of arbitrary structure (graph data). MAgNET is built upon the MAg (Multichannel Aggregation) operation, which generalizes the concept of multi-channel local operations in convolutional neural networks to arbitrary non-grid inputs. The MAg layers are interleaved with the proposed novel graph pooling/unpooling operations to form a graph U-Net architecture that is robust and can handle arbitrary complex meshes, efficiently performing supervised learning on large-dimensional graph-structured data. We demonstrate the predictive capabilities of MAgNET for several non-linear finite element simulations and provide open-source datasets and codes to facilitate future research.

This chapter is reproduced from: S. Deshpande, S.P.A. Bordas, J. Lengiewicz, *MAgNET: A Graph U-Net Architecture for Mesh-Based Simulations,* arXiv, 2023, https://doi.org/10.48550/arXiv.2211.00713

5.1 Introduction

Computational models are essential tools for studying, designing, and controlling complex systems in many fields, including engineering, physics, biology, economics, and social networks. These models are often based on physical laws and mathematical equations, with partial differential equations (PDEs) being a common tool for describing how quantities change over space and time. In mechanics and physics, the PDEs are most commonly solved with numerical methods upon earlier space- and time- discretization, and a large number of domain-specific computational models have been developed so far, with the finite element method (FEM) and the finite volume method (FVM) being the most commonly used approaches in solid- and fluid mechanics, respectively. However, despite significant advances in computational performance over the last decade, such high-fidelity numerical simulations remain prohibitively expensive for many important applications, including emerging areas such as real-time feedback/control in the computer-assisted surgery [Bui et al., 2018; Johnsen et al., 2015] or soft robotics [Goury and Duriez, 2018b; Rus and Tolley, 2015]. Speeding up such models whilst maintaining the desired accuracy is an active area of research, and one of the main motivations of the present work.

Recently, deep learning (DL) techniques have taken a center stage across many disciplines. The DL models have proven to be accurate and efficient in predicting non-trivial nonlinear relationships in data. For that reason, they have been tried for a variety of applications also in mechanics, such as surrogate modelling [Deshpande et al., 2022; Krokos et al., 2022a; Mendizabal et al., 2019a; Šarkić Glumac et al., 2023] or model discovery and calibration [Huang et al., 2020; Thakolkaran et al., 2022]. The deep neural network approaches can be categorized with respect to how they use the data and *a priori* knowledge about the modelled system. In purely data-driven approaches, DL models rely on performing supervised learning on either experimental or numerically generated data and are agnostic to the underlying physics or model. As such, they are able to reproduce the physics-based relationship by implicitly learning on a relatively large amount of data [Aydin et al., 2019; Daniel et al., 2020; Kochkov et al., 2021; Runge et al., 2017]. If the *a priori* information about the modelled system is introduced, such networks are termed as Physics Informed Neural Networks (PINNs) [Henkes et al., 2022; Klein et al., 2022; Raissi et al., 2019; Samaniego et al., 2020; Zhang et al., 2022]. With respect to the purely data-driven approaches, PINNs are generally more accurate, require less data for training, and possess better generalization capabilities. The framework presented in the present work is generally applicable to both cases, however, for the sake of clarity, we will later only focus on purely data-driven types of networks. In any case, once trained, the DL models can be used as fast surrogates for computationally expensive high-fidelity numerical methods.

The focus of the present work is on high-dimensional relationships in which the sizes of inputs and/or outputs are large. Examples of such relationships can be found, for instance, in

experimental full-field measurement data, such as our recent work on medical imaging [Lavigne et al., 2022], or in synthetic mesh data generated from finite element simulations, [Lorente et al., 2017; Pellicer-Valero et al., 2020]. Although DL techniques have generally shown great success as efficient surrogates to computationally expensive numerical methods in scientific computing, some of the popular existing machine learning approaches are still based on fully-connected deep networks which are not suitable for high-dimensional inputs/outputs. As an alternative, the application of Convolutional Neural Networks (CNNs) has proven a promising performance in a wide variety of applications, also including accelerating non-linear finite element/volume simulations [Deshpande et al., 2022; Krokos et al., 2022a; Obiols-Sales et al., 2020; Rao and Liu, 2020]. CNNs are designed to learn a set of fixed-size trainable local filters (convolutions), thus reducing the parameter space while being capable to capture non-linearities. In the context of computational mechanics, local convolutions leverage the natural local correlation of nearby nodes, which leads to more efficient neural network architectures, both in terms of training- and prediction times. Moreover, one can observe that the CNN architectures have a close analogy to some iterative solution schemes known in scientific computing [Brenner and Scott, 2008; Wang et al., 2020a]. This provides them with an additional interpretation of being trainable iterative computational schemes to solve sets of non-linear equations, rather than general-purpose black-box approximators.

However, there is one important limitation that prevents CNNs from being of general purpose. The problem is that they only work well with grid-like structure data, such as images or structured meshes, which greatly hinders their use for many real-world applications where data is structured differently. Although there are some attempts to alleviate that problem in the context of FEM data, for instance, combining finite elements with an immersed-boundary method [Brunet et al., 2019], or embedding a precomputed coordinate mapping into the classic grid [Gao et al., 2021], the effectiveness of those methods is limited to simple irregular domains and remains challenging for complex geometries in general. A definitive solution to that problem has only been brought by Graph Neural Networks (GNNs)–architectures that directly handle arbitrarily-structured inputs/outputs. They belong to the recently emerged family of Geometric Deep Learning (GDL) methods which focus on neural networks that can learn from non-Euclidean input such as graphs and, more generally, manifolds [Bronstein et al., 2017][Wu et al., 2021]. Because of their ability to handle more general structured data, GNNs are gaining increasing importance also in surrogate modelling in scientific computing [Sanchez-Gonzalez et al., 2020][Vlassis et al., 2020][Pfaff et al., 2021][Krokos et al., 2022b][Gao et al., 2022]. However, these approaches are based on relatively simple message passing schemes, which are sub-optimal for learning on high non-linear regression tasks. In this work, we propose a novel local aggregation technique, which we denote as Multichannel Aggregation layer, MAg, which performs multichannel localised weighted aggregations, that can be seen as a direct extension of the traditional convolution layer in CNNs. Thanks to that, we are able to directly adapt

some of the mechanisms/layers developed for CNNs to create efficient graph neural network architectures.

One mechanism that can improve the efficiency and predictive capabilities of convolutional and graph neural networks is the application of down-sampling (coarsening) and up-sampling (refinement) layers. In the context of CNNs, the focus is on encoder-decoder architecture frameworks, such as U-Net, which has been successfully implemented in various applications, including computer vision [Çiçek et al., 2016; Ronneberger et al., 2015], signal processing [Hennequin et al., 2020; Ren et al., 2021], and scientific machine learning [Le et al., 2022; Mendizabal et al., 2019a; Pant et al., 2021; Wang et al., 2020b]. While the CNN-based U-Net approaches are limited to grid data, their graph-based version, known as graph U-Net, can provide the desired generality. Recently, various graph coarsening approaches have been proposed [Bianchi et al., 2019; Cai et al., 2021; Gao and Ji, 2019; Lee et al., 2019], which serve the same function as pooling layers in CNNs, helping to reduce the size of a graph while maintaining essential properties of the processed data. In this work, we propose a novel graph pooling/unpooling operation (coarsening/refinement), that enables us to create a graph U-Net architecture, MAgNET, that can operate on arbitrary graphs. Our pooling layers are directly inspired by CNNs, where we extend the concept of pooling over local patches in regular grids to variable size non-overlapping cliques in graphs. This allows us to precompute coarsened graphs that are only based on the input graph topology, which is independent of data (i.e., node features). In the context of GNN-accelerated FEM simulations, a similar concept has been proposed by [Black and Najafi, 2022], however, their implementation is limited to regular meshes for simple two-dimensional geometries and linear elastic problems. Our approach enables computationally efficient deep learning models for non-linear problems involving arbitrary meshes, which is an important advancement for this field.

In summary, we introduce a novel graph U-Net framework comprising the proposed MAg and graph pooling/unpooling layers. The MAg layer captures local regularities in the input data, while the interleaved pooling layers reduce the graph representation to a smaller size while preserving important structural information. This enables us to efficiently implement our framework for large-scale problems. The proposed MAg and graph pooling layers are direct analogues of respective CNN U-Net layers and are also compatible with many state-of-the-art graph neural network layers. We elaborate on this point in the paper, providing a qualitative comparison of the proposed MAg layer with several existing graph layers. To validate the predictive capabilities of our framework, we apply it to several non-linear relationships obtained through finite element analysis and cross-validate it with predictions given by our CNN U-Net architecture [Deshpande et al., 2022]. To increase the impact of our work, we provide source codes, datasets, and procedures to generate the datasets utilized in this work, which can be found in the MAgNET repository: https://github.com/saurabhdeshpande93/MAgNET.

The chapter is organized as follows. In Section 5.2 we present the novel MAgNET framework, as well as its particular application to the hyperelastic FEM-based datasets. Then, in Section 5.3, we provide details of implementation and a thorough study of MAgNET for several 2D and 3D benchmark non-linear FEM examples. The conclusions and future research directions are summarised in Section 5.4.

5.2 MAgNET Deep Learning Framework

In this section, we will propose a novel graph-based encoder-decoder (U-Net) deep-learning framework. We will provide a general formulation, in which inputs and outputs follow a certain graph topology (that is expressed by an adjacency matrix $\mathbf{A}$). The graph expresses an assumed structure of correlations within input/output data and allows us to devise a *robust* DNN architecture defining a non-linear mapping between inputs and outputs. We will apply this general framework to *mesh-based* graphs. Such mesh topology of data is characteristic to spatially discretised numerical solution schemes for PDEs in scientific computing. In particular, we will focus on hyperelastic problems in solid mechanics, for which the training/testing data is obtained through the finite element method (see also the schematics in Figure 5.1).

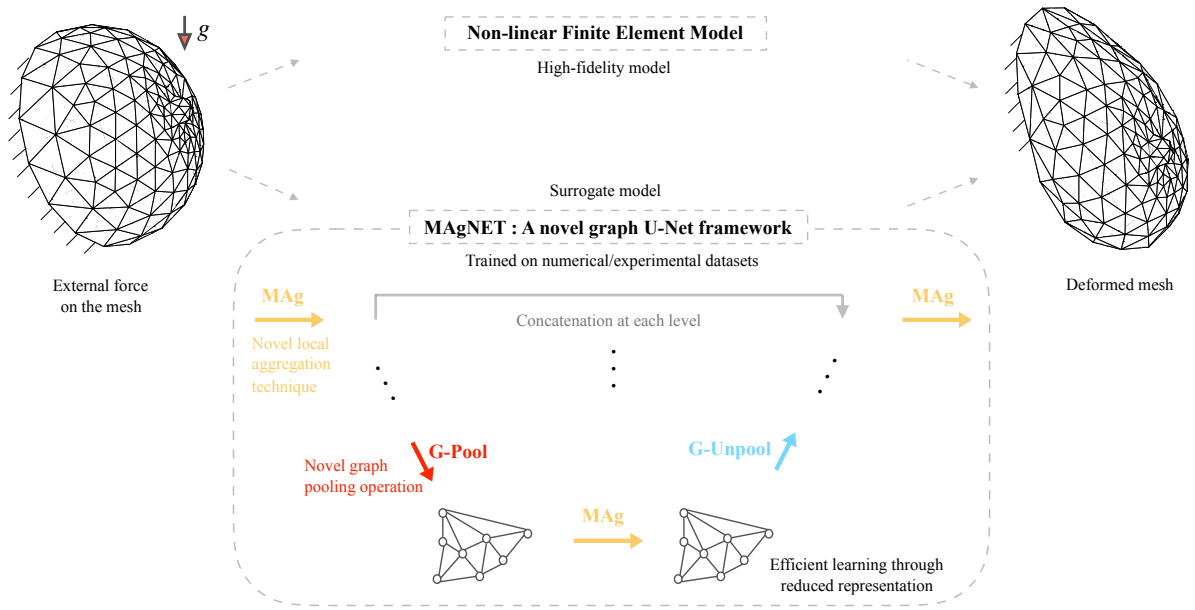


Fig. 5.1 A novel graph U-Net neural network surrogate model for mesh-based simulations. MAgNET accurately captures non-linear FEM responses.

In Section 5.2.1 we will provide an overview of the proposed Graph U-Net framework MAgNET. Next, in Sections 5.2.2-5.2.4 we will introduce the building blocks of MAgNET. In particular, in Section 5.2.2, we will introduce the adjacency matrix representation of the mesh-based

graph, which will be utilised later in this paper; and in Sections 5.2.3-5.2.4 we will specify a new graph Multi-channel Aggregation (MAg) layer, as well as new graph pooling/unpooling layers. Afterwards, in Section 5.2.5, we will provide an information-passing interpretation of the proposed Graph U-Net architecture. Finally, in Section 5.2.6 we will introduce a particular application of the framework to mesh-based datasets that are generated from FEM solutions of problems in hyper-elasticity.

5.2.1 MAgNET architecture overview

The MAgNET graph neural network architecture can be classified as a graph U-Net network and is an extension to the well-known class of convolution-based U-Net architectures (see [Ronneberger et al., 2015]). As such, the graph U-Net comprises of aggregation ('convolution'), pooling, unpooling, and concatenation layers (see the schematics in Figure 5.2), which are here suitably adjusted to work with general (non-grid) topologies of inputs/outputs.

The graph U-Net architecture has two stages: encoding and decoding. In the encoding stage, first, we apply one or more aggregation (MAg) layers, which are analogues of convolution layers in non-graph U-Net networks. Next, we apply a single graph pooling layer, which is a particular contraction of the graph, and which downsamples (coarsens) the problem. This aggregation-pooling sequence is repeated several times to achieve the desired level of contraction (coarsening). At the coarsest level, the MAg aggregation is performed one or more times, after which the decoding stage begins, which is the opposite to the encoding stage. At each level of decoding, the graph unpooling layer is followed by one or more MAg layers. At the upmost level, the last MAg layer is applied with linear activation to get the output.

More formally, the Graph U-Net network, $\mathcal{G}$, is constructed as follows. First, we set the input layer $\boldsymbol{d}^0$ as a vector of N nodes, each of which being a vector of input values (also known as features or channels) of a constant length c_0. (Further on, we will refer to the features as the *channels*.) Next, we subsequently add layers, $\boldsymbol{d}^l$, to form a U-Net architecture. The subsequent layers, $\boldsymbol{d}^{l-1}$ and $\boldsymbol{d}^l$, are linked by the following relationship

$$\boldsymbol{d}^l = \mathbf{T}^l(\boldsymbol{d}^{l-1}; \boldsymbol{\theta}^l), \tag{5.1}$$

where $\boldsymbol{\theta}^l$ is a vector of trainable parameters (e.g., weights and biases, $\boldsymbol{\theta}^l = \mathbf{k}^l \cup \mathbf{b}^l$), and $\mathbf{T}^l(\cdot)$ is one of three already introduced transformations: MAg(), gPool() or gUnpool(), which will be more precisely defined in the following sections. Additionally, we also consider remote concatenation links between respective layers from the encoding and decoding stages, see Figure 5.2. The output layer, $\boldsymbol{d}^L$, is assumed to be of the same mesh format as the input layer but can have a different number of channels (features), c_L. Finally, we define the Graph U-Net

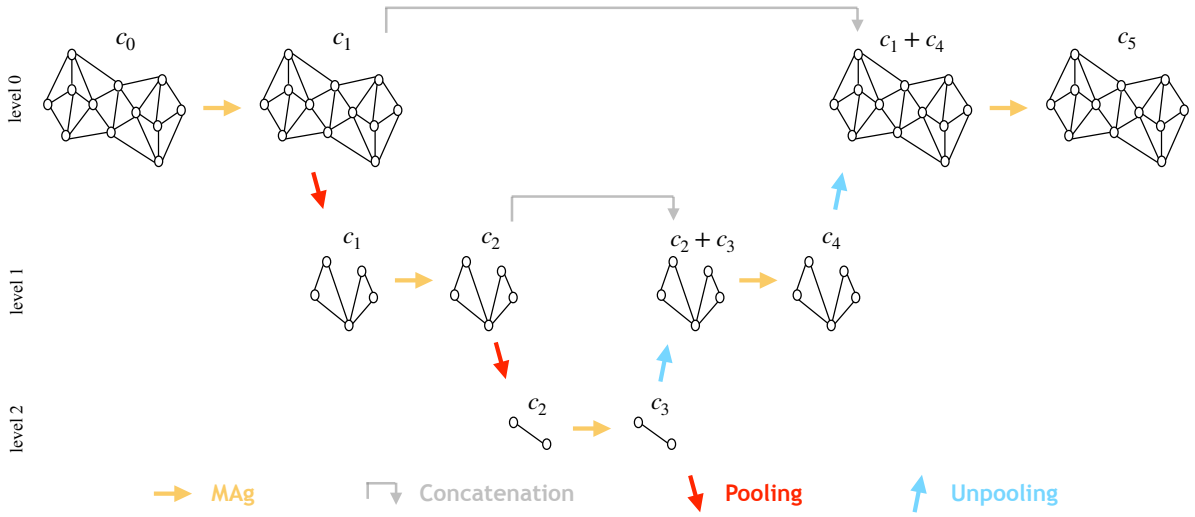


Fig. 5.2 A schematic of Graph U-Net architecture for mesh based inputs. Colors indicate different types of layers. $c_1, c_2, \ldots, c_5$ stand for channel dimensions. Different arrows indicate different layers: the graph Multi-channel Aggregation (MAg) layer, the graph pooling/unpooling layers, and the concatenation layer.

network as a parameterized transformation

$$\mathcal{G}(\boldsymbol{d}^0, \boldsymbol{\theta}) = \boldsymbol{d}^L = \mathbf{T}^L(\mathbf{T}^{L-1}(\mathbf{T}^{L-2}(\ldots); \boldsymbol{\theta}^{L-1}); \boldsymbol{\theta}^L), \tag{5.2}$$

where $\boldsymbol{\theta} = \bigcup_{l=1}^{L} \{\boldsymbol{\theta}^l\}$ is a concatenated vector of network parameters.

The calibration of the Graph U-Net parameters is done through a supervised learning, by fitting a given known input-output training dataset. The training dataset,

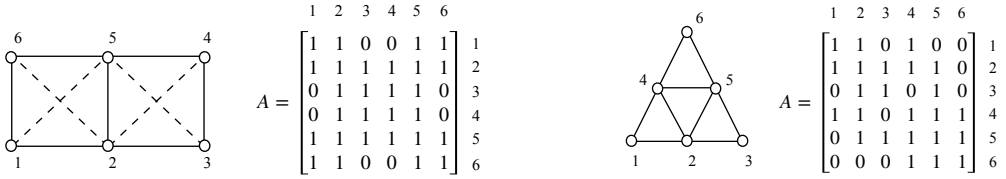
$$\mathcal{D}_{\text{tr}} = \{(\mathbf{f}_1, \mathbf{u}_1), ..., (\mathbf{f}_{M_{\text{tr}}}, \mathbf{u}_{M_{\text{tr}}}), \tag{5.3}$$

is in the mesh format, and the training is done by minimizing the following mean squared error loss function

$$\mathcal{L}(\mathcal{D}_{\text{tr}}, \boldsymbol{\theta}) = \frac{1}{M_{\text{tr}}} \sum_{m=1}^{M_{\text{tr}}} \|\mathcal{G}(\mathbf{f}_m, \boldsymbol{\theta}) - \mathbf{u}_m\|^2 \tag{5.4}$$

which gives the optimal parameters

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}_{\text{tr}}, \boldsymbol{\theta}). \tag{5.5}$$



(a) (b)

Fig. 5.3 Adjacency matrices for the (a) square and (b) triangular meshes. The dashed lines in (a) represent additional edges that are added to the original mesh.

5.2.2 Adjacency matrix of the mesh-based graph

For the purpose of this work, we will focus on sparse graphs that derive from data that is spatially organised in the form of meshes. Those can be 1D, 2D, or higher-dimensional meshes, of an arbitrary connection topology (see Figure 5.3 for examples of 2D meshes). The graph can be conveniently represented by a symmetric, square, Boolean adjacency matrix, A, whose order is equal to the number of nodes in the original mesh. To simplify the further notation, all nodes (vertices) are self-connected (have loops), which results in having 1 on the diagonal of $\mathbf{A}$. This allows us to more easily express certain graph operations that are used in this work, for instance, the k-th power of a graph $\mathbf{A}$, and the selection of pooling sub-graphs that is presented in Section 5.2.4.

It is fairly straightforward to generate an adjacency matrix from an element connectivity matrix of the mesh. For that reason we will not discuss it in detail. The only point to be emphasized is that we make all nodes belonging to a given element mutually inter-connected in the resulting graph (as they can be assumed to be strongly inter-related). We can visually represent it by adding more links as compared to a standard wire-frame visualization of finite-elements (see, e.g., the dashed lines in Figure 5.3a).

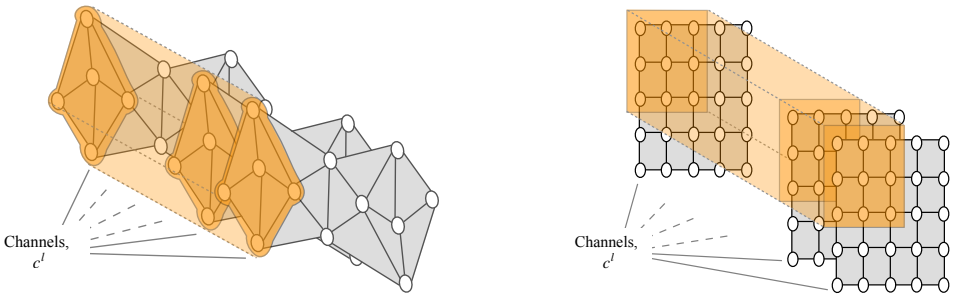
Remark: In our work we do not consider any attributes for the edges of a graph. Therefore, the data is only represented through nodal features and node-node connections which are defined through the adjacency matrix $\mathbf{A}$.

5.2.3 Multi-channel Aggregation (MAg) layer

The proposed novel neural network layer, MAg, is a multi-channel local aggregation layer that can operate on graph-structured data. Its architecture is a direct extension to the standard convolutional layer in CNNs, in which a shareable convolution window is used, making CNNs restricted to grid-structured data. In the MAg layer, instead, we propose to use fully-trainable local weighted aggregations (the so-called message passing scheme), where the aggregation

neighborhood of a given node is prescribed through the graph connectivity (the adjacency matrix). As such, the scheme is very well suited for sparse graphs and can be directly applied to graphs that derive from arbitrary 2D or 3D meshes.

The use of multiple channels aims to improve the capabilities of the network to capture non-linearities. In the multi-channel version, each node represents a vector of values (features), which can be visualised as multiple layers (channels) of the same graph structure (see the schematics in Figure 5.4a). The transformation between the input- and output multi-channel graphs is realised by applying multiple MAg aggregations on vector data to produce respective multiple components of output vectors. Note that the input/output channels of the whole network have usually a certain meaning, and their sizes are fixed (e.g., three RGB channels of a color image at the input and a single channel of a segmented image at the output). The number of channels in the latent layers can be chosen arbitrarily, which is up to the choice of a designer of a particular graph U-Net architecture.



(a) (b)

Fig. 5.4 Local aggregation in MAg (a) works very similar to the filter application in CNN (b). However as opposed to CNN, MAg uses different set of weights at different spatial locations with heterogeneous window size. In CNN, a constant filter slides across the channel.

More formally, we will consider the MAg layer as a parameterized transformation between the input and output nodes, defined as

$$d_{i,\alpha}^{l+1} = \sigma(b_{i,\alpha}^{l+1} + \sum_{\beta=1}^{c^l} \sum_{j \in \mathcal{N}_i} k_{i,j,\alpha,\beta}^{l+1} d_{j,\beta}^{l}), \tag{5.6}$$

where $\mathcal{N}_i = \{j \mid A_{ij} = 1\}$ is a set of neighbours of a node i to be aggregated, α and β represent the output and input channels, respectively, while $\mathbf{k}^{l+1}$ and $\mathbf{b}^{l+1}$ are trainable weights and biases, respectively. In this multi-channel definition, for a given component, $d_{i,\alpha}^{l+1}$, of an output, a single aggregation is performed throughout the neighborhood, $\mathcal{N}_i$, and all the input channels, $\beta \in \{1, \ldots, c^l\}$. The kernel parameters of MAg transformation, $k_{i,j,\alpha,\beta}^{l+1}$, are not shared, i.e, they can be independently trained for each aggregation window (note the free indexes i and α).

Comparison to existing graph aggregation/convolution layers

As already mentioned in the introduction, the very idea of generalisation of convolution layers to arbitrary graph structure is not new. In fact, various concepts have emerged so far, [Zhou et al., 2020][Chen et al., 2020b], most of which are compatible with the U-Net framework proposed in the present work. Below, we will discuss several of them, introducing a unified notation that will facilitate a qualitative comparison with respect to the proposed MAg layer, see Table 5.1.

Layer	Transformation
GCN [Kipf and Welling, 2016]	$d^{l+1}_{i,\alpha} = \sigma(\sum_{\beta=1}^{c^l} w^{l+1}_{\alpha,\beta} \sum_{j\in\mathcal{N}_i} \frac{A_{i,j}}{\sum_{k\in\mathcal{N}_i} A_{i,k}} d^l_{j,\beta})$
GAT [Veličković et al., 2017]	$d^{l+1}_{i,\alpha} = \sigma(\sum_{\beta=1}^{c^l} w^{l+1}_{t,\gamma,\beta} \sum_{j\in\mathcal{N}_i} \mathrm{softmax}_j(\mathrm{attn}(\boldsymbol{w}^{l+1}_t \boldsymbol{d}^l_i, \boldsymbol{w}^{l+1}_t \boldsymbol{d}^l_j, \boldsymbol{\theta}_\mathrm{t})) d^l_{j,\beta})$
SemGCN [Zhao et al., 2019]	$d^{l+1}_{i,\alpha} = \sigma(\sum_{\beta=1}^{c^l} w^{l+1}_{\alpha,\beta} \sum_{j\in\mathcal{N}_i} \mathrm{softmax}_j(k^{l+1}_{i,j,\alpha,\beta}) d^l_{j,\beta})$
MAg [present work]	$d^{l+1}_{i,\alpha} = \sigma(\sum_{\beta=1}^{c^l} \sum_{j\in\mathcal{N}_i} k^{l+1}_{i,j,\alpha,\beta} d^l_{j,\beta})$

Table 5.1 Comparison of the MAg layer with selected state of the art graph convolution layers (biases are omitted for the sake of brevity). In GAT formulation, $\alpha = (t-1)N_\mathrm{g} + \gamma$, which represents the stacking operation for the multi-head attention mechanism, where $t \in (1, \ldots, N^{l+1}_\mathrm{h})$ is the attention head index, and $\gamma \in (1, \ldots, N_\mathrm{g})$ is the internal channel index (cardinality of each node in the layer $l+1$).

Graph neural network layers aim to utilize the information about assumed correlations in data, with the graph structure expressing those correlations. The general approach is to specify a suitable (possibly nonlinear) trainable local transformation that can aggregate the information from a node in consideration and its neighbours. (This aggregation is followed by a chosen activation function before being propagated to the next layer.) Such transformations form a wide class of, so-called, message passing schemes, and can combine shareable (independent of a node) and non-shareable (dependent on a node, i.e., independently-trainable) sets of parameters.

The simplest and most lightweight realisations of the graph aggregation/convolution layer concept only utilise shareable weights, see, e.g., the Graph Convolutional Network (GCN), [Kipf and Welling, 2016]. In those approaches, a non-trainable (arbitrary) weighted aggregation is performed prior to application of a shareable trainable operator – something completely opposite to our MAg layer, which is fully trainable. This enables to keep the number of trainable parameters low, which is achieved at the cost of relatively low capacity of such networks. This low capacity can not be straightforwardly increased by simply deepening the network because of the well-known over-smoothing phenomenon.

We will discuss two out of many available approaches to increase the capacity of graph neural networks. The first approach relies on the multi-head attention mechanism which allows to assign different importance to nodes in the neighbourhood, see, e.g, the Graph Attention Network (GAT), [Veličković et al., 2017]. In the attention mechanism, the weights used in local aggregation depend on input nodal features, which makes the concept qualitatively different from all approaches (including the MAg layer) that use input-independent aggregation weights. The second class of approaches resemble the MAg layer more closely. Particular notable examples of that approach are the Spatial-Temporal Graph Convolution Network (ST-GCN), [Yan et al., 2018], and the Semantic Graph Convolution Network (SemGCN), [Zhao et al., 2019], which have been introduced in the particular context of human pose recognition problem (the computer vision domain). The common features of MAg and SemGCN layers are the input-independent learnable weighted aggregation and the use of channels to increase the model capacity. The difference is that the MAg doesn't use a shared transformation matrix ($\boldsymbol{w}$) nor the softmax normalisation – both used in the case of SemGCN.

To summarize, the proposed MAg layer relies on one of the most flexible message-passing schemes, with no shareable parameters. This promises a very high capacity of the MAg network. Also, as shown above, the proposed MAg layer is compatible with other graph convolution/aggregation layer concepts, and thus can be straightforwardly exchanged, if needed.

5.2.4 Graph pooling- and unpooling layers

Pooling and unpooling are two fundamental operations allowing U-Nets to encode (compress) and decode (decompress) information, respectively, see Figure 5.2. The *pooling* layers are composed of local contracting operations over the mesh-structured data, and are used to coarsen the data at the encoding part of the network. At the decoding part, the original refined mesh structure is restored by the *unpooling* layers (upsampling operations). In U-Nets, the unpooling layer is usually combined with the *concatenation* operation, which creates a direct link between the encoding and decoding part of the network (this will be explained later).

Graph pooling

In this work, we propose a novel clustering-based graph pooling layer that can be applied to arbitrary graph-structured data. It can be seen as an extension to the pooling layers known from CNN U-Nets that are limited to grid-structured data. In our graph pooling approach, we split the graph into disjoint cliques (fully-connected subgraphs), and perform the contraction of all the identified cliques (i.e., every clique is replaced by a vertex, and new edges represent formerly connected cliques), see Figure 5.5. The split into cliques is done statically, i.e., at the graph U-Net construction phase. In particular, the split does not depend on the input data.

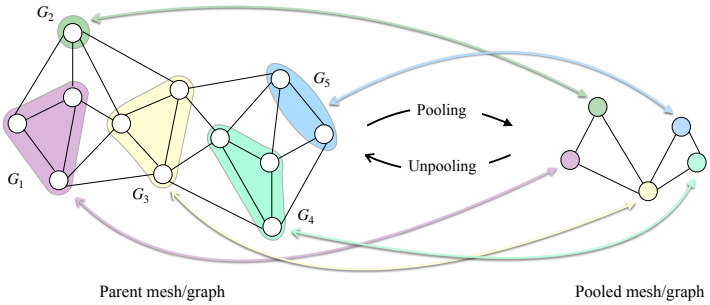


Fig. 5.5 One arbitrary choice of non-overlapping subgraphs to create a pooled graph. Subgraphs $\mathbf{G}_1, \ldots, \mathbf{G}_5$ are represented with different colors and are generated by the Algorithm 1.

Below, we will provide a more formal construction of the pooling layer. For a given input graph $\mathbf{G}$ that is represented by the vertices $\mathbf{S}$ and the connectivity matrix $\mathbf{A}$, we first generate an arbitrary set of $\tilde{N}$ non-overlapping fully-connected subgraphs (cliques) $\mathbf{G}_1, \mathbf{G}_2 ..., \mathbf{G}_{\tilde{N}}$, i.e.,

$$\mathbf{S} = \bigcup_{i=1}^{\tilde{N}} \mathbf{S}_i, \qquad \forall_{\mathbf{S}_i} \forall_{j,k \in \mathbf{S}_i} A_{jk} = 1 \qquad \text{and} \qquad \forall_{i \neq j} \mathbf{S}_i \cap \mathbf{S}_j = \emptyset, \tag{5.7}$$

where the sets $\mathbf{S}_i$ represent nodes of the respective subgraphs $\mathbf{G}_i$. The procedure to generate these subgraphs and the respective pooled adjacency matrix $\tilde{\mathbf{A}}$ is described in Algorithm 1. The pooled graph $\tilde{\mathbf{G}}$ is composed of vertexes $\tilde{\mathbf{S}} = \{1, \ldots, \tilde{N}\}$ with edges defined by the adjacency matrix $\tilde{\mathbf{A}}$. The pooling layer is described as:

$$d_{i,\beta}^{l+1} = \underset{j \in \mathbf{S}_i}{\text{aggr}}\, d_{j,\beta}^{l}, \tag{5.8}$$

where the 'aggr' operation can be a max/min/avg, etc. Note that graph pooling layers do not modify the number of channels, i.e., the pooling is performed individually per each channel of the input.

Graph pooling can be applied several times at the encoding part of the U-Net, e.g., see Figure 5.2. For the purpose of future unpooling operations, after each pooling operation, we save the original graph, $\mathbf{G}$, the adjacency matrix, $\mathbf{A}$, and the pooling subgraphs, $\mathbf{G}_i$. After doing so, we substitute $\mathbf{G} \leftarrow \tilde{\mathbf{G}}$, and $\mathbf{A} \leftarrow \tilde{\mathbf{A}}$.

Graph unpooling

Structure-wise, the graph unpooling is a reverse operation to pooling. More precisely, the output graph of an unpooling layer will have the same topology as the input graph of the related pooling layer, see Figure 5.5. The operation is defined via the previously saved subgraphs $\mathbf{G}_j$ (with nodes $\mathbf{S}_j$) as

$$d_{i,\beta}^{l+1} = d_{j,\beta}^{l} \qquad \text{for} \quad i \in \mathbf{S}_j, \tag{5.9}$$

Algorithm 1: Generate a pooled graph from an arbitrary parent graph

Input: $N \times N$ adjacency matrix, A
Result: list of subgraphs, S; $\tilde{N} \times \tilde{N}$ pooled adjacency matrix, $\tilde{A}$

$S \leftarrow \{\}$ /* initialisation of the subgraph list */
$P \leftarrow \{1, 2, ..., N\}$ /* node indices of the parent graph */
$A' \leftarrow A$ /* temporary copy of matrix A */
/* Loop for generating non-overlapping subgraphs, S, see Figure 5.5 */
while $P \neq null$ **do**
  $p \in P$ /* randomly select a single node */
  $S_i \leftarrow \{p\}$ /* initialise subgraph */
  $\mathcal{N}_p \leftarrow \{m \neq p \mid A'[m, p] = 1\}$ /* nodes connected to selected node */
  for n *in* $\mathcal{N}_p$ **do**
    if $\forall\ m \in S_i\ \ A'[m, n] = 1$ **then**
      $S_i \leftarrow S_i \cup n$ /* append node to the subgraph */
    end
  end
  $P \leftarrow P \setminus S_i$ /* remove subgraph from parent graph */
  $\forall\ m \in S_i\ \ A'[m, :] \leftarrow 0;\ A'[:, m] \leftarrow 0$ /* remove subgraph from parent graph */
  $S \leftarrow S \cup S_i$ /* add subgraph to subgraphs list */
end
$\tilde{N} = \text{sizeof}(S)$ /* number of pooled nodes = number of pooling subgraphs */
$\tilde{A} \leftarrow \text{zeros}(\tilde{N}, \tilde{N})$ /* zero initialisation of pooled matrix */
/* Loop for constructing pooled adjacency matrix $\tilde{A}$ from subgraphs S */
for r *in* $\{1, 2, .., \tilde{N}\}$ **do**
  for c *in* $\{1, 2, .., \tilde{N}\}$ **do**
    if $\exists\ n \in S[r],\ m \in S[c] \mid A[n, m] = 1$ **then**
      $\tilde{A}[r, c] \leftarrow 1$ /* if $S[r]$ and $S[c]$ are connected by an edge */
    end
  end
end

and it simply replicates the features of a node j to the nodes specified by $\mathbf{S}_j$. As such, this operation is analogous to the related upsampling operation used in CNNs.

Graph unpooling + concatenation

Concatenations, also known as skipped connections, are characteristic to U-Net architectures. Thanks to them, the layers from the decoder part gain a direct access to features from the encoder part. Concatenations help to mitigate the issue of vanishing gradients, and add extra information that could have been lost due to the earlier downsampling (pooling).

In our case, the concatenations are always related to the respective pooling/unpooling operation pairs, see Figure 5.2. It is done by stacking the output of an unpooling layer l, given by

Equation (5.9), with the input of a respective pooling layer l':

$$d^{l+1}_{i,c^l+\alpha} = d^{l'}_{i,\alpha}. \tag{5.10}$$

In the formula above, c^l is the number of channels in unpooling inputs. As the result, the total number of output channels of unpooling+concatenation is $c^l + c^{l'}$.

5.2.5 Information-passing interpretation of MAg and pooling layers

During a single forward pass of the MAg layer, the aggregation is performed locally for each individual node, i.e., each node of the graph will have an access to the aggregated feature information from its adjacent nodes only, specified by the adjacency matrix, $\mathbf{A}$, see Equation (5.6). Therefore, the nodes that are not directly connected through $\mathbf{A}$ do not exchange information at a single MAg operation (see Figure 5.6). Such long-distance exchange across the network is fundamental to allow the neural network model to express correlations between topologically distant input- and output nodes (e.g., how the output displacements at node C depend on the input loads at the node B, in Figure 5.6).

One way to handle this issue would be to apply the MAg layer several times as shown in Figure 5.6. However, in that case, the number of subsequent layers would be proportional to the diameter of the underlying graph, which could increase the number of training variables and the depth of the network, deteriorating its performance. A natural simple improvement, also utilised in the present paper, is to increase the support (neighbourhood) of the MAg operations. In the proposed framework, this can be straightforwardly done by using higher powers of the adjacency matrix, e.g., $\mathbf{A}^2$ or $\mathbf{A}^3$, instead of $\mathbf{A}$. This improvement alone, however, would still require the number of MAg layers to be proportional to the graph diameter.

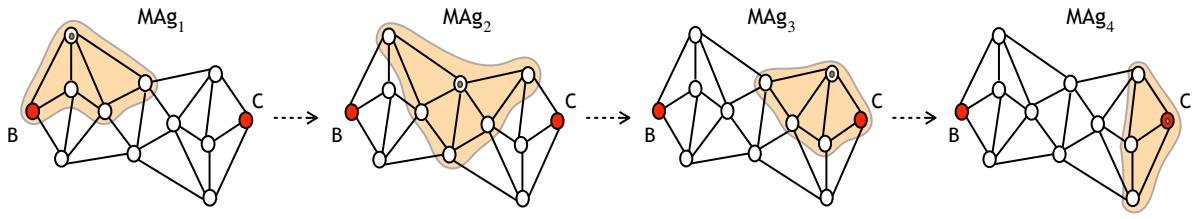


Fig. 5.6 This 2D mesh requires at least 4 subsequent local aggregation operations (orange areas with center nodes marked by dots) to propagate the feature information from node B to the distant node C.

The above observations explain a natural motivation behind using the pooling/unpooling layers, and hence creating the U-Net architecture. The pooled graph can be seen as a reduced space representation of the parent graph, and each pooled node aggregates the feature information

corresponding to multiple nodes of the parent graph, see Figure 5.7. The pooled graph is of a coarsened topology when compared to the parent graph, and this allows for the feature information exchange with a lower number of MAg layers. The pooling/unpooling layers can be nested, which provides an exponential reduction rate of the graphs' diameters.

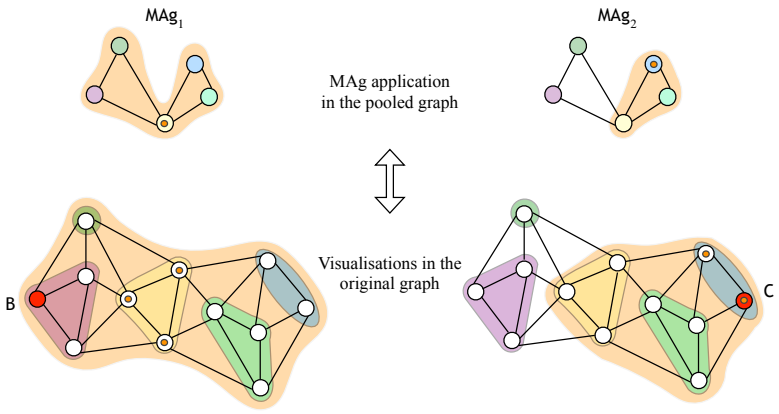


Fig. 5.7 Visualisation of feature information exchange between nodes in the pooled graph. In the pooled space, only 2 MAg operations are sufficient to exchange feature information between spatially further located nodes in the original graph. The orange region shows the window of MAg operation.

Remark: Note that the pooling layer proposed in this work represents a clique-pooling approach in which the cliques are non-overlapping. This allows us to achieve a very good level of graph coarsening (contraction). It is unlike a similar clique-based strategy that has been recently proposed, [Luzhnica et al., 2019], in which the pooling cliques overlap, providing a much lower level of coarsening.

5.2.6 Application to FEM-based datasets

We will now focus on a particular graph structure of inputs/outputs that will be in a form of finite element mesh. Specifically, the MAgNET framework will be applied as a surrogate model to a finite element model in large-deformation elasticity. The finite element model will be shortly introduced below.

We consider a boundary value problem expressed in the weak form over the domain Ω:

$$\int_{\Omega} \boldsymbol{P}(\boldsymbol{F}(\boldsymbol{u})) \cdot \nabla \delta \boldsymbol{u} \, dV - \int_{\Omega} \rho \, \bar{\boldsymbol{b}} \cdot \delta \boldsymbol{u} \, dV - \int_{\Gamma_t} \bar{\boldsymbol{t}} \cdot \delta \boldsymbol{u} \, dS = 0 \qquad \forall \delta \boldsymbol{u}, \tag{5.11}$$

where $\boldsymbol{P}(\bullet)$ is the first Piola-Kirchhoff stress tensor, $\bar{\boldsymbol{b}}$ are prescribed body forces, $\bar{\boldsymbol{t}}$ are prescribed tractions on the Neumann's boundary Γ_N, while the solution $\boldsymbol{u}$ and the variation $\delta \boldsymbol{u}$ belong

to appropriate functional spaces, with $\boldsymbol{u} = \bar{\boldsymbol{u}}$ and $\delta\boldsymbol{u} = \mathbf{0}$ on the Dirichlet boundary Γ_u. The hyperelastic constitutive relationship is expressed through the strain-energy density potential $W(\boldsymbol{F})$ as

$$\boldsymbol{P}(\boldsymbol{F}) = \frac{\partial W(\boldsymbol{F})}{\partial \boldsymbol{F}}, \tag{5.12}$$

where the deformation gradient tensor $\boldsymbol{F} = \mathbf{I} + \nabla\boldsymbol{u}$.

For all the cases considered in the present work, the Neo-Hookean hyperelastic law with the following strain energy potential is used, see Simo and Taylor [1982],

$$W(\boldsymbol{F}) = \frac{\mu}{2}(I_c - 3 - 2\ln J) + \frac{\lambda}{4}(J^2 - 1 - 2\ln J), \tag{5.13}$$

where the invariants J and I_c are given in terms of deformation gradient $\boldsymbol{F}$ as

$$J = \det(\boldsymbol{F}), \quad I_c = \mathrm{tr}(\boldsymbol{F}^T\boldsymbol{F}), \quad , \tag{5.14}$$

with μ and λ being Lame's parameters, which can be expressed in terms of Young's modulus, E, and Poisson's ratio, ν, as

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}, \quad \mu = \frac{E}{2(1+\nu)}. \tag{5.15}$$

Note that one can use other forms of the volumetric part of the above potential, see Doll and Schweizerhof [2000], or other hyperelastic models, such as the Mooney-Rivlin and a more general class of Ogden models, see Ogden [2005].

Finite element discretisation transforms the weak form expressed by Eq.(7.14) into the system of non-linear equations

$$\mathbf{R}(\mathbf{u}; \mathbf{f}_{\text{ext}}) = \mathbf{f}_{\text{int}}(\mathbf{u}) - \mathbf{f}_{\text{ext}} = \mathbf{0}, \tag{5.16}$$

that expresses the balance between external and internal nodal forces. In this work, the vector of external forces, $\mathbf{f}_{\text{ext}}$, represents boundary conditions, which can be surface tractions or body forces. Given $\mathbf{f}_{\text{ext}} = \mathbf{f}_m$, the system is solved for an unknown vector $\mathbf{u}$ with the Newton-Raphson scheme, giving as a result the solution $\mathbf{u}_m$. A pair $(\mathbf{f}_m, \mathbf{u}_m)$ makes an element of the dataset $\mathcal{D}$ introduced in Eq. (5.3), and the FE mesh that results from the FE discretization produces the adjacency matrix $\boldsymbol{A}$ introduced in Section 5.2.2.

5.3 Results

In this section, we will study the performance of the proposed framework, and for that purpose, we use four benchmark problems. In Section 5.3.1, we give a detailed specification of the

benchmark problems and the procedure for obtaining FEM-based datasets. In Section 5.3.2, we provide details of neural network architectures for each of the studied cases and will describe the training procedure. In Section 5.3.3, we study the predictions of neural network models by cross-validating results from MAgNET and CNN models, and by comparing them with the FEM results. Finally, in Section 5.3.4 we demonstrate the capabilities of the MAgNET framework to provide a surrogate model for the unstructured mesh cases.

5.3.1 Generation of FEM based datasets

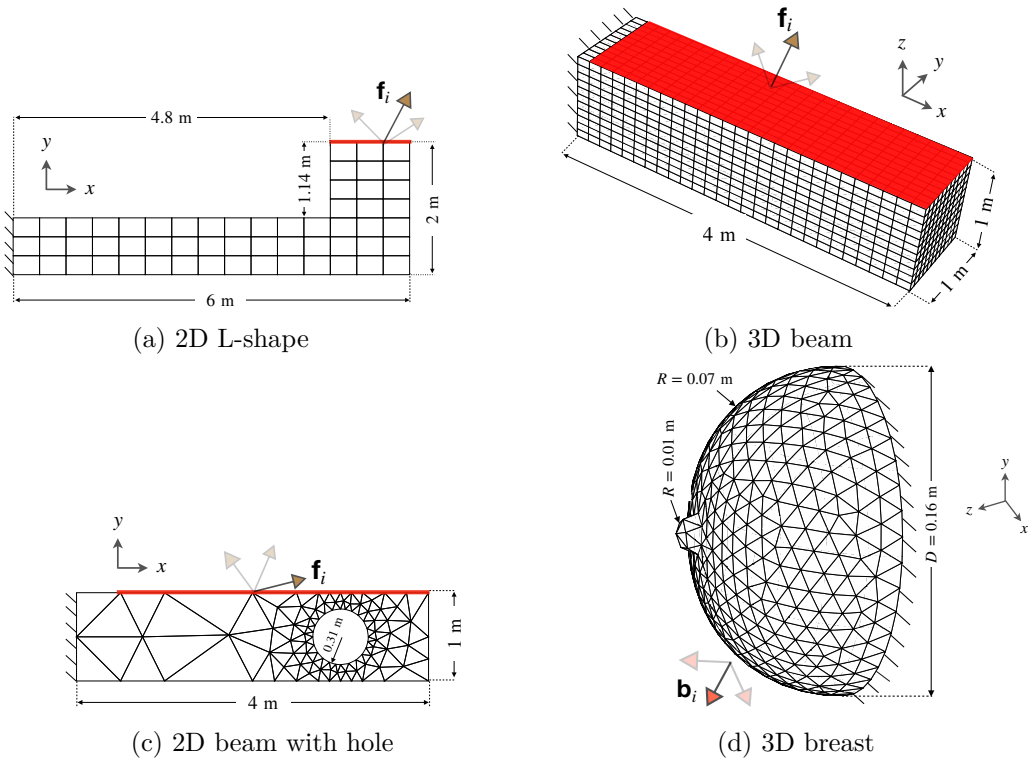
We consider four benchmark problems, see Figure 5.8. Two of them, Figure 5.8(a-b), utilise simple meshes, which makes it possible to assure structured (grid) inputs. They will be used to cross-validate between our MAgNET architecture and the standard CNN U-Net architecture. The other two examples, Figure 5.8(c-d), are more complex and will serve us to demonstrate the applicability of MAgNET for general (unstructured) meshes. Each of those two groups consists of a 2D and a 3D problem, thanks to which the framework can be tested for four different finite element topologies: triangular, quadrilateral, tetrahedral, and hexahedral.

For all considered cases, we use the neo-Hookean material model, see Section 5.2.6, with material parameters provided in the Table 5.2. In order to generate training/testing datasets, for each discretized problem we individually specify a family of boundary conditions, as described below, see also schematics in Figure 5.8. For the cases shown in Figures 5.8(a-c), nodes on one side are fixed (Dirichlet boundary conditions), and only a single random nodal force is applied at a selected node in a prescribed region of interest (denoted by red line/surface in Figure 5.8(a-c)). For the remaining nodes, the external forces are set to $\mathbf{0}$. For the case shown in Figure 5.8(d), the uniform body force density is prescribed (force per unit mass, with density $\rho = 1000$ kg/m^3). The body force field is integrated through element shape functions to obtain respective nodal forces that are used in datasets.

	Problem (element topology)	Is structured?	Young's modulus, E [Pa]	Poisson's ratio, ν	Density, ρ [kg/m^3]
a)	2D L-shape (quad)	No (Yes)	500	0.4	-
b)	3D beam (hexahedron)	Yes	500	0.4	-
c)	2D beam with hole (triangular)	No	500	0.3	-
d)	3D breast (tetrahedron)	No	800	0.4	1000

Table 5.2 Material properties used for the benchmark cases.

All the finite element computations were implemented and performed within the AceGen/AceFem framework [Korelc, 2002]. For a given problem, for each loading case, i, the entire vector $\mathbf{f}_{(i)}$ of external nodal forces and the vector $\mathbf{u}_{(i)}$ of computed nodal displacements were saved, which allowed to generate the final training/testing dataset $D = \{(\mathbf{f}_{(1)}, \mathbf{u}_{(1)}), ..., (\mathbf{f}_{(M_{\text{tr}}+M_{\text{te}})}, \mathbf{u}_{(M_{\text{tr}}+M_{\text{te}})})\}$.



(a) 2D L-shape

(b) 3D beam

(c) 2D beam with hole

(d) 3D breast

Fig. 5.8 Schematics of four benchmark problems. (a) 2D L-shape geometry (quad mesh), (b) 3D beam geometry (hexahedron mesh), (c) 2D beam with hole geometry (triangular mesh), and (d) 3D breast geometry (tetrahedron mesh). In examples (a)-(c), single nodal loads are applied on the region of boundary indicated with red color. In example (d), only body forces are considered.

The datasets were randomly split into training sets, M_{tr} (95%), and testing sets, M_{te} (5%). The sizes of datasets and the distribution of force magnitudes are provided in Table 5.3.

	Problem	N.of FEM DOFs ($\mathcal{F}$)	Range (External forces/ body force density)	Dataset size $M_{\text{tr}} + M_{\text{te}}$	samples per node
a)	2D L-shape	160	f_x, f_y = -1 to 1 N	3800 + 200	1000
b)	3D beam	12096	f_x, f_y, f_z= -2 to 2 N	33858 + 1782	110
c)	2D beam (hole)	198	f_x, f_y = -5 to 5 N	4560 + 240	400
d)	3D breast	3105	b_x, b_y = -6 to 6 N/kg , b_z = -3 to 3 N/kg	7600 + 400	-

Table 5.3 Specification of FE-based datasets. For cases (a-c), the external force is applied at a selected node, and for case (d), external body forces are applied. The magnitudes of forces are randomly sampled from the multivariate uniform distribution, with ranges specified in the table. For cases (a-c), multiple samples per node are generated, for all nodes in the prescribed area of interest.

5.3.2 Design, implementation and training of neural network models

The implementation of the layers and mechanisms of the MAgNET framework described in Section 5.2 and of CNN U-Net framework introduced in [Deshpande et al., 2022] has been performed within the TensorFlow libraries. We use them to build and train deep neural network models for the cases described in Section 5.3.1. Table 5.4 outlines individual properties of the network architectures implemented in this work. The codes and datasets are publicly available open source [Deshpande et al., 2023a], which makes it possible for other researchers to reproduce the present results and also to extend our frameworks to new cases/problems.

Example	Network type	(N. of poolings, N. of MAg/conv. layers per level, window size)	N. of channels per level	N. of parameters
2D L-shape	MAgNET CNN U-Net	(3, 2, A^2) (2, 2, 3×3)	16, 32, 64, 128 64, 128, 512	∼ 4 E+6
3D beam	MAgNET CNN U-Net	(5, 1, A^2) (4, 2, 3×3×3)	3, 3, 3, 12, 24, 48 256, 256, 256, 512, 512	∼ 75 E+6
2D beam (hole)	MAgNET	(3, 2, A^2)	8, 16, 32, 64	∼ 2 E+6
3D breast	MAgNET	(4, 1, A^2)	6, 12, 12, 24, 48	∼ 19 E+6

Table 5.4 Neural network architectures implemented in this work. The leaky ReLU activation function is used in all MAgNET cases, while ReLU activation is used for CNN cases. For the last layers, the linear activation function is always applied.

To provide a complete understanding of the neural network architectures listed in Table 5.4, we will now delve into the details of the MAgNET architecture used for the 2D L-shape example. Its schematics is shown in Figure 5.9. As indicated in the third column in Table 5.4, it is a three-level graph U-Net architecture with two MAg operations at each level. The fourth column specifies the number of channels utilized for the MAg operations at each level of the graph U-Net. The forward pass starts with the input mesh (2D), to which the MAg layer is applied twice (with 16 output channels). This is followed by the graph pooling layer, which coarsens the mesh and transitions to the next level of the U-Net (from zeroth to the first level). This process repeats twice, with the first and second levels of the graph U-Net having MAg layers with 32 and 64 output channels, respectively, leading to the coarsest third level of the U-Net. At this level, two MAg layers (with 128 output channels) are applied. In the subsequent decoding phase, the graph unpooling layer is employed with the concurrent concatenation operation, and followed by two MAg operations (with 64 output channels). This upsampling sequence repeats twice with the use of 32- and 16-channel MAg layers. Finally, a single MAg layer (with 2 output channels) is applied, using a linear activation to produce the desired output mesh (the displacement mesh must have the same structure as the input mesh of forces). It is worth noting that analogous architectures of CNN U-Net networks are similar, with the only

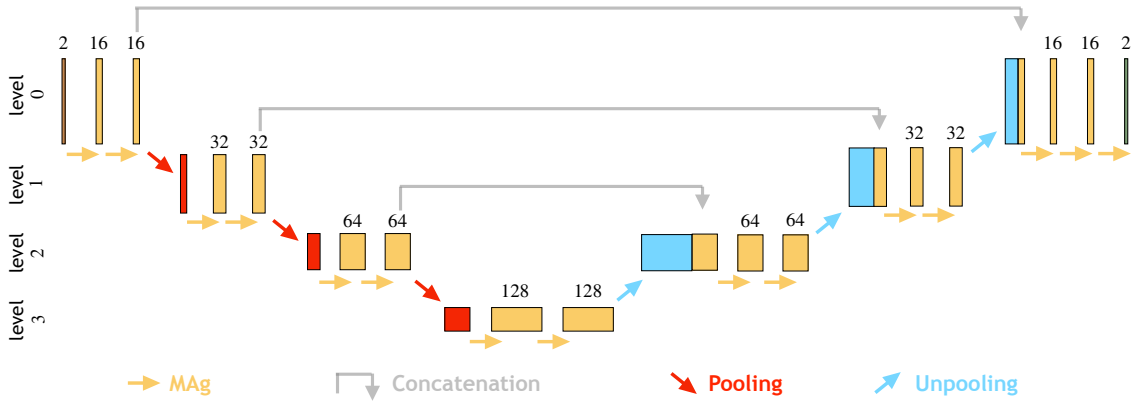


Fig. 5.9 MAgNET architecture used for the 2D L-shape example.

distinction being the use of convolution layers in place of MAg layers and CNN U-Net max poolings instead of graph poolings.

As demonstrated in Table 5.4, both 2D L-shape and 3D beam examples have been modeled utilizing both MAgNET and CNN U-Net architectures. These networks were designed to have a similar number of trainable parameters, thus facilitating a fair comparison of their fitting capabilities. The number of parameters in both types of networks is controlled by having a higher number of channels in the CNN U-Net architecture compared to its corresponding MAgNET architecture. This difference in the number of channels is attributed to the convolution operators in the CNN architecture sharing parameters across a layer, which may necessitate a larger number of channels to ensure an optimal fit, while the aggregation operators in the MAg layer use individual weights per aggregation window, allowing for more flexible fitting across the mesh with a smaller number of channels. However, caution must be exercised when selecting the number of channels, as setting it too low can result in increased prediction errors (as seen in Figure 5.15).

The number of neural network levels (pooling operations) and the size of convolution/aggregation windows have been adjusted on a case-by-case basis to obtain the desired fitting capabilities while keeping the number of trainable parameters low and comparable between the respective CNN U-Net and MAgNET models. The fitting capabilities heavily depend on the successful propagation of information from the input throughout the network. This can be compromised when the number of poolings or the window size is too small, as explained in Section 5.2.5. For this reason, a larger number of pooling operations is used for mesh graphs with larger diameters (e.g., the 3D cases in Table 5.4). Additionally, in the case of MAgNET models, a global optimization of graph pooling operations is performed to reduce the number of nodes at the coarsest level. In this optimization, Algorithm 1 is run 1000 times with different random seeds, and the case with the least number of nodes at the lowest level is selected.

Remark: Note that the example presented in Figure 5.8(a) utilizes a non-structured mesh. As such, it can not be directly used by the CNN U-Net model, and an additional preprocessing step needs to be done to make the input and output meshes structured. In this case, we apply zero padding to convert the L-shape mesh into a structured mesh, see Figure 5.10, which is then used for training with the CNN U-Net architecture. We do not need to do this preprocessing step for the MAgNET architecture.

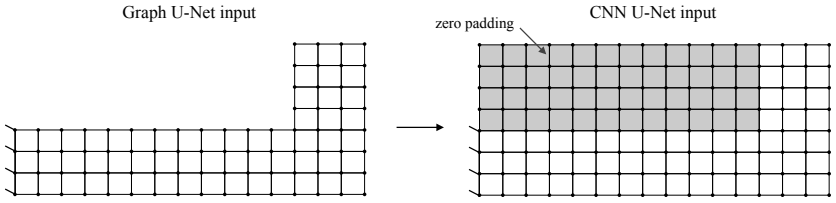


Fig. 5.10 Zero-padding is applied to make the L-shape topology compatible with the CNN framework. The additional nodal values for inputs (forces) and outputs (displacements) are set as zero vectors.

The models presented in Table 5.4 were trained by minimizing the loss function, as described in Equation 7.2, using the datasets introduced in Section 5.3.1. The Adam optimizer, an extension of the stochastic gradient descent algorithm, was used for this purpose. A mini-batch size of 4 and an initial learning rate of 1×10^{-4}, with a linear decay to 1×10^{-6} during training, were employed. The number of epochs (i.e., iterations of the Adam optimizer) was manually tailored on a case-by-case basis to achieve low values of the loss function. An example of the training loss is provided in Figure 5.11. The network trainings were conducted using TensorFlow on a Tesla V100-SXM2 GPU at the HPC facilities of the University of Luxembourg, see [Varrette et al., 2014].

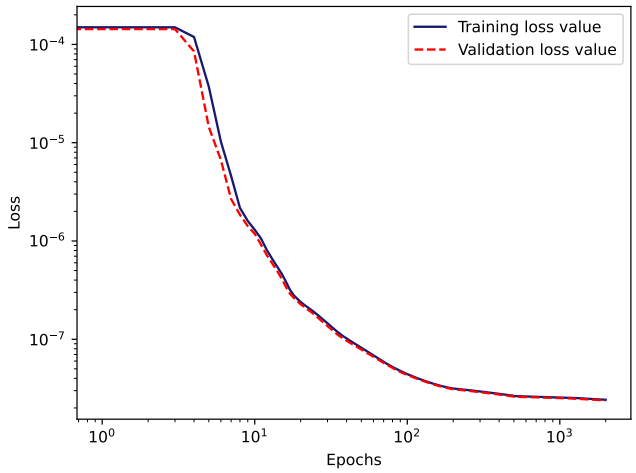


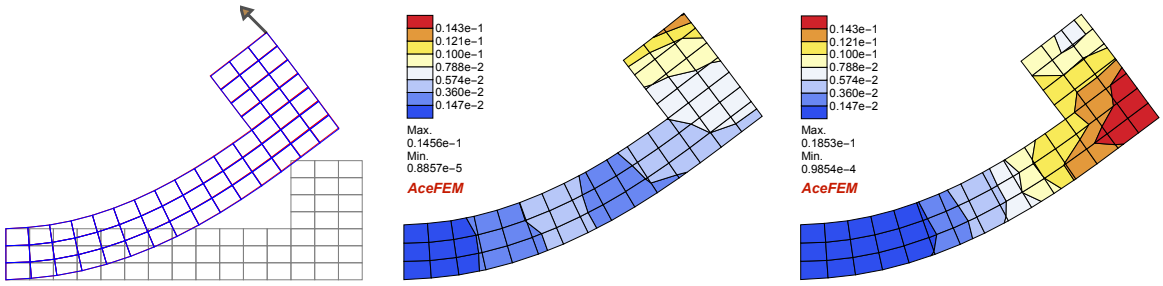
Fig. 5.11 Training loss curve for the 3D breast MAgNET model.

5.3.3 Cross validation of CNN U-Net and MAgNET predictions

We are going to compare the predictions of MAgNET and CNN U-Net models for two problems with structured inputs/outputs that were introduced in Figure 5.8(a,b). Let us look at the individual examples with the highest nodal displacement magnitudes. In the 2D L-shape example, shown in Figure 5.12a, MAgNET predictions visually coincide with the reference FEM solution very well. This is quantitatively shown in Figures 5.12b and 5.12c, where the the L_2 error field

$$\mathrm{err}(\mathbf{X}) = ||u_{\mathrm{FEM}}(\mathbf{X}) - u_{\mathrm{pred}}(\mathbf{X})||_2 \tag{5.17}$$

is presented for MAgNET and CNN U-Net, respectively, demonstrating low level of errors for both models. A similar tendency can be observed in the 3D beam case shown in Figure 5.13. Here, although the level of errors is relatively a bit higher than in the 2D example, the MAgNET and CNN U-Net perform similarly, which proves good capabilities of the proposed MAgNET model as compared to the CNN U-Net model.



(a) (b) (c)

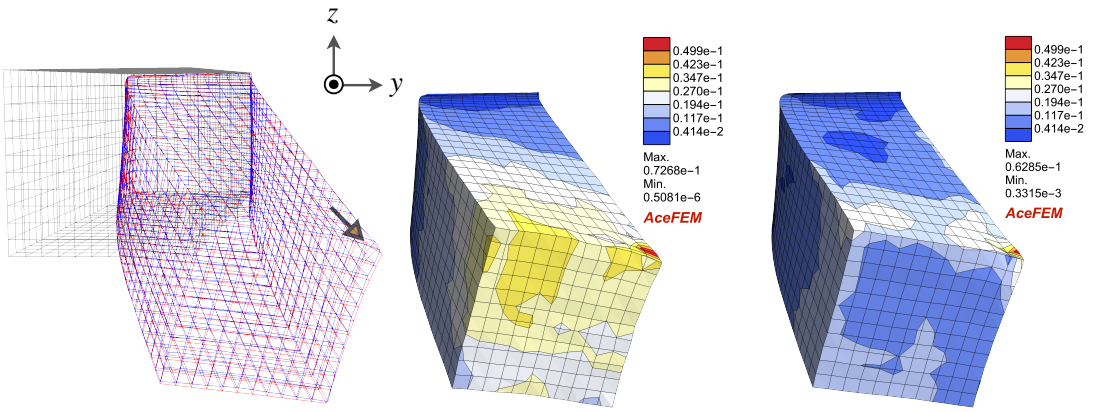
Fig. 5.12 Deformation of 2D L-shape under point load (-0.93, 0.91)N on the corner node (a) Deformed mesh predicted using MAgNET (blue), for comparison FEM solution is presented (red) (b) L_2 error of nodal displacements between MAgNET and FEM solution. The relative error for the corner node displacement using MAgNET is 0.5% (c) L_2 error of nodal displacements between CNN U-Net and FEM solution. The relative error for the corner node displacement using CNN is 0.3%.

In the following, we will analyze and compare the performance of both models for all cases in the text datasets. For that purpose, we need aggregated error metrics. As an error metric for a single test example, we use the mean absolute error,

$$e_m = e(\mathbf{f}_m, \mathbf{u}_m) = \frac{1}{\mathcal{F}} \sum_{i=1}^{\mathcal{F}} |\mathcal{G}(\mathbf{f}_m)^i - \mathbf{u}_m^i|, \tag{5.18}$$

where the force-displacement pair $(\mathbf{f}_m, \mathbf{u}_m)$ is an element of the test dataset

$$\mathcal{D}_{\mathrm{te}} = \{(\mathbf{f}_{M_{\mathrm{tr}}+1}, \mathbf{u}_{M_{\mathrm{tr}}+1}), ..., (\mathbf{f}_{M_{\mathrm{tr}}+M_{\mathrm{te}}}, \mathbf{u}_{M_{\mathrm{tr}}+M_{\mathrm{te}}})\}, \tag{5.19}$$



(a) (b) (c)

Fig. 5.13 Deformation of the 3D beam under point load (-1.75,1.31,-1.7)N on the second last node (a) Deformed mesh predicted using MAgNET (blue), for comparison FEM solution is presented (red) and undeformed mesh is represented by gray (b) L_2 error of nodal displacements between MAgNET and FEM solution. The relative error in predicting displacement of the node of application of load using MAgNET is 4.4% (c) L_2 error of nodal displacements between CNN U-Net and FEM solution. The relative error in predicting displacement of the node of application of load using CNN is 3.0%.

and $\mathcal{F}$ is the number of dofs of the mesh. The metric e_m gives us the notion of error between an expected finite element solution, $\mathbf{u}_m$, and the prediction of the neural network, $\mathcal{G}(\mathbf{f}_m)$. To analyze the overall quality of fitting, we define a single error metric over the entire test set as the average mean absolute error

$$\bar{e} = \frac{1}{M_{\text{te}}} \sum_{m=M_{\text{tr}}+1}^{M_{\text{tr}}+M_{\text{te}}} e_m, \tag{5.20}$$

with the corrected sample standard deviation (standard deviation of averaged errors) defined as

$$\sigma(e) = \sqrt{\frac{1}{M_{\text{te}}-1} \sum_{m=M_{\text{tr}}+1}^{M_{\text{tr}}+M_{\text{te}}} (e_m - \bar{e})^2}. \tag{5.21}$$

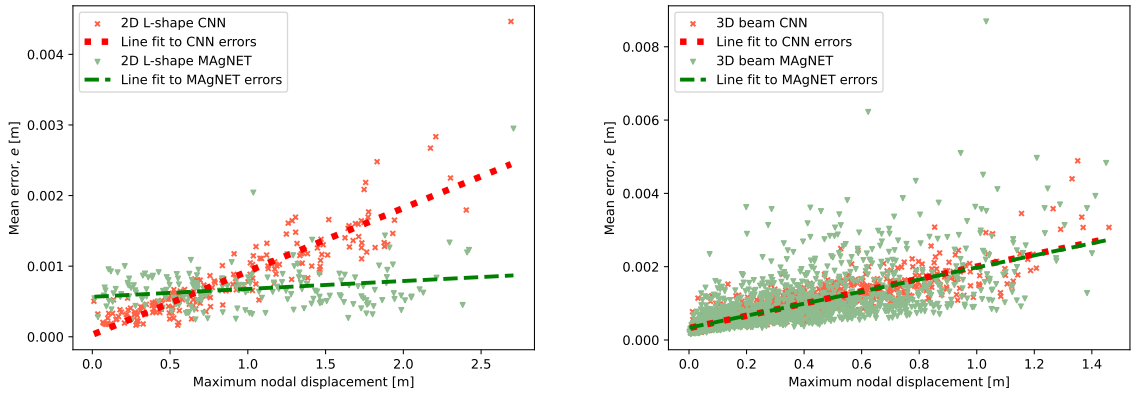
Finally, in addition to that, we also use the maximum error per degree of freedom over the entire test set

$$e_{\max} = \max_{m,i} |\mathcal{G}(\mathbf{f}_m)^i - \mathbf{u}_m^i|. \tag{5.22}$$

The aggregated error metrics for the entire test sets of structured mesh examples obtained using MAgNET and CNN approaches are summarised in Table 5.5. The first observation is that both MAgNET and CNN models exhibit similar prediction accuracy, demonstrating that the MAgNET architecture can achieve a comparable predictive capacity to the CNN

Example	M_{te}	$\bar{e}$ [m]	$\sigma(e)$ [m]	$e_{\max}$ [m]
2D L-shape (MAgNET)	200	0.5 E-3	0.2 E-3	1.1 E-2
2D L-shape (CNN U-Net)		0.7 E-3	0.6 E-3	1.8 E-2
3D beam (MAgNET)	1782	0.8 E-3	0.7 E-3	7.7 E-2
3D beam (CNN U-Net)		0.7 E-3	0.5 E-3	5.4 E-2

Table 5.5 Error metrics for the structured mesh examples. M_{te} stands for the number of test examples, and $\bar{e}$, $\sigma(e), e_{\max}$ are error metrics defined by Equations (7.19)-(7.20).



(a) (b)

Fig. 5.14 Mean absolute errors (see Equation (5.18)) as a function of maximum nodal displacements for all test examples for 2D L-shape and 3D beam cases for CNN U-NET and MAgNET nerworks.

U-Net architecture for a similar number of trainable parameters. The prediction errors, with respect to a characteristic length of 1m, fall below 0.1% for the average mean absolute error (Equation (7.19)), which is a promising result given the presence of geometric and constitutive nonlinearities. Additionally, we analyze the performance of the MAgNET model as a function of the maximum nodal displacement per test example. This dependency is visualized in Figure 5.14 for both benchmark examples. Although there is a general trend of increased errors for larger maximum displacement magnitudes, the sensitivity is low, and the errors remain small (the regression lines are $e(d) \propto 1.0 \cdot d \cdot 10^{-4}$ (2D L-shape) and $e(d) \propto 1.6 \cdot d \cdot 10^{-3}$ (3D beam) for MAgNET and $e(d) \propto 7.0 \cdot d \cdot 10^{-4}$ (2D L-shape) and $e(d) \propto 1.6 \cdot d \cdot 10^{-3}$ (3D beam)) for the CNN U-NET case.

As explained in Section 5.2.3, one can modulate the model capacity to capture non-linearities in the underlying data by modulating the number of channels in MAg and CNN layers. Importantly, the convolution windows are shareable in CNN architectures, whereas the aggregation windows in MAg architectures are independent. To this end, we expect CNN networks to require more channels than their respective MAgNET networks to achieve the same level of accuracy. We

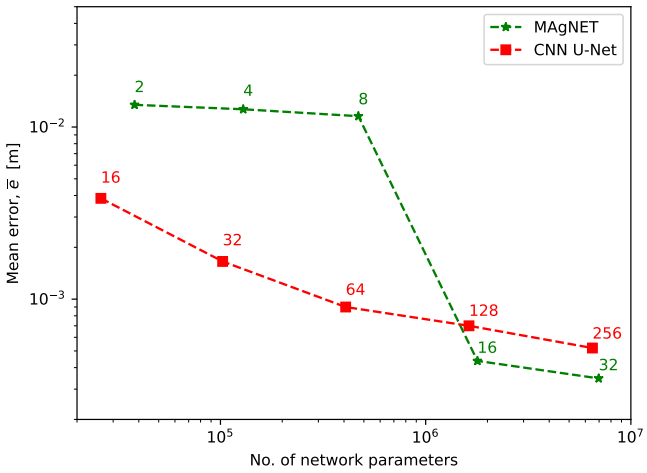


Fig. 5.15 Average mean error over the test set for the L-shape case as the number of network parameters is changed by altering the number of channels used for MAgNET and CNN U-Net architectures. The numbers in the plot represent the number of channels used at each level in the MAgNET and CNN U-Net networks.

used this fact when designing CNN and MAgNET architectures in Section 5.3.2. To verify this hypothesis and demonstrate this effect, we trained five MAgNET and five CNN U-Net networks on the L-shape dataset with different numbers of channels. In all analyzed cases, we used 4-level MAgNET and 3-level CNN U-Net architectures, with two MAg/Conv layers per level, and with a constant number of channels at all levels. Figure 5.15 shows that there is indeed a strong dependency of accuracy on the number of channels for both analyzed network architectures. For a comparable number of trainable parameters, CNN U-Nets can use more channels than their respective MAgNETs, providing them with comparable predictive accuracy. We can also observe that too few channels significantly reduce the fitting capabilities of both networks, with a step jump between the 8- and 16-channel case for MAgNET. For the two largest cases (16 and 32 channels for MAgNET and 128 and 256 channels for CNN U-Net), the accuracy of both architectures is comparable.

5.3.4 Predictions of MAgNET for general (unstructured) meshes

In Section 5.3.3, we demonstrated that the MAgNET architectures can achieve very good predictive capabilities for structured mesh cases, which was also cross-validated against respective CNN U-Net architectures. In this section, we aim to show that the high prediction accuracy of MAgNET can also be expected for unstructured mesh cases, which is the central point of the results section. We consider two cases: the first one is deformation under the application of

point loads (the 2D beam with hole case), similar to the case of structured examples, and the second is deformation under body forces (the 3D breast case).

Let us first analyze individual examples. In Figure 5.16 we present two particular loading cases of the 2D beam with hole. One of them is loaded at the tip, featuring the highest nodal displacement magnitude of all test cases, and the other one is loaded close to the hole, representing high local distortions. Similarly, for the three-dimensional problem, in Figure 5.17 we show the case featuring the highest nodal displacement magnitude of all test cases. In all mentioned examples we can observe overall good accuracy when visually comparing MAgNET predictions with the respective FEM solutions. This can also be checked quantitatively by analyzing maximum displacement errors. In the cases of the 2D beam with hole, those errors are 1.4% and below when related to the characteristic length of 1m. In the case of 3D breast geometry, such relative maximum error is higher, reaching almost 3.1% (related to the breast diameter of 0.16m). Despite this fact, we can observe that high local shape distortions are very well recovered. This property is more emphasized in Figure 5.17c where one can additionally observe that also the Dirichlet boundary conditions are very well predicted, even though they were only introduced implicitly by training data.

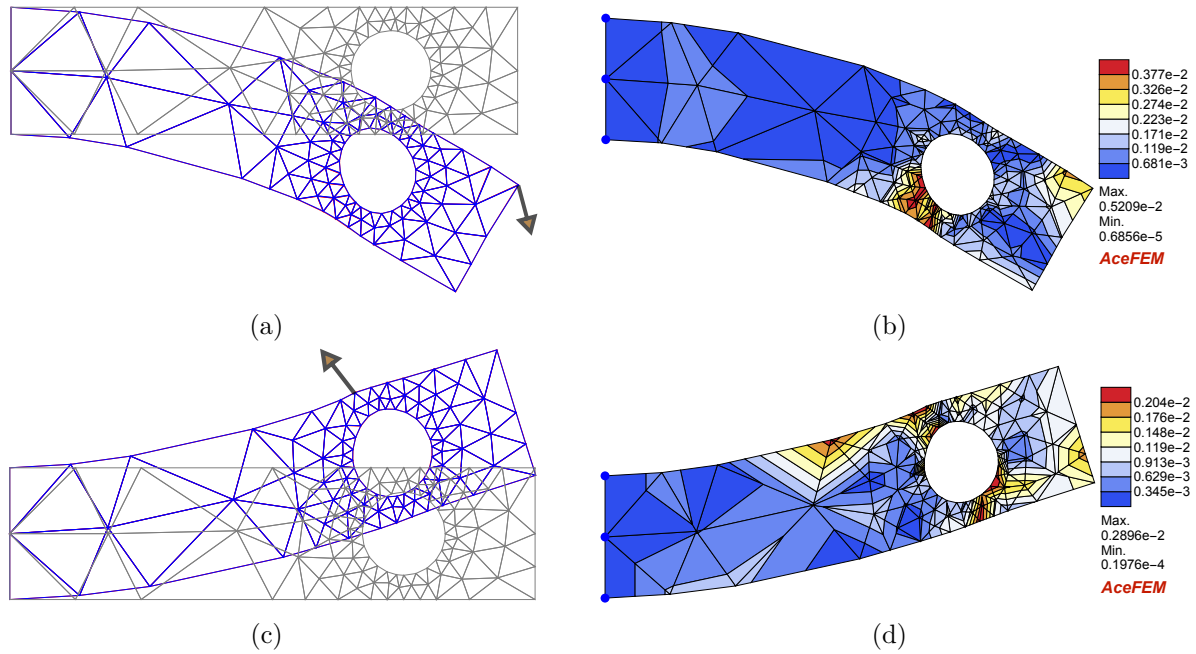
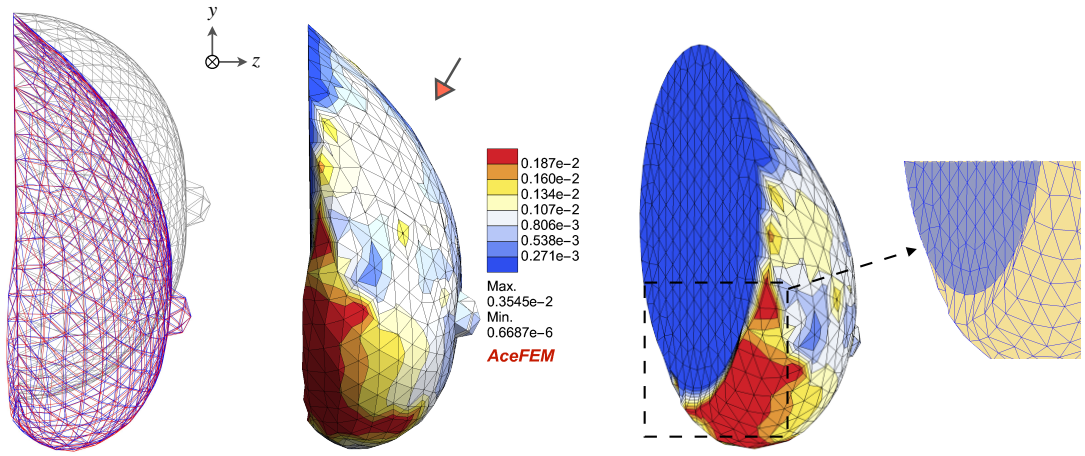


Fig. 5.16 Deformation of the 2D beam under two different point loads (upper case: $(1.28, -4.43)$N, lower case: $(-3.38, 4.04)$N). (a)&(c) Deformed meshes computed using MAgNET (blue) and FEM (red), with the undeformed configuration (gray). (b)&(d) L_2 error of nodal displacements between MAgNET and FEM solutions.

The aggregated error metrics for the entire test sets are provided in Table 5.6. The maximum displacement errors over all test cases, $e_{\max}$, are at the levels observed for particular cases in



(a) (b) (c)

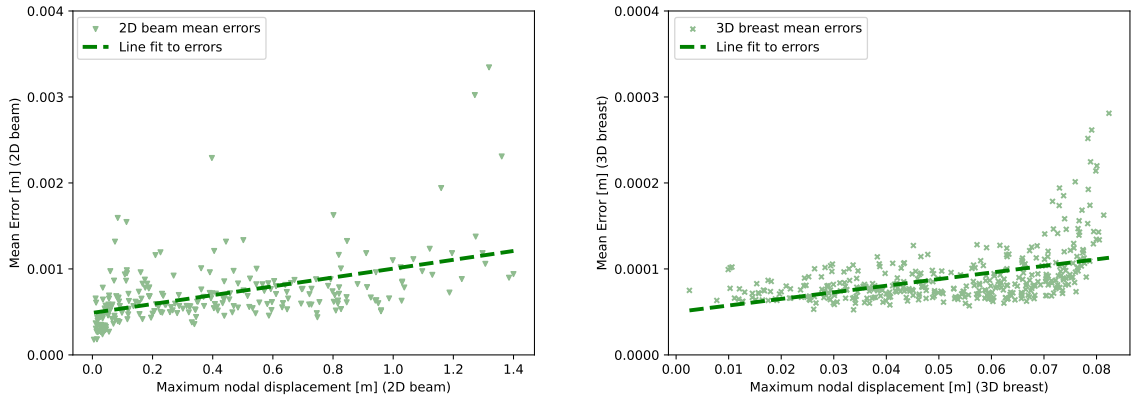
Fig. 5.17 Deformation of the 3D breast geometry with force density of $(-5.94, -5.23, -2.56)$ N/kg. (a) Deformed meshes computed using MAgNET (blue) and FEM (red), with the undeformed configuration (gray). (b) L_2 error of nodal displacements between MAgNET and FEM solutions. (c) Titled view of the figure(b), MAgNET efficiently captures fixed boundary and nearby high non-linear deformations by learning implicitly from the data.

Figures 5.16 and 5.17. At the same time, the average mean errors, $\bar{e}$, are at least an order of magnitude lower, which suggests that the errors close to maximum levels are not that often. The average mean errors are further analyzed in a case-by-case manner in Figure 5.18, which is analogous to the analysis done for the structured cases in Figure 5.14. Again, we plot the mean error e, of each test example as a function of the maximum nodal displacement. The regression lines $e(d) \propto 5.0 \cdot d \cdot 10^{-4}$ (2D-beam) and $e(d) \propto 8.0 \cdot d \cdot 10^{-4}$ (3D Breast) show low sensitivity of the MAgNET predictions to displacement magnitudes.

Example	M_{te}	$\bar{e}$ [m]	$\sigma(e)$ [m]	$e_{\max}$ [m]
2D beam (hole)	240	0.7 E-3	0.4 E-3	1.4 E-2
3D breast	400	8.9 E-5	3.1 E-5	5.1 E-3

Table 5.6 Error metrics for the unstructured mesh examples. M_{te} stands for the number of test examples, and $\bar{e}$, $\sigma(e)$, $e_{\max}$ are error metrics defined in Section 5.3.3.

Above, we have demonstrated a good prediction accuracy of MAgNET within the test dataset (which is located in the interpolated domain). However, it is well known that this accuracy can gradually deteriorate when moving to the extrapolated region, see, e.g., Deshpande et al. [2022]. We are going to study this effect for MAgNET for a particular case that is based on the 3D breast geometry. As described in the Table 5.3, during the training, the b_z component of body force density is varied from -3 to 3 N/kg only. At the inference time, we applied b_z



(a) (b)

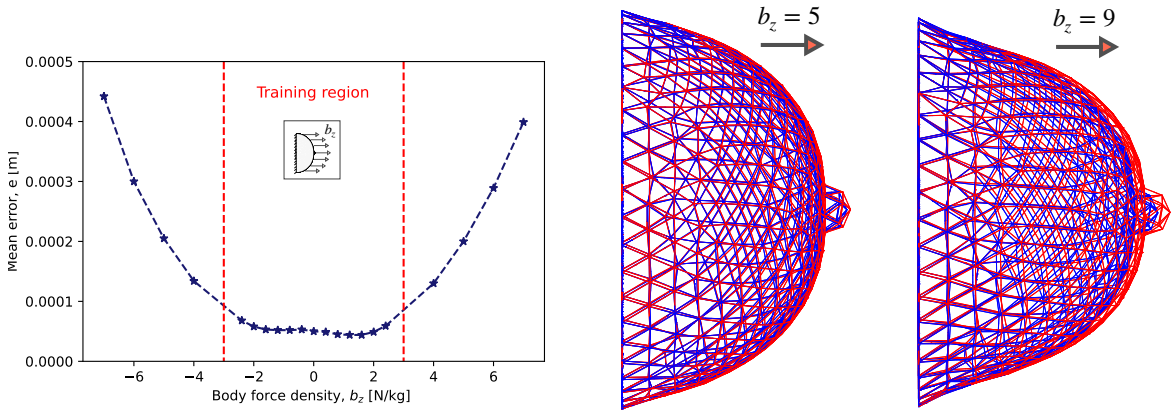
Fig. 5.18 Mean absolute errors (see Equation (5.18)) as a function of maximum nodal displacements for all test examples (with unstructured meshes) predicted using MAgNET for (a) 2D beam with hole (b) 3D breast case.

from -7 to 7 N/kg (keeping other components 0) to see how the predictions perform within and outside the training magnitudes. Figure 5.19 shows that the error is fairly low and is not increasing within the training region and it increases rapidly outside, which confirms this well-known effect. Figures 5.19b and 5.19c show deformed meshes predicted for b_z= 5 and b_z = 9 N/kg, respectively, both outside the training data region. MAgNET is observed to give visually acceptable results although the accuracy of the framework decreases as we move away from the training data.

A note on physics-informed errors

The proposed MAgNET framework has only been trained by minimizing the loss function for displacement errors, with no additional explicit information about the underlying physics/mechanics. As demonstrated earlier in this work, such training can provide very good accuracy in terms of predicted displacements. However, this accuracy is not of machine precision. To this end, a natural question arises: how far the displacement errors can violate physics? To answer that question, we are going to analyse some problem-based quantities of interest, such as residuals (balance of forces) or stresses, in comparison to the expected ground-truth results.

In Figure 5.20 we show nodal internal residual forces for the 2D beam with hole cases that we introduced earlier (compare Figure 5.16 for respective displacement errors). Ideally, the residual forces should be zero (the balance of forces), except for the boundary condition areas in which they should be exactly opposite to the reaction at the support and the applied external force. However, due to inaccuracies in displacements obtained from the MAgNET model, differences

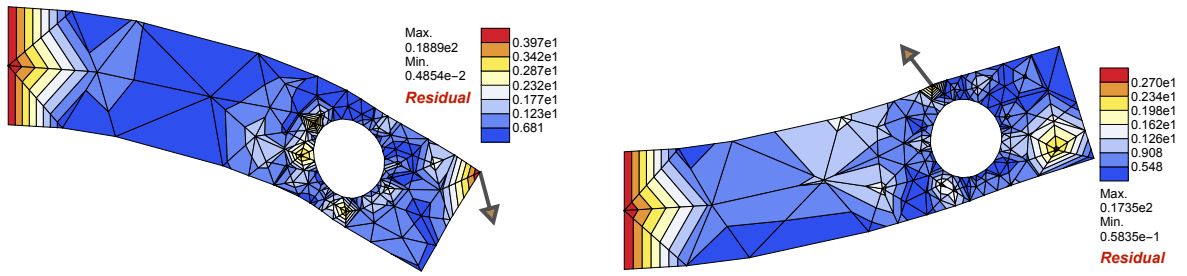


(a) (b) (c)

Fig. 5.19 3D Breast deformation under horizontal body force densities, $(0, 0, b_z)$ N/kg. (a) Mean absolute error for testing cases in interpolated and extrapolated regions. The error increases rapidly in the extrapolated region while it remains low in the training (interpolated) region. (b)&(c) Visualisation of deformed meshes for force densities outside the training region computed using MAgNET (blue) and FEM solution (red).

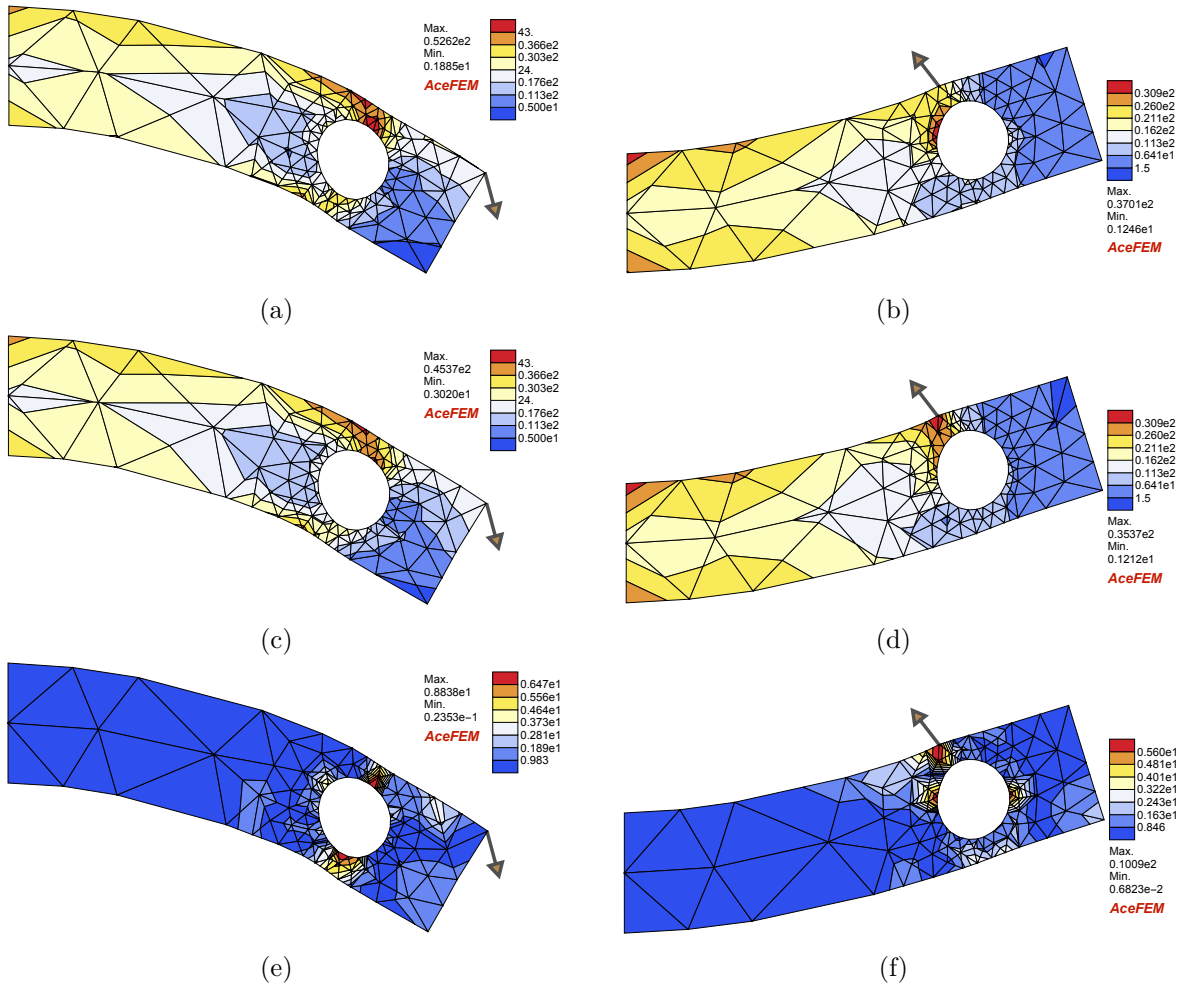
with respect to the ground true residuals can be noted. In Figure 5.20, we can observe the expected high residual forces in the areas where Dirichlet and Neumann boundary conditions are applied, however, also localised residual force spots are present in the fine mesh region around the hole. The magnitude of those errors in the localised spots can go up to 20% of the maximal magnitude of applied forces. Also, when more closely analysing the residuals at boundary condition areas, it turns out that they do not fully match the respective FEM residuals. For instance, the relative error in residuals at the support in Figure 5.20a is almost 5%. When analysing the entire test set, we observed that the mean error in retrieving residuals at the Dirchilet boundary related to the maximum external force is 2.4%. A similar relative mean error value for retrieving the Neumann boundary residual is 14.6%. The higher error for Neumann boundary residual is attributed to high local non-linear deformations at the vicinity of the point of application of force. Though the displacement errors provoked by these non-linearities are not so high, they can result in high residual errors through the relatively high magnitude of element stiffness.

The errors observed in Figures 5.16 and 5.20 can have a direct impact on some application-dependent quantities of interest. As an example, in Figure 5.21, we present the field of von Mises stresses, which is a commonly used measure of shear stresses. We can observe that the MAgNET solution provides similar profiles of stresses as compared to respective FEM solutions, however, high localised errors are present at the fine mesh region (up to 30% of the reference FEM maximal von Mises stresses).



(a) (b)

Fig. 5.20 Nodal residual forces obtained using MAgNET solutions for the examples in Figure 5.16 (plotted on deformed meshes). The relative error for retrieving the total reaction force at the fixed interface is (a) 4.7% for the first example (b) 0.1% for the second example.



(a) (b)

(c) (d)

(e) (f)

Fig. 5.21 Von Mises stresses obtained for the two examples as in Figure 5.16 using (a)&(b) MAgNET solution (c)&(d) FEM solution. In (e)&(f) the absolute error between the MAgNET and FEM von Mises stresses is shown.

The above mentioned localised errors in residual forces, von Mises stresses and other relevant quantities of interest can be reduced by enriching the loss function with physics-informed terms. For instance, in the context of mesh-based force-displacement data, in [Odot et al., 2021] such enrichment has been introduced by scaling individual components of the loss function with the respective computed residual values, which proved to reduce residual errors. In [As'ad et al., 2022], the authors introduced an energy-based approach that provided purely physics-informed training for the Gauss-point stress-strain relationship, which allowed them to satisfy the expected frame indifference. Similar concepts of physics-informed loss functions can be seamlessly integrated into the MAgNET framework, which would convert it into a Physics Informed MAgNET.

5.4 Conclusion

In this work we proposed MAgNET, a novel framework for efficient supervised learning on graph-structured data using geometric deep learning. The framework comprises two neural network operations: MAg and graph pooling/unpooling layers, which together form a graph U-Net architecture capable of learning on large-dimension inputs/outputs. Notably, the MAgNET framework is not restricted to any particular input→output relationship or any specific mesh- or discretization scheme, making it superior to existing convolutional neural network architectures. MAgNET allows for arbitrary non-grid inputs/outputs, meaning it can handle arbitrary meshes and support complex geometries and local mesh refinements, making it suitable for a wide range of engineering applications.

We demonstrated and studied the capabilities of MAgNET in capturing nonlinear relationships in data. For this purpose, we conducted quantitative cross-validation of predictions made by MAgNET and the well-known convolutional U-Net architecture, both of which have been verified against the ground-truth results obtained with FEM. The benchmarks have proved that MAgNET has similar predictive capabilities as CNN U-Net for structured meshes, and it can also be extended to arbitrary meshes while preserving similar accuracy of predictions.

There are several natural directions for extending the capabilities of MAgNET. Firstly, the inclusion of underlying physics into the training process would enhance the overall performance of the framework. Although we have numerically demonstrated that the current data-based MAgNET framework can capture the underlying physics of the problem, we have also identified areas of increased errors, especially near regions of refined mesh. Integrating quantities of interest with the learning objective function would improve performance, and the implementation of a Physics Informed MAgNET is a very natural extension of the framework in the immediate future. Secondly, extending MAgNET to path-dependent processes, such as elasto-plasticity, is

a promising direction, which would require keeping track of the evolution of state variables in a time-stepping manner. Recurrent neural network architecture, as described in [Mozaffar et al., 2019] or recent developments from our group [Vijayaraghavan et al., 2021b], may be utilized for this purpose. Finally, there is a direct possibility to extend the proposed MAg layer to a Bayesian version, which would convert MAgNET into a probabilistic version. This could be achieved by performing local aggregations with probability distributions instead of discrete weights, similar to what we did in the case of CNNs in our previous work [Deshpande et al., 2022]. A Bayesian MAgNET would be capable of tracking uncertainties that are inherent to the choice of network architecture, as well as those inherent to real-world data.

We have made all the codes, datasets, and examples presented in this paper available open-access in the MAgNET repository at https://github.com/saurabhdeshpande93/MAgNET. Given the generality of MAgNET in supporting arbitrary non-linear relationships and arbitrary discretizations, we believe that the repository will provide a useful surrogate modeling framework for researchers and practitioners in various application areas across disciplines. We see it not only as a ready-to-use machine-learning library but also as a reference point and foundation for future developments and extensions in this emerging direction of research. The generality of MAgNET will enable the community to explore a range of new applications and modeling scenarios. By sharing our work, we hope to foster collaboration and advance the state-of-the-art in deep-learning surrogate modeling.

Chapter 6

Convolution, aggregation and attention based deep neural networks for accelerating simulations in mechanics

Abstract

Deep learning surrogate models are being increasingly used in accelerating scientific simulations as a replacement for costly conventional numerical techniques. However, their use remains a significant challenge when dealing with real-world complex examples. In this work, we demonstrate three types of neural network architectures for efficient learning of highly non-linear deformations of solid bodies. The first two architectures are based on the recently proposed CNN U-NET and MAgNET (graph U-NET) frameworks which have shown promising performance for learning on mesh-based data. The third architecture is Perceiver IO, a very recent architecture that belongs to the family of attention-based neural networks–a class that has revolutionised diverse engineering fields and is still unexplored in computational mechanics. We study and compare the performance of all three networks on two benchmark examples, and show their capabilities to accurately predict the non-linear mechanical responses of soft bodies.

This chapter is reproduced from: S. Deshpande, R.I. Sosa, S.P.A. Bordas, J. Lengiewicz, *Convolution, aggregation and attention based deep neural networks for accelerating simulations in mechanics,* Frontiers in Materials, 2023, Volume 10, 1128954, https://doi.org/10.3389/fmats.2023.1128954

6.1 Introduction

The ability to make fast or real-time predictions of the response of physical systems is essential for a variety of engineering applications. Notable examples of this can be found in the field of robotics [Choi et al., 2021; Rus and Tolley, 2015] and medical simulations [Bui et al., 2018; Courtecuisse et al., 2014b; Mazier et al., 2021; Mendizabal et al., 2019b], which have the potential to advance personalized medicine and improve computer-assisted and robotic surgery [Chen et al., 2020a; Dennler et al., 2021]. In computational physics and chemistry, fast and accurate predictions are fundamental for studying complex systems, such as those arising in biology and materials science [Friesner, 2005], or in drug discovery [De Vivo et al., 2016]. In many cases, the necessary accuracy of these predictions requires complex models that can be expressed through partial differential equations and solved numerically using methods such as the finite element method (FEM) at continuum scales or specialized *ab initio* approaches at the atomic or quantum scales. However, these high-fidelity computational models are often too slow for real-time or practical purposes, and therefore approximate or surrogate models must be developed to achieve the necessary speed-ups.

At the same time, the 21st century has seen an explosion of measurement data, much of which is available as public datasets in various scientific domains, including structural mechanics, material science, and meteorology [Elouneg et al., 2022; Gholamalizadeh et al., 2022; Zakutayev et al., 2018]. The availability of this data, combined with the rapid growth in computational resources, has led to the increasing importance of machine learning (ML) techniques [Bock et al., 2019; Butler et al., 2018; Schleder et al., 2019] for solving forward and inverse engineering problems. This includes surrogate and data-driven approaches that aim to enable modeling [Barrios and Romero, 2019], accelerate computationally costly direct numerical simulations [Capuano and Rimoli, 2019; Rupp et al., 2012; Weerasuriya et al., 2021; Wirtz et al., 2015], and even discover new material laws [Flaschel et al., 2021; Liu et al., 2017]. The increasing use of ML in engineering and other fields has also spurred the development of various methods and algorithms for improving the accuracy and efficiency of these techniques.

Within the class of machine learning methods for surrogate and data-driven modeling, deep learning (DL) approaches have seen great success due to their ability to efficiently extract complex relationships present in the underlying data. DL models have been successfully employed for a range of tasks in diverse fields such as computational physics&chemistry, material science, computational mechanics, computer vision, natural language processing, and many others [Brown et al., 2020; Choudhary et al., 2022; Jha et al., 2018; Oishi and Yagawa, 2017; Schmidt et al., 2019; Schütt et al., 2017; Voulodimos et al., 2018]. In computational chemistry, machine learning force fields (MLFFs), see [Unke et al., 2021], have seen great success in recent years for accelerating costly *ab initio* simulations. For instance, Deep Tensor Neural

Network (DTNN) [Schütt et al., 2017] and SchNet [Schütt et al., 2017] models have been shown to accurately predict forces in a variety of molecules and could be used in applications such as protein folding and material design. Similarly, computational mechanics has witnessed an increasing use of DL surrogate models as a replacement for costly direct numerical simulations [Abueidda et al., 2021; Mianroodi et al., 2021]. What is common to all the above-mentioned cases is that deep learning techniques rely on deep artificial neural networks (deep ANNs, or DNNs), which must be trained on a sufficiently large amount of data. While this training process is computationally costly, once trained, the predictions of DL models are extremely efficient.

Obtaining necessary amount of training data is often difficult when it originates from physical experiments. This can be due to multiple factors, such as high costs, risks & difficulties associated with the experiments, or data privacy clauses. There are two possible approaches to deal with the scarcity of experimental data. The first approach relies on enhancing the DL model with the information on underlying physics – an approach popularly termed as Physics Informed Neural Networks (PINN) [Mao et al., 2020; McFall and Mahan, 2009; Odot et al., 2021; Samaniego et al., 2020]. The second approach includes the underlying physics implicitly, through high-fidelity simulations done *in silico* to provide the necessary amount of synthetically generated data, which has shown to be useful in various applications [Aydin et al., 2019; Kim et al., 2022; Le et al., 2017; Pfeiffer et al., 2019; Vijayaraghavan et al., 2021a]. In this work, we will follow the latter approach and will focus on DL surrogate models that are trained on synthetically generated data from finite element simulations in non-linear elasticity.

One of the most important aspects that will be studied in this work is the architecture of deep neural networks. The majority of DL approaches that are present in the literature are based on fully connected networks, which can be inefficient and prone to overfitting when applied to high-dimensional inputs. If such large inputs are structured, they fall under the umbrella of geometric deep learning (GDL) [Bronstein et al., 2021], a concept that has gained increasing interest in recent years. In this work, we will compare three architectures that can efficiently handle high-dimensional structured inputs: convolutional neural networks (CNNs), graph neural networks (GNNs), and attention-based networks.

Convolutional neural networks (CNNs) are known to outperform traditional fully-connected ANNs, and this has been demonstrated in various domains, including physics-based simulations [Deshpande et al., 2022; El Haber et al., 2022; Guo et al., 2016; Krokos et al., 2022a]. CNNs work on the principle of parameter sharing and local convolution operations, which enables efficient training on large inputs. Their disadvantage is that the inputs/outputs of CNNs are restricted to grid inputs, such as images, videos, or structured FE meshes. However, CNNs have found their successors, the graph neural networks (GNNs), that can work with any structure of inputs/outputs.

Graph-based approaches leverage the topological information of the input to perform local operations in the respective neighborhood only, and can learn efficiently on generally structured data. Recently, GDL methods have shown promising performance for their applications as well in the field of mechanics, [Battaglia et al., 2018; Krokos et al., 2022b; Pfaff et al., 2021; Strönisch et al., 2022; Vlassis et al., 2020]. More recently, [Deshpande et al., 2023b] proposed MAgNET, a novel graph U-Net framework for efficiently learning on mesh-based data. In this work we utilise it to accurately predict non-linear deformations of solids.

Attention-based approaches, similar to human cognitive attention, work by allowing the DL model to focus on certain parts of the input data that are relevant to the task at hand. This is done through a fully trainable process that, without the need to introduce topological information or enforce structural restrictions, allows the neural network to extract dependencies from throughout the whole input domain. This type of approach has led to significant strides in a wide range of areas, starting from computer vision [Xu et al., 2015] to natural language processing [Baevski et al., 2020; Devlin et al., 2018], as well as becoming the basic building block of the Transformer architecture [Vaswani et al., 2017]. Recently the Perceiver IO [Jaegle et al., 2022], a new type of architecture that builds upon Transformers, has been proposed as a general-purpose model that can handle data from arbitrary settings. Since Perceiver IO has been shown to achieve several state-of-the-art results without the need for problem-specific architecture engineering, we will compare its performance on non-linear deformation prediction of solids based on mesh data against the previously discussed models.

To summarise, in this work we will compare three DNN architectures: two architectures presented in our earlier works, i.e., CNN U-Net framework [Deshpande et al., 2022], and MAgNET framework [Deshpande et al., 2023b], as well as the attention-based architecture, Perceiver IO [Jaegle et al., 2022], which has not been explored for its applications in mechanics yet. We show the capabilities of three frameworks by learning on non-linear FEM datasets and by cross-comparing their performance. In Section 6.2, we will introduce the three DNN architectures, in Section 6.3, we will study their performance, and in Section 6.4 we will summarize the results and discuss future directions.

6.2 Method

As previously mentioned in the introduction, in this paper we propose three types of deep neural network (DNN) frameworks that can be used as surrogate models to replace computationally expensive non-linear FEM solvers. The proposed DNN frameworks are trained on force-displacement FEM datasets that are given in the mesh format. Once trained, these surrogate

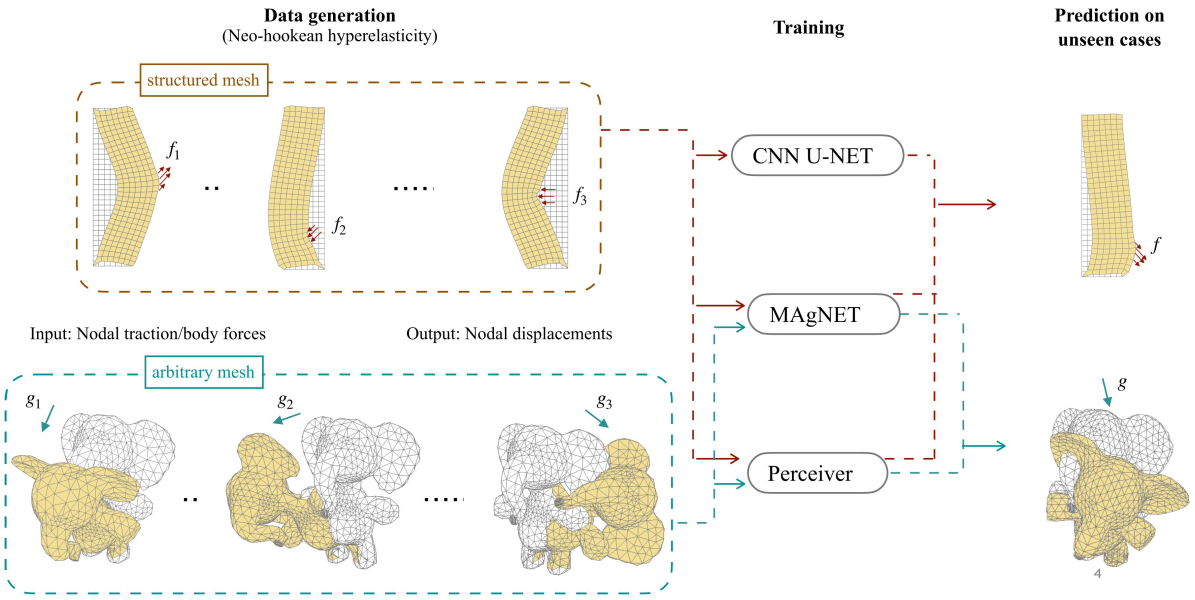


Fig. 6.1 Outline of the neural network surrogate frameworks for predicting body deformations. (Left) Training datasets for structured and arbitrary mesh cases are generated by using a non-linear FEM solver. (Middle) Proposed neural network frameworks are trained on these datasets. For structured mesh case, all NN frameworks are used while for arbitrary unstructured meshes only MAgNET and Perceiver IO networks are used. (Right) Trained networks are then used as surrogate models to predict the deformation of bodies under unseen forces.

DNN models are able to quickly and accurately simulate the mechanical responses of bodies subjected to external forces. The outline of study pursued in this paper is shown in Figure 6.1.

Any input mesh can be categorised into either a structured mesh or an arbitrary unstructured mesh. In this work we introduce three types of DNN network architectures. The CNN U-Net network can only be (straightforwardly) used for structured meshes, while the MAgNET and Perceiver IO networks are more general and are capable of handling arbitrary mesh inputs. All these frameworks are discussed in detail in the following subsections.

6.2.1 DNN frameworks for predicting mechanical deformations

Below we introduce three different types of neural network architectures which can efficiently predict non-linear deformations of bodies subjected to external traction and body forces. All the proposed DNN frameworks directly operate on the finite element mesh data thereby making it very convenient to be used as surrogate models in place of conventional FEM solver. The first two i.e. CNN U-Net and MAgNET belong to the family of U-Net architecture while Perceiver IO is based on Transformer-type attention, see Table 6.1.

Framework	Type	Supported mesh
CNN	U-Net (convolution operation)	structured
MAgNET	Graph U-NET (MAg operation)	arbitrary
Perceiver IO	Transformer (attention mechanism)	arbitrary

Table 6.1 Properties of deep neural network architectures studied in this work.

CNN U-Net

CNNs were originally proposed for performing classification and regression tasks on image, video like data but lately are even being used for generic inputs such as mesh data which is crucial to many scientific applications. In particular, U-Net like architectures have shown great potential in learning on large-scale inputs and lately have been successfully used for simulating mechanical responses of materials as well [Deshpande et al., 2022; Mianroodi et al., 2021]. The name U-Net comes from the particular U-shaped architecture which involves a series of convolutional and pooling operations. Convolutional layers are responsible for non-linear transformations whereas pooling enables learning through low-fidelity representation thus making the network capable of learning on high-dimensional inputs. Experiments presented in this work are carried out by using the CNN U-Net framework proposed by [Deshpande et al., 2022], see Figure 6.2.

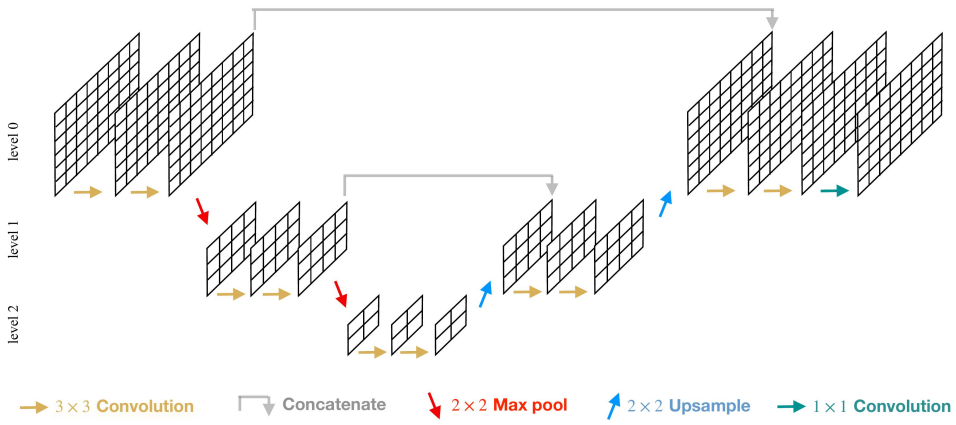


Fig. 6.2 Schematic of CNN architecture used for generic structured 2D mesh inputs.

One major limitation of the CNN is that it cannot straightforwardly accommodate unstructured mesh inputs. To overcome this issue, the simplest approach embeds a structured grid on unstructured meshes with a naive mapping between unstructured and grid node values [Mendizabal et al., 2019a]. While more sophisticated approaches are proposed to make unstructured meshes compatible to be used with CNN framework [Brunet et al., 2019]. However, they perform

poorly on complicated geometries and come with an associated preprocessing cost; they are not considered in the scope of this work.

MAgNET

In an attempt to generalise CNN to arbitrary unstructured meshes, very recently [Deshpande et al., 2023b] proposed the MAgNET framework, see Figure 6.3. MAgNET architecture belongs to the family of graph U-Net architectures and it is proposed for efficient learning on mesh structured data. MAgNET directly accepts arbitrary mesh inputs (such as forces/stresses/displacements of nodes in the mesh) values thus making it very convenient to be used with existing numerical solvers.

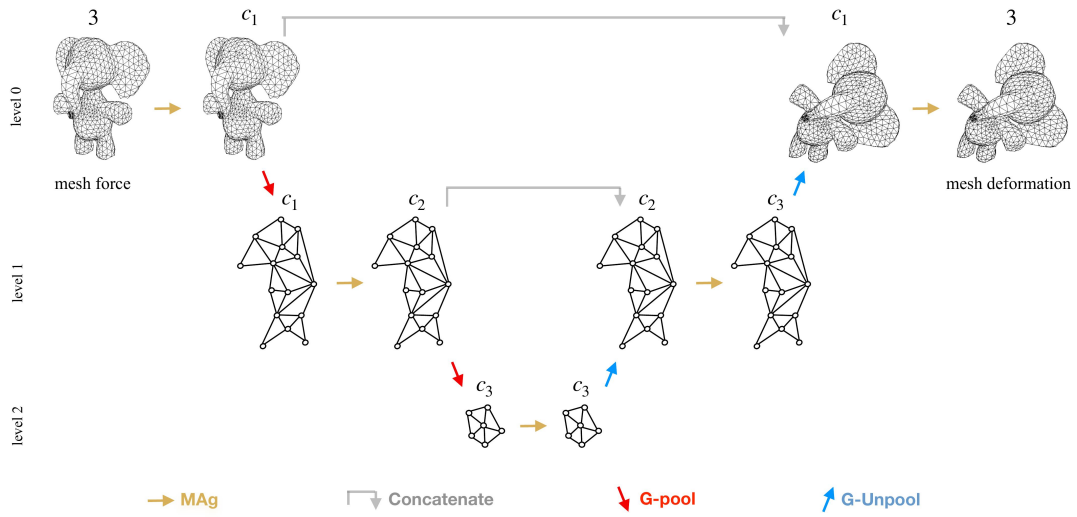


Fig. 6.3 Schematic of the MAgNET architecture used in this work. It takes external forces on arbitrary mesh as an input to gives mesh displacements as output.

MAgNET relies on the so-called MAg layer (Multichannel Aggregation layer) which is capable of learning nonlinear transformations between input and output data existing in the mesh format. MAg extends the concept of local operations in convolution layers to arbitrary mesh inputs by performing aggregation of nodal feature values in the respective neighborhood nodes only. It leverages the topology of inputs and performs learnable local aggregations with heterogeneous window sizes as opposed to the fixed-size window in the case of CNN. While its graph pooling/unpooling layers enable efficient learning on large-dimension inputs through reduced graph representation.

Perceiver IO

The Perceiver IO architecture, [Jaegle et al., 2022], was developed with the goal of achieving a DL scheme that can easily integrate and transform arbitrary information for arbitrary tasks. This architecture employs an attention encoder that maps inputs from a wide range of modalities to a fixed-size latent space using cross-attention, this latent space is then further processed using self-attention as an usual Transformer and decoded into the output domain via cross-attention, see Figure 6.4. This process allows the network to scale to large and multi-modal data since it decouples the bulk of the network's processing from the size and modality-specific details of the input. In this work, we will leverage this property and use Perceiver IO to learn non-linear deformations on unstructured meshes without adding any information or restrictions about how to treat the underlying data structure. During training, Perceiver IO automatically learns the important dependencies that exist in the input domain, composed of arbitrarily unstructured mesh data, and transforms them into the corresponding output which consists of displacement data.

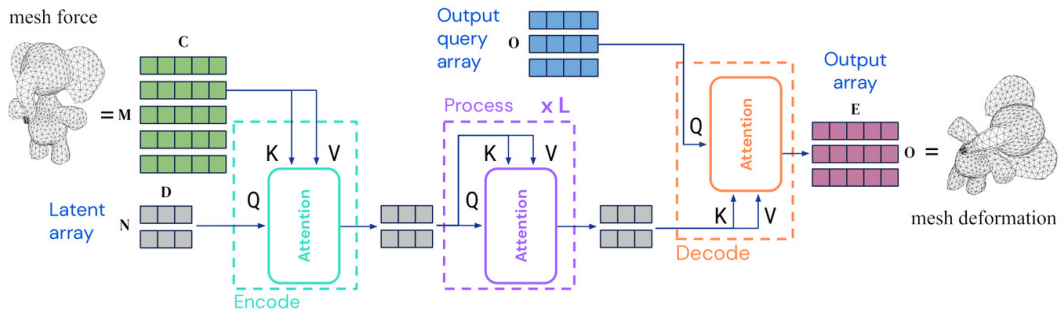


Fig. 6.4 Schematic of the Perceiver IO architecture, [Jaegle et al., 2022], used for external forces on arbitrary mesh as inputs and mesh displacement as outputs.

6.2.2 Input/output & training of DNN surrogate models

As motivated in Section 6.2, the proposed DNN frameworks are trained on the force-displacement datasets. Let us denote the neural network in consideration as h, it is parameterised by trainable parameters, $\boldsymbol{\theta}$. In all the cases, h accepts external forces, $\mathbf{f}$, on all degrees of freedom (dofs) of mesh as the input. And as an output it predicts displacement vector, $\mathbf{u}$, of all dofs (same size as the input), i.e., $h : \boldsymbol{f} \rightarrow \boldsymbol{u}$.

In the case of CNN and MagNET, forces associated with X, Y, Z degrees of freedom are kept in different channels. For instance, in the case of CNN U-Net, X, Y directional forces on a 2D quad mesh with $n_x \times n_y$ nodes are fed to the network as a $2 \times n_x \times n_y$ tensor. For MAgNET, they

are provided as a single dimensional tensor of shape $2 \cdot n_x \cdot n_y$, with X and Y directional forces concatenated together (each representing a different channel). On the other hand, Perceiver IO inputs are first kept as a single array of size $2 \cdot n_x \cdot n_y$ to which we apply 1×1 convolution kernels to project it to a tensor of shape $2 \cdot n_x \cdot n_y \times 256$, adding 256 channels. Finally, to this channel dimension we concatenate trainable 1D positional embeddings, thus leaving us with a tensor of shape $2 \cdot n_x \cdot n_y \times 512$ that we will use as an input for the network.

Now, for a given training dataset $\{(\mathbf{f}_1, \mathbf{u}_1), ..., (\mathbf{f}_N, \mathbf{u}_N)\}$, h is trained by minimizing mean squared error between true and predicted values as to get optimised parameters $\boldsymbol{\theta}^*$ as:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \|h(\mathbf{f}_i, \boldsymbol{\theta}) - \mathbf{u}_i\|_2^2 \tag{6.1}$$

Performance of h (i.e. the neural network in consideration) over respective test dataset $\{(\mathbf{f}_1, \mathbf{u}_1), ..., (\mathbf{f}_M, \mathbf{u}_M)\}$ is measured in terms of mean absolute error which is computed for each example (e_m) and for the entire test set ($\bar{e}$) as follows:

$$e_m = \frac{1}{\mathcal{F}} \sum_{i=1}^{\mathcal{F}} |h(\mathbf{f}_m)^i - \mathbf{u}_m^i|, \qquad \bar{e} = \frac{1}{M} \sum_{m=1}^{M} e_m, \qquad \sigma(e) = \sqrt{\frac{1}{M-1} \sum_{m=1}^{M} (e_m - \bar{e})^2}, \tag{6.2}$$

where $\mathcal{F}$ stands for the number of dofs of the mesh. The maximum error over the entire test dataset is defined as

$$e_{\max} = \max_{m,i} |h(\mathbf{f}_m)^i - \mathbf{u}_m^i|. \tag{6.3}$$

6.3 Results

We validate proposed neural network frameworks on two examples, representing a 2D and a 3D problem respectively. For the 2D example, a structured mesh is considered so that all frameworks including CNN can be applied to it. Whereas for the 3D example an arbitrary unstructured mesh is considered.

6.3.1 Generation of hyperelastic FEM training data

As motivated in the methodology section, training datasets of non-linear displacement solutions are generated by applying random traction and body forces on the given discretisation. The number of cases generated randomly (dataset size) has been chosen large enough to generalise well to unseen arbitrary forces. The dataset is split into the training (95%) and testing

(5%) part, and the pairs of input force and output displacement solutions from the training dataset are then fed to train different types of neural networks. The proposed neural network frameworks are validated on two examples, both following Neo-Hookean hyperelastic law. To avoid the divergence of the non-linear FEM solver, both traction and body forces are applied in incremental load steps. All the computations are performed using AceFEM framework [Korelc, 2002].

For the 2D case, a rectangular domain made of soft material and discretized by 8×32 mesh with 217 quad elements is considered. It is constrained at 4 corner nodes as shown in Figure 6.5a. It is subjected to traction forces of random magnitude, location, and direction in the region prescribed by the pink line. Body forces are ignored in this case. Table 7.1 provides detailed information about datasets including the material properties and external force ranges used for the generation of 2D and 3D datasets. For the 2D case, a lower range of Y-direction force density is chosen since it doesn't contribute much to generating large deformation solutions.

In the 3D case, a continuum toy elephant model is discretised with 6627 tetrahedron elements. It is subjected to fixed boundary conditions by constraining nodes on the bottom region of the legs. Body forces in random magnitude and direction are applied in the transverse directions (see Figure 6.5b) to generate datasets of force-displacement pairs. External tractions are ignored in this case.

Case	Material properties (E [Pa], ν)	N.of FEM DOFs ($\mathcal{F}$)	External traction/body force density range	Dataset size (train + test)
2D domain	100, 0.3	512	f_x = -24 to 24 N/m, f_y = -8 to 8 N/m	$7124 + 372$
3D elephant	3×10^6, 0.4	5835	b_x, b_z = -0.35 to 0.35 N/kg, b_y=0 N/kg	$7600 + 400$

Table 6.2 Desciption of FEM datasets

6.3.2 Implementation details

As introduced in the methodology section, CNN and MAgNET frameworks belong to the family of U-Net architectures, while Perceiver IO leverages Transformer-style attention. It has to be noted that all the frameworks are robust, and do not need fine hyperparameter tuning.

2D case: For the CNN U-Net, a 4-level architecture with 3 max-pooling/upsampling operations is used. At each level, two convolution layers with 3×3 filters are applied with 64, 128, 256, and 256 channels at respective levels. In the case of MAgNET, a 5-level graph U-Net architecture with 4 graph pooling/unpooling operations is used. At each level 2 MAg layers (with A^2

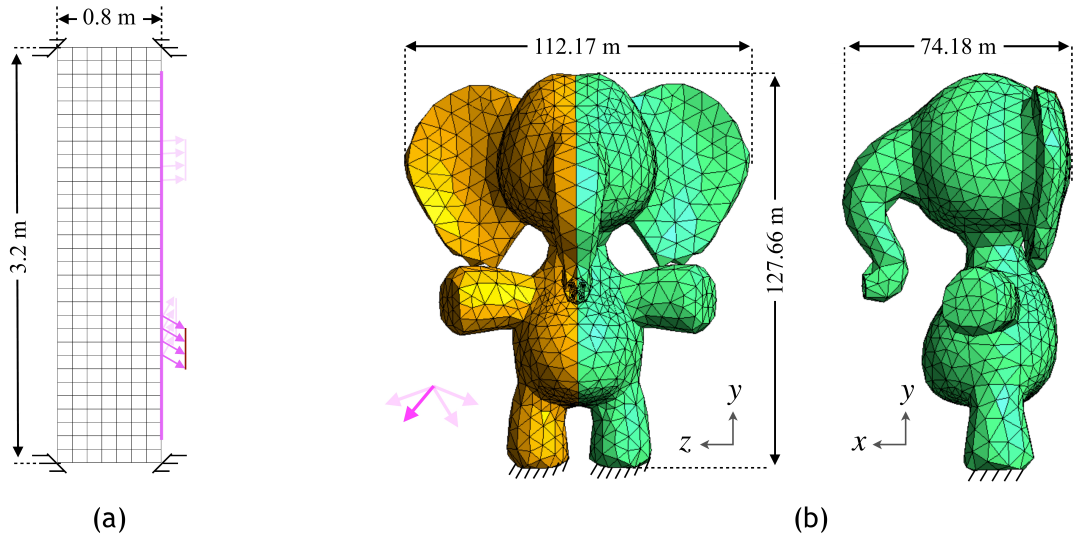


Fig. 6.5 Schematics of dataset generation for **(a)** 2D example subjected to external traction forces. **(b)** 3D example subjected to external body forces. External tractions and body forces are indicated with pink arrows.

adjacency, refer [Deshpande et al., 2023b]) are applied with 8, 16, 16, 32, and 32 channels at respective levels. For both 2D and 3D case MAgNET architectures, the seed of the first graph pooling operation is chosen by grid search, while seeds for other pooling layers are kept constant. This is done to have the maximum possible coarsened representation of the lowest-level graph. This ensures propagating boundary condition information with a minimum number of MAg operations at the lowest level. CNN U-Net is trained with a batch size of 16 for 32,000 epochs and MAgNET is trained with a batch size of 4 for 10,000 epochs.

In the case of Perceiver IO we defined 512 inputs (standing for dofs of the example) with a total embedding size of 512 following the procedure detailed in Section 6.2.2. We also selected a total of 128 latent arrays of dimension 210 for performing cross-attention in the encoder, self attention in latent space and inputs for the decoder. For the decoder's output query array we used an index dimension of 512, which defines the size of the outputs, and a channel dimension of 210 equal to the dimension size of the latents. We used a total of 3 blocks for the latent array processing, with 2 self-attention layers per block and 2 self-attention heads per layer. Both the encoder and decoder worked with 2 cross-attention heads each. The selection of these hyper-parameters was determined in a coarse exploratory fashion, with the goal of reducing the number of network parameters while maintaining its performance. Perceiver IO is trained with a batch size of 16 for 2,64,140 epochs.

3D case: For the MAgNET, 7-level graph U-Net architecture is used with 6 graph pooling/unpooling operations. Again at each level, 2 MAg layers (with A^2 adjacency) are applied

with 6, 6, 6, 12, 12, 24, 24 channels at respective levels. The complex topology of this particular mesh demands more number graph pooling layers, this ensures propagating boundary condition information with a minimum number of MAg operations at the lowest level. On the other hand, the only change with respect to the 2D case for Perceiver IO is an increase of the input and output dimension from 512 to 5835 in both the encoder and decoder. MAgNET architecture for this case is trained for 1200 epochs with a batch size of 4, whereas Perceiver IO is trained for 32,580 epochs with a batch size of 16.

CNN U-Net and MAgNET networks are trained using the Adam optimizer [Kingma and Ba, 2014], whereas Perceiver IO is trained using AdamW optimizer [Loshchilov and Hutter, 2017] as implemented in the original paper. CNN and MAgNET are implemented using TensorFlow [Abadi et al., 2015], while Perceiver IO is implemented using PyTorch [Paszke et al., 2019]. All the implementations in this work are performed using HPC facilities of the University of Luxembourg [Varrette et al., 2014].

6.3.3 Performance on unseen examples

Proposed DNN frameworks are trained and tested on the datasets generated as illustrated in Section 6.3.1. The maximum nodal displacement for the 2D case is 0.35 m and for the 3D case it is 140.04 m, i.e. we compute displacements of all the nodes for every single example, and then choose particular examples for which the maximum nodal displacement is observed. Table 6.3 summarises the performance of neural networks on the two test datasets. It shows that all three networks are capable of predicting mechanical deformation responses with a very low error.

To compare, we observed that CNN U-NET and Perceiver IO gave lower error metrics than MAgNET for the 2D case, which has relatively low dimensional input. As the size and complexity of the mesh increased in the 3D case, both MAgNET and Perceiver IO performed well, with Perceiver IO giving slightly better error metrics.

6.3.4 Training and inference of DNN frameworks

First, we compare the training convergence of proposed DNN frameworks by comparing the mean square loss plots for both 2D and 3D cases. Figure 6.6a shows that for the relatively smaller dimension inputs as in the 2D case, both CNN and Perceiver IO observed to learn more efficiently when compared to MAgNET. Figure 6.6b shows that both MAgNET and perceiver are able to learn efficiently on the complex mesh data as observed in the 3D case. However, MAgNET could learn more quickly than Perceiver IO, also MAgNET can learn efficiently even

Example	Framework	M	$\bar{e}$ [m]	$\sigma(e)$ [m]	$e_{\max}$ [m]
2D	CNN	372	0.06 E-3	0.2 E-4	0.001
	MAgNET		0.17 E-3	0.7 E-4	0.021
	Perceiver IO		0.02 E-3	0.1 E-4	0.001
3D	MAgNET	400	8.92 E-3	1.9 E-3	0.307
	Perceiver IO		2.60 E-3	1.1 E-3	0.098

Table 6.3 Error metrics over the test set using the proposed NN frameworks. M stands for the number of test examples, and $\bar{e}$, $\sigma(e), e_{\max}$ are error metrics defined in Section 6.2.2.

with the increased mesh complexity and input size. In case of Perceiver IO, as the size and complexity of mesh further increases, it becomes less and less robust and fails to learn efficiently. We observed that Perceiver IO failed to learn on the data with input dimension higher than 10^4.

A possible interpretation of this behavior is that in the case of MAgNET topological information is externally provided through the adjacency matrix. Hence MAgNET can efficiently learn by leveraging inter-dependencies between different nodal feature values in the data. On the other hand Perceiver IO implicitly learns the nodal data dependencies and as the size of the input data increases, the task to find these inter-dependencies gets more difficult. Evidence of this behavior can be seen in Figure 6.6, where it is clear that Perceiver IO is optimizing through a much more complex objective function with a higher density of local minimas.

Once trained, proposed DNN frameworks are fast at the inference stage while predicting unseen examples. Table 6.4 provides training and inference time (for a single test example) for all three frameworks, for comparison FEM solution time is also provided. In particular, Perceiver IO takes a much longer time during the training phase but is extremely fast at the inference stage. It could make predictions on both small scale (2D) and large scale (3D) inputs in almost similar time. It has to be noted that to ensure the convergence of the iterative solver, the non-linear FEM problem is solved with incremental load steps. Hence the solution time for FEM increases with the magnitude of external force. Whereas trained DNN frameworks take almost similar time at the inference stage irrespective of external force magnitudes.

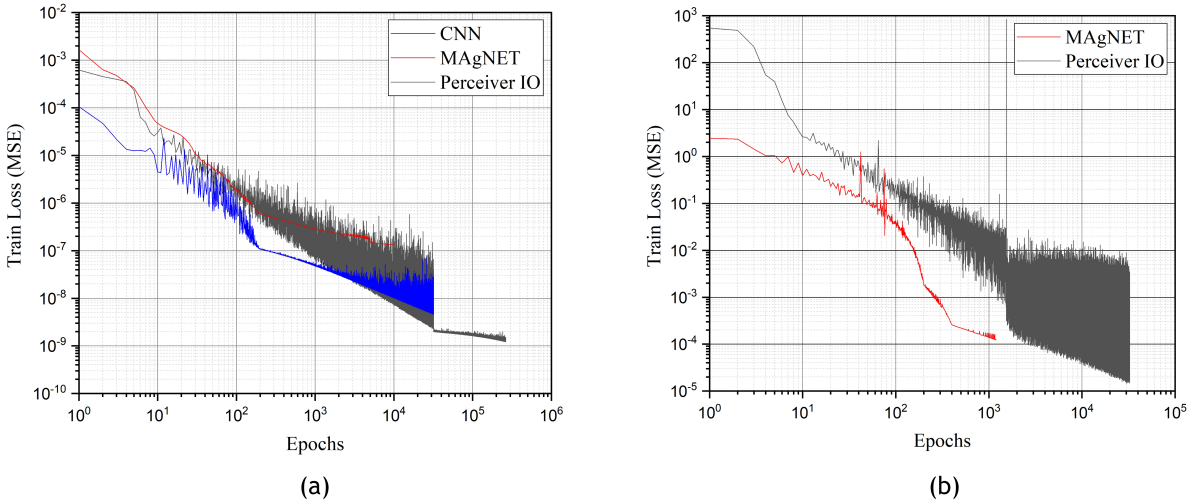


Fig. 6.6 Training convergence for the proposed neural network frameworks for the **(a)** 2D case **(b)** 3D case.

6.3.5 Qualitative analysis of individual examples

In this section, we analyze deformations of individual examples by giving a qualitative comparison of predictions obtained using different networks. In particular, we analyze test examples with the maximum nodal displacement for 2D as well as the 3D case. In both cases, we plot nodal error contours standing for the absolute difference between the DNN prediction and the true FEM solution.

2D case

The analyzed example in the Figure 6.7 stands for the maximum nodal displacement example in the 2D test dataset. The node indicated by the green dot has the maximum nodal displacement of 0.35 m. While the pink arrows represent corresponding nodal forces for the line density force applied on those four nodes.

Figure 6.7 shows absolute error counters of nodal displacements predicted using proposed DNN frameworks when compared with the FEM solution. All the proposed neural network frameworks can accurately predict the deformed mesh. Percentage prediction error (when compared to the true FEM solution) for the green node is 0.03%, 0.42%, 0.06% using CNN, MAgNET, and Perceiver IO network respectively. We observed that for the small-scale structured inputs, both Perceiver IO and CNN U-Net could make better predictions when compared to MAgNET. In case of Perceiver IO, advantage likely comes from the network leveraging its capability of

Example	Framework	N. parameters (× E6)	Training time (hrs)	Inference time (s)	FEM solver time (s)
2D	CNN U-Net	4.8	18	0.021	0.6
	MAgNET	4.5	132	0.040	
	Perceiver IO	1.9	521	0.006	
3D	MAgNET	33.9	161	0.217	2.5
	Perceiver IO	4.4	312	0.006	

Table 6.4 Comparison of training and inference times for all the three networks implemented in this work.

learning long-range correlations more accurately. This stems from the fact that Perceiver's inputs are not constrained by any topological assumption.

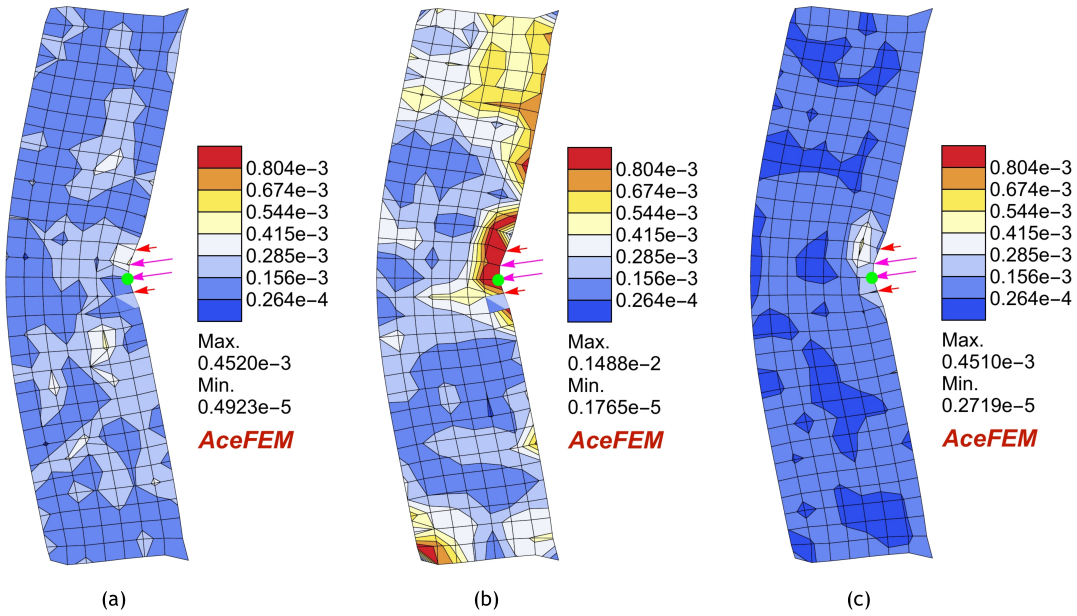


Fig. 6.7 Prediction error for different neural network frameworks when compared to true FEM solution, plotted on the deformed mesh (obtained using the same framework). Force with line density of (-21.6645,-2.99384) N is applied as shown with pink arrows. True displacement of the green node is 0.35m. Nodal error contours obtained **(a)** using CNN U-Net **(b)** using MAgNET **(c)** using Perceiver IO.

3D case

The 3D case is aiming to demonstrate the performance for unstructured meshes, that are commonly used when dealing with real world geometries. Tetrahedron discretisation of the continuous 3D domain is irregular and the mesh topology is complex. The fixed nodes are topologically far from the tip of the elephant trunk, thus making it challenging to communicate the boundary condition information. We show that both MAgNET and Perceiver IO can learn on such complex real-world examples efficiently.

Again we consider the test example with maximum nodal displacement. Figure 6.8 shows that both MAgNET (Figure 6.8a,6.8c,6.8e) and Perceiver IO (Figure 6.8b,6.8d,6.8f) solutions are able to predict non-linear mesh deformations accurately. We further analyse the absolute nodal error counters for both predictions by plotting them on the deformed meshes predicted using respective DNN frameworks. Both front (Figure 6.8e-6.8f) and side views (Figure 6.8c-6.8d) obtained using MAgNET and Perceiver IO solutions respectively indicate low prediction errors for both frameworks. The green node (at the tip of the ear) shown in (Figure 6.8e-6.8f) has a maximum nodal displacement of 140.04 m for this example. The percentage prediction error when compared to the true FEM solution for this green node is 0.03%, and 0.02% using MAgNET and Perceiver IO networks respectively. Both MAgNET and Perceiver IO are observed to make efficient predictions, with Perceiver IO giving relatively low nodal errors for this demanding case. Also, owing to the lesser number of trainable parameters, Perceiver IO is much faster at the inference stage.

6.4 Conclusion

In this work, we demonstrated the capabilities of three promising deep neural network (DNN) frameworks for accurate and fast predictions of non-linear deformations of solid bodies. We compared their performance on two benchmark examples, in which data was generated by the finite element method. Although we only tested the frameworks for the Noe-Hoohean material model, they are compatible with more general hyperelastic models, such as Mooney–Rivlin or Ogden models. As such, they promise to be used as surrogate models for non-linear computational models in mechanics.

The comparison included two very recent DNN frameworks, MAgNET and Perceiver IO, that are naturally able to work with arbitrarily structured data at inputs/outputs, including complex finite element meshes that originate from real-world applications. The third compared framework, CNN U-Net, could only operate on grid inputs/outputs, and we suggested possible remedies to extend it to work with arbitrary unstructured meshes. When looking at prediction

capabilities, especially interesting are the capabilities of the Perceiver IO network, which demonstrated to give better predictions with a lesser number of parameters, as compared to MAgNET and CNN U-Net. Additionally, the use of Perceiver IO creates a direct link to rapidly advancing research in ML and AI communities, which promises further advancements.

MAgNET and Perceiver IO are designed to be flexible in terms of the input and output structures, allowing them to potentially be applied to a wide range of problems. One possible application for these types of neural networks is in *ab initio* multi-scale modeling, which is also pursued in our team, see Hauseux et al. [2020]. These methods could be used to accelerate computationally expensive accurate simulations of large atomic systems by helping to connect atomic-level simulations with the macroscopic continuum description of materials. As such, these neural networks could lead to significant strides in the field of materials science.

One of the first possible future extensions of the presented frameworks would be to incorporate the physics-informed neural network paradigm. This can be easily achieved by incorporating relevant physical laws in the optimization objective of the training procedure. Such extension can further increase the accuracy of predictions and accelerate the training procedure. Another possible extension is to consider a much wider class of phenomena and models, including buckling instabilities and more general history/time-dependent phenomena (visco-elasticity, dynamics, plasticity, etc.), which would allow tackling more challenging problems in solid mechanics, see e.g., [Vijayaraghavan et al., 2021a]. Going beyond mechanics, these approaches can be also adopted for a much wider range of engineering and scientific applications.

Data Availability Statement

The datasets generated for this study can be found at https://doi.org/10.5281/zenodo.7585319 zenodo repository, and source codes are made available in the convolution-aggregation-attention GitHub repository.

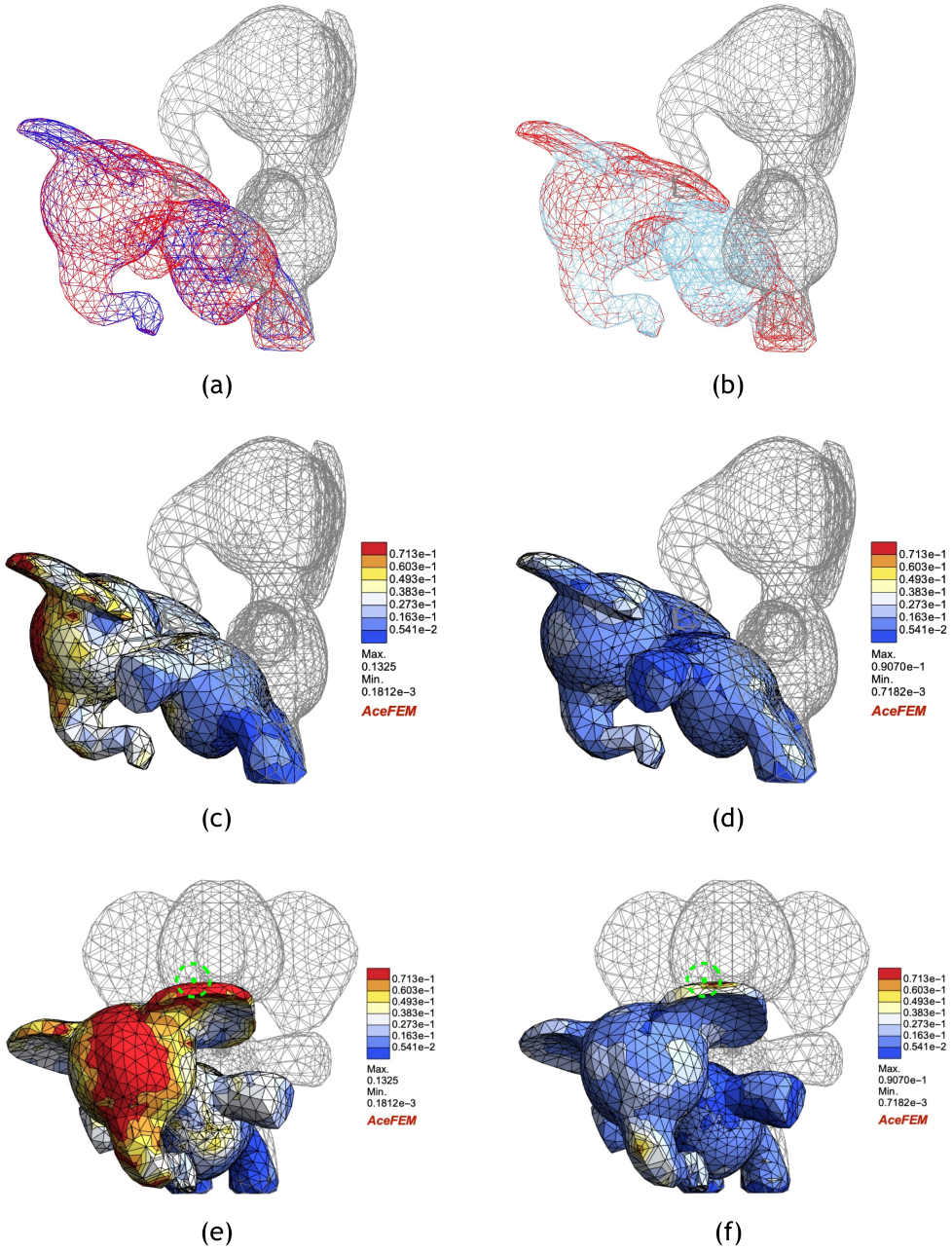


Fig. 6.8 Deformation of elephant mesh subjected to external body force density (0.34, 0.0, 0.35) N/kg. First column represents MAgNET solutions while the second column represents Perceiver IO solutions. **(a)&(b)** Deformed meshes using MAgNET (dark blue) and Perceiver IO (sky blue) respectively, for comparison FEM mesh is presented in red. The rest position is indicated with gray mesh. **(c)&(d)** Side view of nodal error contours when compared to the FEM solution, plotted on the deformed meshes for MAgNET and Perceiver IO solution respectively. **(e)&(f)** Front view of nodal error contours for MAgNET and Perceiver IO respectively. The true displacement of the green node is 140.04 m.

Chapter 7

A probabilistic reduced-order emulation framework for nonlinear solid mechanics

Abstract

In many real-life applications, it is crucial to produce reliable uncertainty estimates in addition to the predictions. But quantifying uncertainty on high-dimensional solutions is still a severely under-invested problem, especially for non-linear simulations. This work introduces an innovative approach that combines autoencoder neural networks with Gaussian processes—a Bayesian machine learning method—to address the challenge of finding probabilistic mappings between high-dimensional inputs and outputs. We validate the proposed framework for its application to surrogate modeling of non-linear finite element simulations. Our early findings highlight that the proposed framework is computationally efficient as well as accurate in predicting non-linear deformations of solid bodies subjected to external forces, all the while providing insightful uncertainty assessments.

This chapter is to be improved and submitted as: S. Deshpande, H. Rappel, S.P.A. Bordas, J. Lengiewicz, *A probabilistic reduced-order emulation framework for nonlinear solid mechanics*

7.1 Introduction

Real-world systems often exhibit complicated and nonlinear mechanical behavior. The most common approach to model the response of these systems is the finite element method. However, for systems with a large number of degrees of freedom, executing standard numerical methods such as FEM can become prohibitively expensive. Furthermore, regardless of the mathematical complexity, every numerical model inherently simplifies reality. Consequently, quantifying uncertainty becomes indispensable in any numerical framework aimed at real-world problems. However, the computational cost of the standard methodologies makes uncertainty quantification for such high dimensional problems impossible. Therefore, in this contribution, we aim at addressing these challenges with a novel framework that combines probabilistic surrogate models (Gaussian process emulators) and autoencoder neural networks. In simple words, our framework predicts non-linear deformations of high dimensional meshes (corresponding to solid bodies) subjected to external forces, along with their uncertainties.

As the number of degrees of freedom in a problem increases, numerical challenges arises when using conventional methods such as FEM. In these scenarios, reduced ordered methods (ROMs) offer a means to reduce the dimensionality of the problem, enabling more efficient and manageable simulations. Conventional ROMs accomplish this by linearly projecting higher-order information to lower-dimensional space using dimensionality reduction techniques such as Principal Component Analysis (PCA) [Aversano et al., 2019; Grassi et al., 2014], and Proper Orthogonal Decomposition (POD) [Goury and Duriez, 2018a; Niroomandi et al., 2009]. However, these approaches yield undesirable results when simulating highly nonlinear phenomena [Kerfriden et al., 2011b; Kerschen et al., 2005; Schölkopf et al., 1998].

More recently, there's been a widespread adoption of machine learning techniques to accelerate computationally intensive numerical methods [Papavasileiou et al., 2023; Šarkić Glumac et al., 2023]. In particular, deep learning (DL) models have been successfully used as accurate and computationally inexpensive surrogates to solve non-linear problems in mechanics [Deshpande et al., 2023b,c; Krokos et al., 2022a,b]. In context of ROMs, DL methods are being extensively used to find reduced order representations of the high dimensional data [Wang et al., 2016]. DL based approaches are not only capable of efficient non-linear compressions but are also computationally inexpensive at the inference stage, and hence are highly suitable of applications requiring real time simulations [Fresca et al., 2021; Reddy et al., 2020].

The type of deep neural networks used to compress and reconstruct data to and from reduced states are also termed as autoendcoder networks. Autoencoder neural networks, as an unsupervised learning technique, has been commonly adopted to learn efficient codings across various domains [Liou et al., 2014; Masci et al., 2011; Wang and Cha, 2021]. The primary focus of

an autoencoder is to reduce the dimensionality of the input data in a non-linear way, while preserving its essential features [Hinton and Salakhutdinov, 2006]. Autoencoder networks work much better than POD or PCA, as a tool to reduce the dimensionality of the data [Almotiri et al., 2017]. They have also been implemented in the context of computational fluid dynamics [Murata et al., 2020; Pant et al., 2021] as well as computational solid mechanics [Fresca et al., 2022; Shinde et al., 2023]. In this work we use fully connected and convolutional neural network based autoencoder networks to find reduced representations of full field displacement solutions. In the context of proposed framework, creating latent representations of the data reduces complexity and computational burden of the probabilistic mapping task.

As mentioned earlier, it is essential in this framework to find a probabilistic mapping between reduced inputs and outputs obtained by the autoencoder framework. To this end, we employ a probabilistic regression model based on the Gaussian process regression (GPR). A regression model maps the input data to the output data (corresponding to the latent state) with a low computational cost, therefore once the related parameters are identified, the computational overhead of the regression model in our framework will be negligible. As a result, the process of estimating uncertainty—something that involves many simulations for various parameters, as shown in [Hauseux et al., 2017a]—becomes manageable and practical.

GPs are well-established, and the related theories are extensively discussed in standard textbooks [Rasmussen and Williams, 2006; Rogers and Girolami, 2016]. The flexibility (i.e., they can mimic different functions with a few parameters) and intrinsic capability of GPs to quantify uncertainties make them a perfect candidate for the application of this contribution. In mechanics also, GPs have been employed in a wide range of studies both as surrogate models Arendt et al. [2012]; Bayarri et al. [2007]; Kennedy and O'Hagan [2001]; Rappel et al. [2018] and an approach to model spatially varying parameter fields Koutsourelakis [2009]; Rappel et al. [2019, 2022]. Ding et al. [Ding et al., 2023] employed a combination of GPR and PCA to model displacement fields in problems with nonlinear materials (i.e., elastoplastic material), [Jidling, 2017] and [Poloni et al., 2023] used GPR to model a full-field of strain from sensor observations.

While GPR is a strong regression technique, multi-dimensional output GPs have a drawback in terms of computational efficiency. They exhibit cubic scaling in relation to both the number of data points and the dimensionality of each output [Bruinsma et al., 2020]. This work focuses on alleviating this issue, by significantly reducing the output dimension, while keeping same number of data points. In simple words, we initially reduce the 2D/3D-FE problem into a reduced problem and then use GPR to map inputs to reduced outputs. During the prediction phase, for a given force in its sparse representation, GPs predict the latent space displacement solutions which are eventually projected to the full field solutions using the autoencoder network.

We demonstrate the robustness and versatility of our framework by applying it to synthetic datasets generated from the non-linear finite element simulations.

The remainder of this paper is organized as follows. In Section 2 we present overview of the framework. In Section 3 we describe general methodology of the framework. Then, in Section 4, an extensive study of the proposed framework is performed, which is based on the 2D and 3D benchmark examples. The conclusions and future research directions are outlined in Section 5.

7.2 Methodology

As motivated in the introduction, the framework aims to compute full field displacements of solids subjected to external forces, along with the associated uncertainties. In this section, we begin by providing an overview of the framework, followed by an in-depth discussion of its individual components.

7.2.1 Overview of the framework

The framework is categorized into following two phases.

Offline phase: Training of GP+autoencoder framework

In the offline phase, an autoencoder network is trained only on full-field displacement data to get the corresponding compressed representations. Following that, the GP is trained on the force inputs (provided in their sparse representation) and compressed displacements obtained using the autoencoder network as described in Figure 7.1. Autoencoder and GP training procedures are performed independently.

Online phase: Prediction for unseen force

In the online phase, for a given unseen input force, GP predicts the latent displacement distribution. These latent displacements are then reconstructed to the full space using decoder part of the autoencoder network as illustrated in the Figure 7.2.

7.2.2 Obtaining latent representations with autoencoder neural networks

Autoencoders are neural networks that are primarily used for unsupervised learning and dimensionality reduction tasks. They are designed to encode high-dimensional input data into

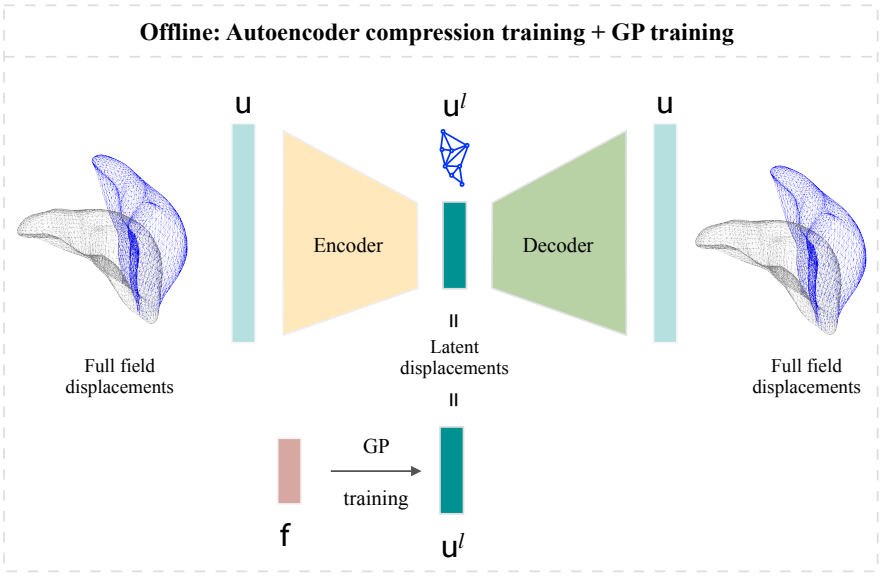


Fig. 7.1 First, the autoencoder neural network is used to compress full field displacement data to its latent space representation. Next, GP training is performed to find force-displacement probabilistic mapping in the latent space.

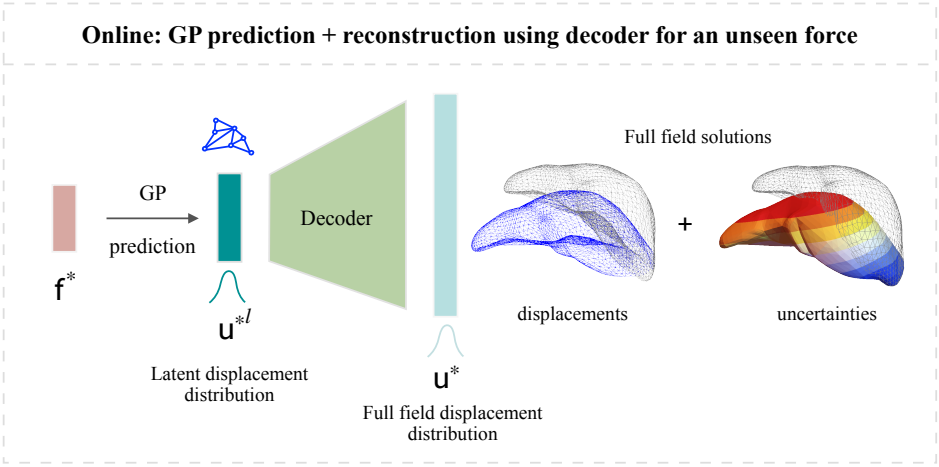


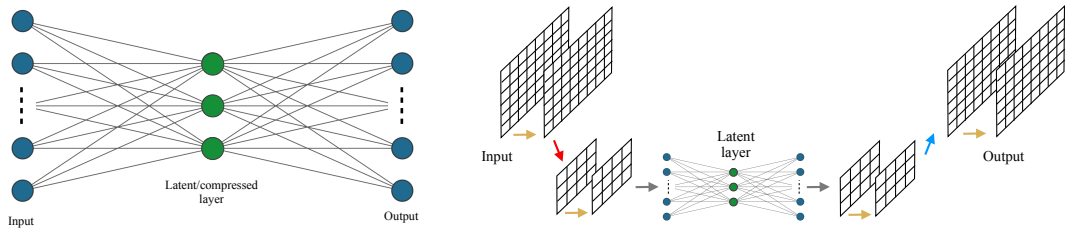
Fig. 7.2 For an unseen input force $\mathbf{f}^*$, the GP is used to predict the probability distribution of latent displacements. Subsequently, these latent displacements are mapped to the full field state by using the decoder component of the autoencoder network.

a lower-dimensional latent space representation, and then decode it to the original input space, aiming to reconstruct the input data as accurately as possible.

The architecture of an autoencoder network typically consists of two main components: an encoder and a decoder. The encoder takes the input data and maps it to a compressed representation in the latent space, while the decoder takes this compressed representation and

reconstructs the original input data. The encoder and decoder are usually symmetrical, meaning that they have the same number of layers and the layer sizes are mirrored. One of the key aspects of autoencoders is that the latent space, also referred to as the bottleneck layer, has a lower dimensionality than the input space. This forces the network to learn a more efficient and meaningful representation of the data. By compressing the data into a lower-dimensional space and then reconstructing it, autoencoders can capture the most salient features by discarding less relevant information.

In our study, we employ two distinct autoencoder architectures. The first type utilizes fully connected networks, suitable for handling arbitrary unstructured meshes. Meanwhile, the second type utilizes convolutional neural networks (CNNs) specifically designed for structured mesh scenarios [Deshpande et al., 2022; Mendizabal et al., 2019b]. The CNN autoencoder networks integrate fully connected layers with convolution and maxpooling layers, see Figure 7.3.



(a) (b)

Fig. 7.3 By introducing a bottleneck within the network, we create a compressed representation of the original input, allowing for efficient knowledge encoding. Note that inputs and outputs of the network are identical, which are full field displacements in our case. (a) Schematic of a fully connected autoencoder neural network. (b) Schematic of a convolution neural network autoencoder network.

This work utilizes numerically generated force-displacement datasets, denoted as $\mathcal{D}^f = \{(\mathbf{f}_i, \mathbf{u}_i)\}_{i=1,..,N}$. This datasets denotes non-linear displacements, $\mathbf{u}$, of solid bodies when subjected to external forces, $\mathbf{f}$. The solid body can be subjected to either loads applied to a single or selective set of nodes or body forces acting throughout the entire region. When dealing with point loads, we describe the force using its magnitude in the (x, y) direction and the position where the force is applied relative to the fixed boundary. This approach allows us to represent force inputs efficiently, resulting in a sparse representation. In scenarios involving body forces, we provide the force density components in either the (x, y) directions or the (x, y, z) directions, depending on the dimensionality of the specific problem.

For the displacement outputs, arrays match the size of the problem's degrees of freedom (DOFs), representing full field displacements. Conversely, force arrays are considerably smaller than displacement arrays. This eliminates the need for their separate compression into a latent state. Hence, autoencoder networks are used to compress only displacement fields. However, if

necessary, latent forces can also be computed by compressing their full field representation with autoencoder networks.

The autoencoder network, $\mathcal{U}$, is constructed by subsequently applying the encoder and decoder networks

$$\mathcal{U}(\mathbf{u}, \boldsymbol{\theta}_{\text{auto}}) = \mathcal{U}_{\text{decoder}}(\mathcal{U}_{\text{encoder}}(\mathbf{u}, \boldsymbol{\theta}_{\text{auto}}), \boldsymbol{\theta}_{\text{auto}}), \tag{7.1}$$

where $\mathbf{u}$ stands for the full field displacements and $\mathbf{u}^l = \mathcal{U}_{\text{encoder}}(\mathbf{u}, \boldsymbol{\theta}_{\text{auto}})$ is their corresponding latent/compressed representation. For a given full field displacement dataset, $\{\mathbf{u}_i\}_{i=1,..,N}$, the autoencoder network is trained by minimizing the following mean squared error loss

$$\mathcal{L}(\mathcal{D}^f, \boldsymbol{\theta}_{\text{auto}}) = \frac{1}{N} \sum_{i=1}^{N} \|\mathcal{U}(\mathbf{u}_i, \boldsymbol{\theta}_{\text{auto}}) - \mathbf{u}_i\|_2^2, \tag{7.2}$$

where $\boldsymbol{\theta}_{\text{auto}}$ are trainable parameters of the autoencoder network. The optimal parameters, $\boldsymbol{\theta}^*_{\text{auto}}$, are retrieved by minimizing the loss function:

$$\boldsymbol{\theta}^*_{\text{auto}} = \arg\min_{\boldsymbol{\theta}_{\text{auto}}} \mathcal{L}(\mathcal{D}^f, \boldsymbol{\theta}_{\text{auto}}). \tag{7.3}$$

So once the autoencoder network is trained, for a given full field displacement field $\mathbf{u}$, latent representation $\mathbf{u}^l$ is computed using the encoder part of the autoencoder network as follows:

$$\mathbf{u}^l = \mathcal{U}_{\text{encoder}}(\mathbf{u}, \boldsymbol{\theta}^*_{\text{auto}}) \tag{7.4}$$

Hence, we obtain latent state dataset $\mathcal{D}^l = \{(\mathbf{f}_i, \mathbf{u}^l_i)\}_{i=1,..,N}$, from the full field dataset $\mathcal{D}^f = \{(\mathbf{f}_i, \mathbf{u}_i)\}_{i=1,..,N}$. Note that forces are identical for both datasets and they are always provided as smaller dimensional arrays, which is discussed in detail in Section 7.3.1.

In the next section, we will use the latent dataset, $\mathcal{D}^l$, to elaborate Gaussian process model, which will provide the transformation between the input forces, $\mathbf{f}$, and the latent solutions, $\mathbf{u}^l$.

7.2.3 Gaussian Process Regression in the latent space

GPR is a key component of our framework. This subsection provides a brief and practical introduction to GPs. Interested readers are referred to [Rasmussen and Williams, 2006] for more detail. One may consider GPs as an extension of multivariate normal distributions into an infinite-dimensional Gaussian distribution [Gelman et al., 2003]. Consider inputs for two data points $(\mathbf{f}, \mathbf{f}')$, a GP is completely specified by its mean function and covariance function. We define mean function $m(\mathbf{f})$ and the covariance function $k(\mathbf{f}, \mathbf{f}')$ of a real process $w(\mathbf{f})$ as:

$$\begin{aligned} m(\mathbf{f}) &= \mathbb{E}(w(\mathbf{f})) \\ k(\mathbf{f},\mathbf{f}') &= \mathbb{E}[(w(\mathbf{f}) - m(\mathbf{f}))(w(\mathbf{f}') - m(\mathbf{f}'))] \end{aligned} \tag{7.5}$$

We use a GP to describe a distribution over functions, a realization of GP reads as:

$$w(\mathbf{f}) \sim (m(\mathbf{f}), k(\mathbf{f},\mathbf{f}')), \tag{7.6}$$

where $\mathbf{f},\mathbf{f}'$ are two GP's input vectors of dimension D, which are force vectors in the scope of present framework. Hence, in general, for a finite collection of inputs $[\mathbf{f}_1 \cdots \mathbf{f}_N]$:

$$\mathbf{w} \sim (\mathbf{m}, \mathbf{K}), \tag{7.7}$$

where $\mathbf{w} = \begin{bmatrix} w_1 & \cdots & w_N \end{bmatrix}^T$, $\mathbf{m} = \begin{bmatrix} m(\mathbf{f}_1) & \cdots & m(\mathbf{f}_N) \end{bmatrix}^T$ and $\mathbf{K}$ denotes the covariance matrix between two inputs, i.e., $(\mathbf{K})_{ij} = k(\mathbf{f}_i, \mathbf{f}_j)$. Frequent choices for the covariance function are given in [Rasmussen and Williams, 2006].

In simple words, in GPR, we set a GP (Eq. (7.6)) as a prior for the function that is mapping inputs to outputs and then update the prior GP with available observations using Bayes' rule. In our framework, GP predicts the displacements which correspond to the latent space representations as shown in the Figure 7.1& 7.2, hence we will use superscript-l notation, $()^l$, for all the GP outputs.

Let $\mathbf{u}^l$ be a set of Gaussian distributed observations, i.e., $\mathbf{u}_i^l = (w(\mathbf{f}_i), \sigma^2)$, with $i = 1, \cdots, N$, without loss of generality we assume that function that maps inputs to outputs is a realization of a GP with a zero mean and a covariance function $k(\mathbf{f},\mathbf{f}')$ which is defined by its parameter set $\boldsymbol{\theta}_{\text{GP}}$, i.e., $u_i^l = w(f_i) + \omega$ and $\omega \sim (0, \sigma^2)$. Let $\boldsymbol{y} = [\mathbf{u}_i^l, \cdots, \mathbf{u}_N^l]$, then the predictions of a GPR for a new data point $(\mathbf{f}^*, w^*)$ will be as follows:

$$w^* | \boldsymbol{\theta}_{\text{GP}}, \mathbf{u}^l, \mathbf{f}, \sigma^2 \sim (\mathbb{E}(w^*), \mathbb{V}(w^*)), \tag{7.8}$$

$$\mathbb{E}(w^*) = \mathbf{k}_*^T (\mathbf{K} + \sigma^2 \mathbf{I}_n)^{-1} \boldsymbol{y}, \tag{7.9}$$

and

$$\mathbb{V}(w^*) = k(\mathbf{f}^*, \mathbf{f}^*) - \mathbf{k}_*^T (\mathbf{K} + \sigma^2 \mathbf{I}_n)^{-1} \mathbf{k}^*, \tag{7.10}$$

where $\mathbb{E}$ denotes mean or expected value, $\mathbb{V}$ denotes the variance, $\mathbf{I}_N$ is an $N \times N$ identity matrix, $\mathbf{K}$ denotes the $N \times N$ covariance function given in Eq. (7.7), $\mathbf{k}_* = \mathbf{K}(\mathbf{f}^*, \mathbf{F})$ is an $N \times 1$ row vector, and $\mathbf{F}$ is a $D \times N$ matrix storing the input column vectors for all N observations $(\mathbf{f}_1, \mathbf{f}_2, .., \mathbf{f}_N)$.

Furthermore, the optimal parameters of the covariance function (hyperparameters), can be identified by maximizing the following equation.

$$\log p(\boldsymbol{y}|\mathbf{F}, \boldsymbol{\theta}_{\text{GP}}, \sigma^2) = -\frac{1}{2}\boldsymbol{y}^T (\mathbf{K} + \sigma^2 \mathbf{I}_n)^{-1} \boldsymbol{y} - \frac{1}{2} \log[\det(\mathbf{K} + \sigma^2 \mathbf{I}_n)] - \frac{n}{2} \log(2\pi) \tag{7.11}$$

Note that, $p(\boldsymbol{y}|\mathbf{F}, \boldsymbol{\theta}, \sigma^2)$ in Eq. (7.11), is the conditional likelihood of observations given the parameters set $\boldsymbol{\theta}_{\text{GP}}$, the input matrix $\mathbf{F}$, σ^2, hence one also can use Bayesian inference for identification of parameters by setting priors on unknowns (i.e., $\boldsymbol{\theta}_{\text{GP}}$, and σ).

Moreover, GP and autoencoder network are trained independently. Autoencoder network needs to be trained first in order to generate the latent state dataset, which is then used to train the GP.

7.2.4 Projecting latent GP predictions to the full field space using decoder

As illustrated in Figure 7.2, the latent space predictions obtained using the GP are transformed into the full space through the decoder component of the autoencoder network. During the inference stage, when presented with a new input force $\mathbf{f}^*$, the GP initially generates a distribution of latent displacements denoted as $p(\mathbf{u}^{*l}|\mathbf{f}^*)$, which takes the form of a Gaussian distribution. For simplicity, we refer to this distribution as $\mathcal{N}(\mathbb{E}_{gp}, \mathbb{V}_{gp})$, where, $\mathbb{E}_{gp}, \mathbb{V}_{gp}$ are obtained from Eq 7.9 and Eq 7.10.

$$p(\mathbf{u}^{*l}|\mathbf{f}^*) = \mathcal{N}(\mathbb{E}_{gp}, \mathbb{V}_{gp}), \tag{7.12}$$

Subsequently, the distribution of full field displacements $p(\mathbf{u}^*|\mathbf{f}^*)$ is obtained by translating samples from the latent predictive distribution through the decoder component. Without loss of generality, let's assume that the distribution of full field displacements also follows a Gaussian distribution. The mean and standard deviation of this distribution are computed as follows:

$$\begin{aligned} \mathbf{u}_{\mu}^{*} &\approx \frac{1}{S}\sum_{s=1}^{S}\mathcal{U}_{\text{decoder}}(\mathbf{u}_{s}^{l}), \quad \text{where} \quad \mathbf{u}_{s}^{l} \sim p(\mathbf{u}^{*l}|\mathbf{f}^{*}), \\ \mathbf{u}_{\sigma}^{*2} &\approx \frac{1}{S}\sum_{s=1}^{S}\mathcal{U}_{\text{decoder}}(\mathbf{u}_{s}^{l})^{T}\mathcal{U}_{\text{decoder}}(\mathbf{u}_{s}^{l}) - \mathbf{u}_{\mu}^{*T}\mathbf{u}_{\mu}^{*} \end{aligned} \tag{7.13}$$

where $\mathbf{u}_s^l$ is a sample from the latent displacement distribution obtained using GP prediction, for an input $\mathbf{f}^*$. In order to get the full field displacement distributions, we generate 'S' samples in the latent space, and these samples are then projected to the original space using the decoder. Now the final Gaussian distribution of displacement of the full field solution is represented through the mean ($\mathbf{u}_\mu^*$) and standard devidation ($\mathbf{u}_\sigma^*$) of these 'S' samples. We set $S = 300$ for all the implementations presented in this work.

7.2.5 Finite element formulation for non-linear deformations of solid bodies

We consider a boundary value problem in the domain Ω, Dirichlet and Neumann boundary conditions are applied on Γ_D, Γ_N respectively. The virtual work principle for nonlinear elastostatic equation reads

$$\int_{\Omega} \boldsymbol{P}(\boldsymbol{F}(\boldsymbol{u})) \cdot \nabla \delta\boldsymbol{u}\, dV - \int_{\Omega} \rho\, \bar{\boldsymbol{b}} \cdot \delta\boldsymbol{u}\, dV - \int_{\Gamma_t} \bar{\boldsymbol{t}} \cdot \delta\boldsymbol{u}\, dS = 0 \qquad \forall \delta\boldsymbol{u}, \tag{7.14}$$

where $\boldsymbol{u}$ and $\delta\boldsymbol{u}$ belong to appropriate functional spaces, $\boldsymbol{u} = \bar{\boldsymbol{u}}$ and $\delta\boldsymbol{u} = \boldsymbol{0}$ on Γ_u, and $\boldsymbol{P}(\boldsymbol{F})$ is the first Piola-Kirchhoff stress tensor. The essential constitutive correlation is established using the hyperelastic strain energy potential denoted as $W(\boldsymbol{F})$, defined as follows:

$$\boldsymbol{P}(\boldsymbol{F}) = \frac{\partial W(\boldsymbol{F})}{\partial \boldsymbol{F}} \tag{7.15}$$

where $\boldsymbol{F} = \boldsymbol{I} + \nabla\boldsymbol{u}$ is the deformation gradient tensor. In this work we use following Neo-hookean energy density expression:

$$W(\boldsymbol{F}) = \frac{\mu}{2}(I_c - 3 - 2\ln J) + \frac{\lambda}{4}(J^2 - 1 - 2\ln J), \tag{7.16}$$

where $J = \det(\boldsymbol{F})$ and $I_c = \mathrm{tr}(\boldsymbol{F}^T\boldsymbol{F})$. Following conventional finite element discretization, the challenge represented by Eq. (7.14) will be reformulated into a system of nonlinear equations which are solved iteratively using Newton-Raphson method.

7.3 Results

7.3.1 Dataset generation information

In this research, we examine two distinct scenarios, as illustrated in Figure 7.4: a 2D beam and a 3D liver. These scenarios are subjected to varied loading conditions and are simulated using a non-linear finite element method during the data generation phase.

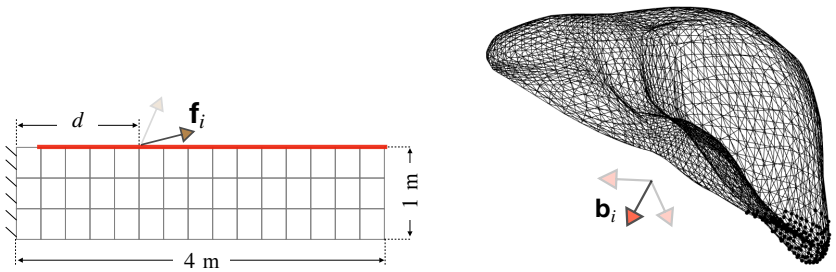
2D beam case: In this scenario, the 2D beam is applied with point forces with random direction and magnitude. These forces are applied to the nodes located along the top line, which is visually represented by red line in the Figure 7.4a. For the 2D beam case, displacement data is provided for all degrees of freedom (DOFs) of the mesh, capturing the full-field displacements of the beam. However, the forces used as inputs to the surrogate framework are represented as 3-dimensional arrays. Specifically, the applied point load is characterized by two components, namely f_x and f_y, representing the force magnitudes in the x and y directions, respectively. Additionally, the distance (d) from the fixed boundary to the point of application is considered. This information is visually depicted in Figure 7.4a.

3D liver case: In this case, the 3D liver model is subjected to body forces with random directions and magnitudes distributed throughout its volume. The liver is also constrained at specific nodes located at the right end. This simulation is aimed at understanding the deformations of the liver under the influence of these distributed body forces and boundary constraints. Similar to the beam case, displacement data is provided for all DOFs of the liver mesh, representing the full-field displacements of the organ. However, in contrast to the beam case, the forces used as inputs to the surrogate framework are the body force density components in the x, y, and z directions, respectively (b_x, b_y, b_z).

Further details, such as the range of external forces applied, as well as other information used to generate the datasets are given in the Table 7.1.

Problem	N.of DOFs ($\mathcal{F}$)	Range (External forces/ body force density)	Young's modulus E [Pa], Poisson's ratio ν, density ρ [kg/m^3]	Dataset size (train+test)
2D beam	128	f_x, f_y = -2.5 to 2.5 N	500, 0.4, -	$5700 + 300$
3D liver	9171	b_x, b_y, b_z= -0.02 to 0.02 N/kg	5000, 0.45, 1000	$7600 + 400$

Table 7.1 Details of FE datasets.



(a) (b)

Fig. 7.4 Schematics of examples considered in this work (a) 2D beam discretised with quad elements is applied with point loads on the nodes lying on red line, this example has been taken from [Deshpande et al., 2022]. (b) 3D liver is applied with random body forces within a prescribed region.

7.3.2 Implementation details

Autoencoder networks: Architectures & training details

In this study we demonstrate two autoencoder networks, one based on CNN architecture as shown in the Figure 7.5, which is used for the 2D beam case. And the fully connected autoencoder network as depicted in Figure 7.6, which is used for the liver case. Both networks are composed of an encoder and a decoder part as illustrated in respective figures. Auoencoder networks are used only to compress the displacement solutions of full field space to the corresponding latent states, whereas the forces are originally provided in the sparse format.

Architecture for the 2D beam case

Encoder: The input to the CNN autoencoder is a mesh displacement tensor, which is represented with 2 channels. These channels correspond to the displacements along the x and y DOFs of the mesh. The input tensor undergoes application of two convolutional layers, each with 256 channels and 3×3 filters as shown in Figure 7.5. Convolutional operations enable feature extraction and dimensionality reduction. Subsequently, a max-pooling operation is applied, which reduces spatial dimensions of the tensors while preserving the number of channels. This process is repeated again with 128 channels, further capturing important patterns in the data. At the latent level of the network, two convolution layers with 64 channels are applied, which is followed by flattening of the tensor. Now the flattened tensor is applied with a dense layer with 'l' units, which stands for the dimension of the compressed/latent state. This compressed representation contains crucial information about the input mesh, capturing the most relevant features.

Decoder: The Decoder takes a flattened tensor of dimension 'l' as its input. Initially, this input tensor undergoes processing via a dense layer. This strategic utilization of the dense

layer facilitates a transformation of the arbitrary latent dimensions into a new size that can be conveniently reshaped into a grid structure, a format that seamlessly aligns with the subsequent convolutional layer operations. For example with using a dense layer with 640 units as shown in the Figure 7.5 allows us to reshape the flat tensor to a $64 \times 5 \times 2$ tensor, on which series of upsampling and convolution operations are performed until the original spatial dimensions is achieved. In the end, a 2 channel convolution with 1×1 filter is applied to get back the original mesh shape.

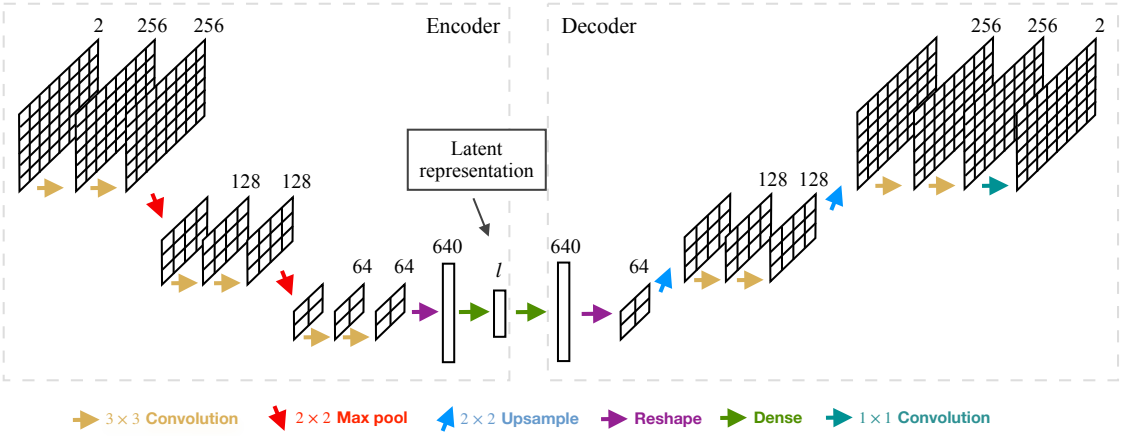


Fig. 7.5 CNN autoencoder architecture for used for the 2D beam case.

Convolutional neural networks are inherently designed to work with grid-like inputs, such as structured meshes. As a result, this methodology isn't immediately adaptable to inputs that lack this grid-based nature. To address this limitation, we introduce an alternative solution in the form of a fully connected autoencoder network as illustrated in Figure 7.6. This network can handle diverse and unstructured meshes.

Architecture for the 3D liver case

Encoder: This component receives input in the form of displacements linked to all degrees of freedom (DOFs) of the mesh. This information is encoded within a flattened tensor, which serves as the input layer. It is further applied with two dense layers with 4096 units each. We also introduce skip connections as illustrated in the Figure 7.6, which avoid vanishing gradient issues and stabalise the training procedure. This procedure applying blocks of dense layers and skip connections is repeated three more times with 2048, 1024, 512 units for dense layers respectively.

Decoder: Decoder takes the '$l = 16$' dimensional 1D tensor as the input (compressed/latent tensor), a similar procedure of applying two dense layers and skip connections is followed until the original size tensor is obtained. The number of units used in each dense layer is detailed in the Figure 7.6.

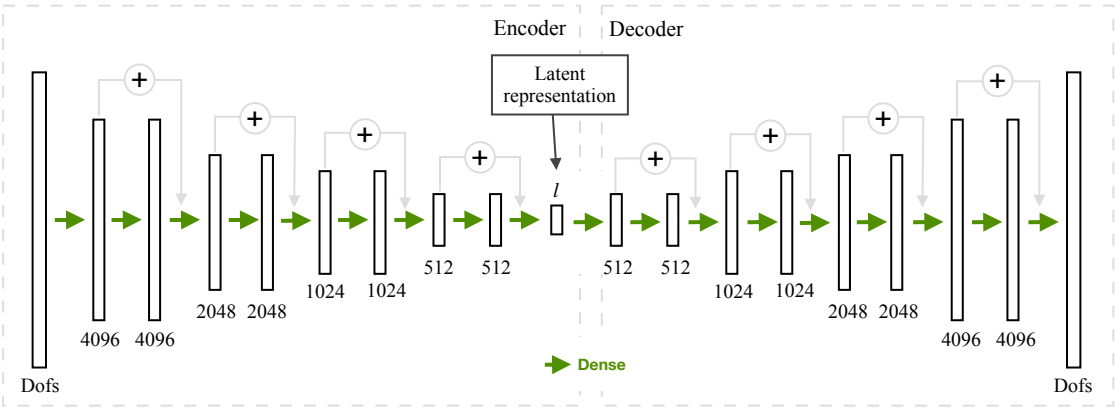


Fig. 7.6 Fully connected autoencoder architecture used for the liver case. The numbers denote respective number of units present in dense layers. We use latent dimension, $l = 16$ for the 3D liver case.

Training of autoencoder networks

Autoencoder network is trained on the full field displacement dataset, using the loss function as described by Eq.(7.2). Minimization is carried out using the Adam optimizer with recommended parameter configurations as presented by Kingma Kingma and Ba [2017]. In both instances, a batch size of 16 is employed, with the 2D beam case trained across 32000 epochs and the 3D liver case for 4000 epochs. The initial learning rate is set at 1e-4 and linearly decays to 1e-6 during the training process. Autoencoder networks are trained using Tensorflow [Dillon et al., 2017] library on a Tesla V100-SXM2 GPU, utilizing the high-performance computing facilities at the University of Luxembourg [Varrette et al., 2014].

Gaussian process details

All the GP implementations presented in the work are performed using radial basis kernel (RBF) and Matern kernel as described below:

$$
\begin{aligned}
k_{\text{RBF}}(\boldsymbol{f_i}, \boldsymbol{f_j}) &= exp(-\frac{\|\boldsymbol{f_i} - \boldsymbol{f_j}\|^2}{2l^2}) \\
k_{\text{Matern}}(\boldsymbol{f_i}, \boldsymbol{f_j}) &= \frac{1}{\Gamma(\nu)2^{\nu-1}}(\frac{\sqrt{2\nu}}{l}\|\boldsymbol{f_i} - \boldsymbol{f_j}\|^2)^{\nu} K_{\nu}(\frac{\sqrt{2\nu}}{l}\|\boldsymbol{f_i} - \boldsymbol{f_j}\|^2)
\end{aligned}
\tag{7.17}
$$

where $\boldsymbol{f_i}, \boldsymbol{f_j}$ are $i^{\text{th}}, j^{\text{th}}$ input vectors, $\Gamma(\nu)$ is the gamma function, K_ν is a modified Bessel function, and l is the length scale. The RBF kernel is one of the mostly commonly used kernels, and it assumes that the function values at nearby inputs are strongly correlated, and the correlation decreases as inputs move farther apart. Whereas the the Matern kernel is a

generalization of the RBF. It has an additional parameter (ν) which controls the smoothness of the resulting function.

The kernel hyperparameters are obtained by training GPs using scikit-learn library [Pedregosa et al., 2011], on the latent datasets as described in Section 7.2.3. Optimisation is performed using the L-BFGS-B optimizer [Fletcher, 2000], while the number of restarts of the optimizer is chosen as 9 for both the cases.

Validation metric

The performance of the framework over the entire test is computed using the mean absolute error metric. For the m^{th} test example, them mean error is given as follows:

$$e_m = \frac{1}{\mathcal{F}} \sum_{i=1}^{\mathcal{F}} |\mathbf{u}_\mu^*(\mathbf{f}_m)^i - \mathbf{u}_m^i|. \tag{7.18}$$

$\mathcal{F}$ is the number of DOFs of the mesh representing the full field space, $\mathbf{u}_\mu^*(\mathbf{f}_m)$ is the mean prediction of the GP+Decoder framework (full field prediction) as described through the Eq.(7.13) and $\mathbf{u}_m$ is the finite element solution for the full field space. To have a single validation metric over the entire test set, we compute the average mean norm $\bar{e}$ and the corrected sample standard deviation $\sigma(e)$ as follows:

$$\bar{e} = \frac{1}{M} \sum_{m=1}^{M} e_m, \qquad \sigma(e) = \sqrt{\frac{1}{M-1} \sum_{m=1}^{M} (e_m - \bar{e})^2}. \tag{7.19}$$

Finally, in addition to that, we also use the maximum error per degree of freedom over the entire test set

$$e_{\max} = \max_{m,i} |\mathbf{u}_\mu^*(\mathbf{f}_m)^i - \mathbf{u}_m^i|. \tag{7.20}$$

7.3.3 Performance over the test sets

The proposed framework predicts probabilistic displacement fields corresponding to the full field mesh. First, we analyse the prediction performance by comparing mean predictions of the framework to the true FEM solutions. Table 7.2 presents error metric for both benchmark cases. Additionally, the maximum nodal displacements for both cases are also provided, i.e., we

compute displacements of all the nodes for every single example, and provide the maximum nodal displacement in the table. Table 7.2 shows that both mean and maximum errors are fairly low when compared to the displacement magnitudes. The proposed framework is capable of predicting mechanical deformation responses with very low errors.

Example	M	$\bar{e}$ [m]	$\sigma(e)$ [m]	$e_{\max}$ [m]	Max nodal disp. [m]
2D Beam	300	0.1 E-3	0.7 E-4	4.3 E-3	1.16
3D liver	400	0.9 E-3	0.9 E-3	0.077	5.74

Table 7.2 Error metrics for 2D and 3D test sets for predictions using the proposed GP+autoencoder framework. M stands for the number of test examples, and $\bar{e}$, $\sigma(e)$ and $e_{\max}$ are error metrics defined in Section 7.3.2.

Further, we analyze test examples with the maximum nodal displacement for both benchmark cases. In both cases, we plot nodal error contours representing the absolute difference between the framework prediction and the true FEM solution. We also plot uncertainty estimates obtained using the proposed framework.

Visualisations

We analyse the test example of 2D beam case, with the maximum nodal displacement of 1.16 m. GP is implemented with the RBF kernel for this case. Figure 7.7 shows the displacement field prediction obtained using the proposed framework. Figure 7.7b and Figure 7.7c show that the mean predictions are in extremely well agreement with the true FEM solution. Figure 7.7d indicates low prediction errors while the Figure 7.7e provides nodal uncertainty estimates predicted by the framework (2 standard deviations), as obtained from the Eq.(7.13). It is evident from the figures that the observed errors are within the uncertainty bounds for respective nodal predictions.

Similarly, we analyse the test example of 3D liver case with the maximum nodal displacement of 5.74 m. GP is implemented with a Matern kernel for this case. Figure 7.8 shows that the framework is able to accurately predict the deformation responses of the 3D liver geometry subjected to external body forces. See Figure 7.8b, 7.8c for the mean prediction of the framework and FEM solution respectively. Figure 7.8d, 7.8e show the error and uncertainty predictions respectively. Prediction errors are extremely low, with the maximum nodal is about 0.2% of the maximum nodal deformation for this example. Figure 7.8e shows that most of the observed errors are within the uncertainty bounds bounds for respective nodal predictions.

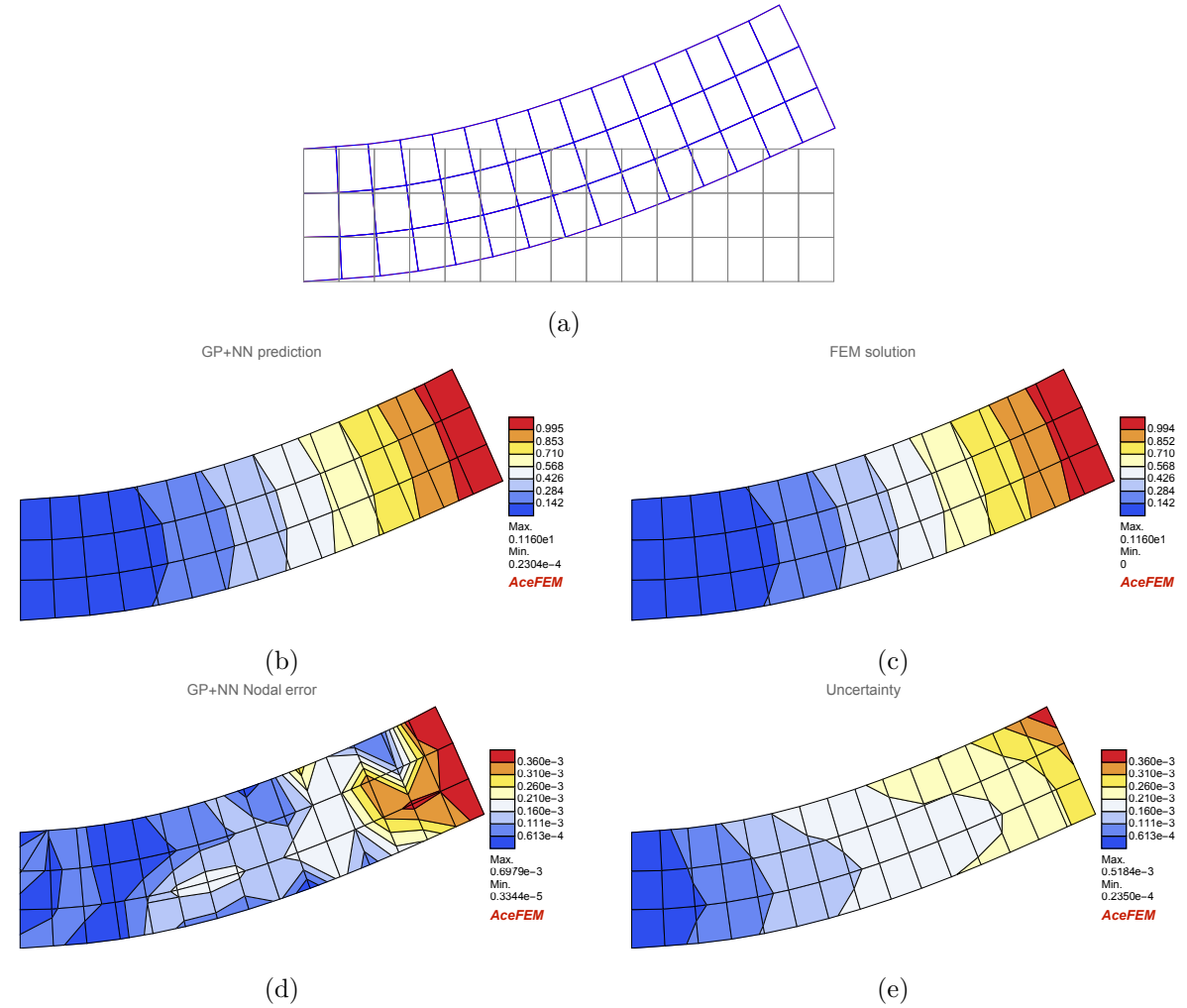


Fig. 7.7 Deformation of the 2D beam when applied with the point load on the top right corner node (a) Deformed mesh predicted using the framework is represented with the blue mesh, which is coinciding with the red mesh which represents the FEM prediction. Gray mesh represents the undeformed mesh. (b) & (c) Nodal displacements obtained using the proposed framework and FEM respectively. (d) Absolute nodal errors between the mean predictions and FEM solutions (e) Uncertainty predictions obtained using the framework.

GP model checking

To ensure the efficiency and accuracy of GP's predictions within the latent space, we plot probability distributions for a randomly chosen test example and compare them to corresponding true latent values. These true latent values are derived from the compression of full field Finite Element Method (FEM) solutions. As described in the Figure 7.6, we chose a latent dimension of 16 to compress 9171 dimensional displacement solution of the 3D liver case. Figure 7.9 shows probability distributions predicted for these 16 latent displacements. Figure shows that all the

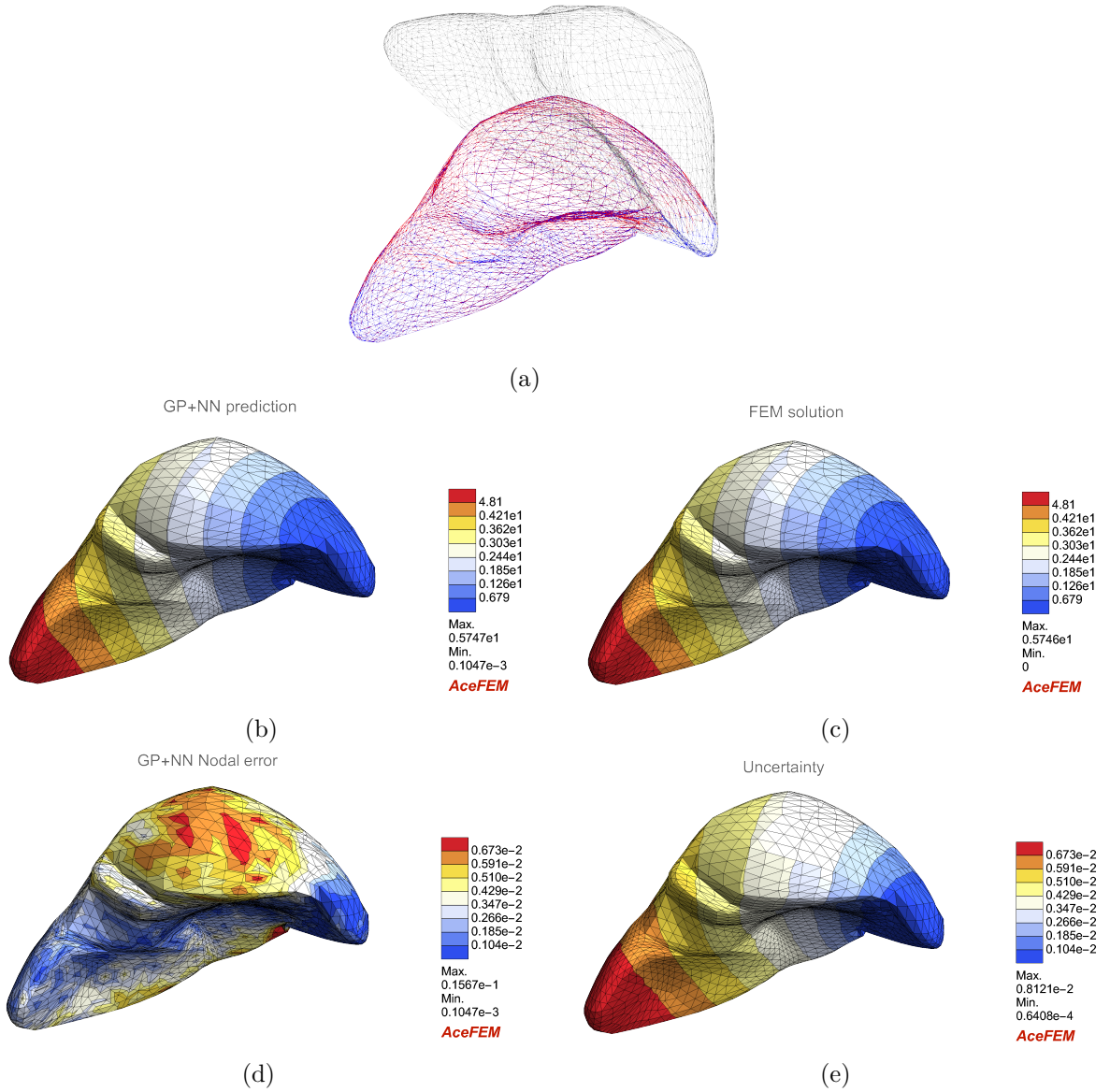


Fig. 7.8 Deformation of the 3D liver subjected to external body force. (a) Deformed mesh predicted using the framework is represented with the blue mesh, FEM solution is presented with the red mesh. Gray mesh represents the undeformed configuration. (b) & (c) Nodal displacements obtained using the proposed framework and FEM respectively. (d) Absolute nodal errors between the mean predictions and FEM solutions (e) Uncertainty predictions (two stds) obtained using the framework.

true latent solutions (represented by vertical red lines) are captured by distributions predicted by the GP.

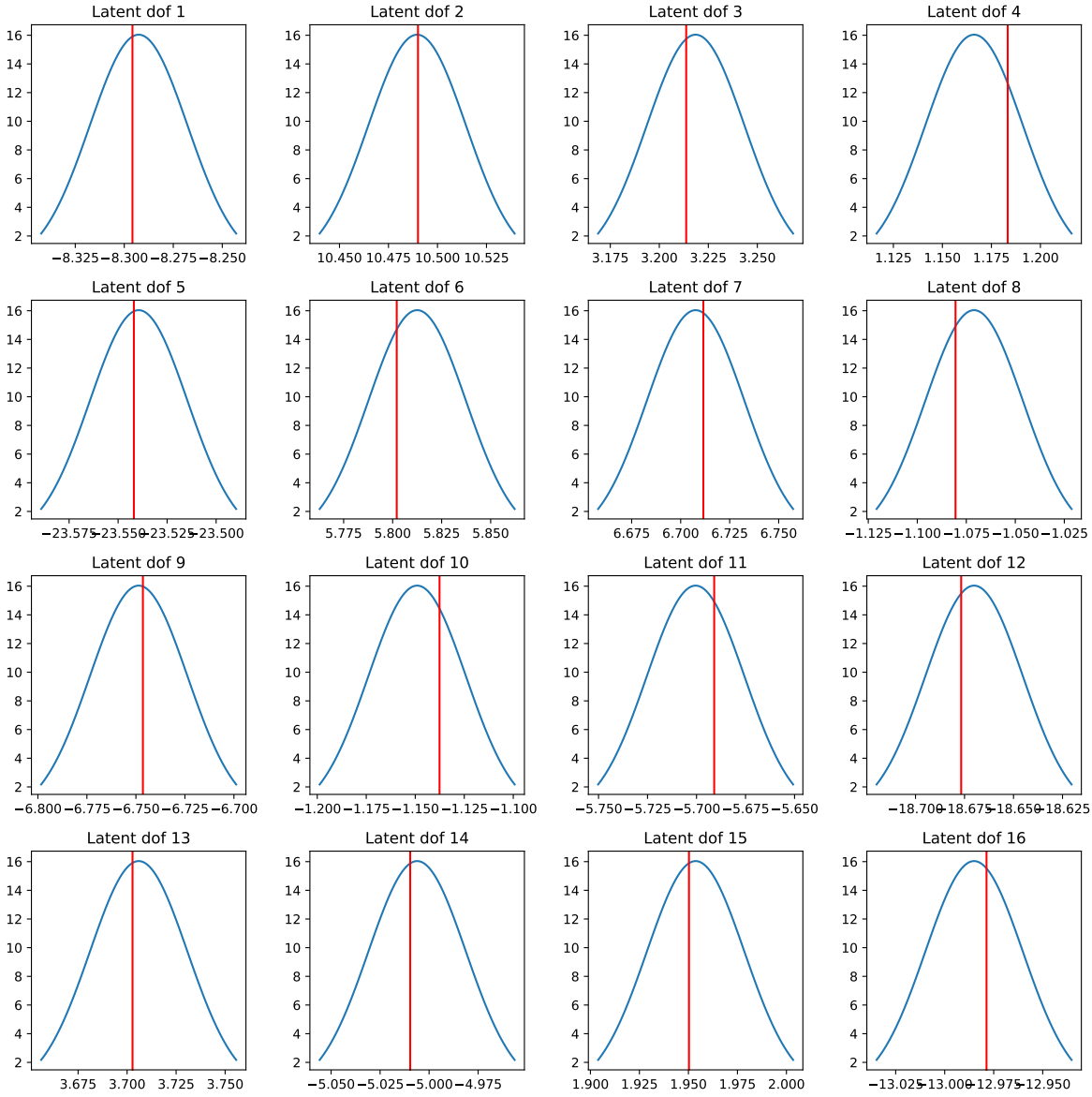


Fig. 7.9 Latent space GP predictions for a randomly chosen test example (test number 385) for the liver case. Latent space has a dimension of 16 for this case. GP is able to accurately capture true latent solutions that are indicated by vertical red lines.

7.4 Conclusion

This work presented a novel approach for probabilistic surrogate modeling of fidelity simulations. It combined Gaussian processes with reduced ordered modeling for efficiently simulating mechanics of solids. The reduced states are computed using two types of auto encoder neural networks. The autoencoder networks enabled to achieve an efficient non-linear compression of high dimensional displacement data, such as reducing $\sim$ 9100 DOFs to less than 20 dimensions

(as showcased through the liver example). This strategy greatly reduced the computational complexity and burden of probabilistic regression task of GP, which otherwise scales cubically with the output dimension size. In particular, we demonstrated that the proposed framework could accurately predict non-linear hyper-elastic deformations of solid bodies, along with the associated model uncertainties.

This study opens up new opportunities for future research. One exciting direction is applying this approach to probabilistic simulations of time and path dependent problems. One could use latent state to track how solutions evolve in such cases, which would make the whole process much faster and more efficient [Nikolopoulos et al., 2022]. Another direction would be to exploit state of the art graph autoencoder networks graph autoencoder networks to find compressed representations for arbitrary high-dimensional meshes [Barwey et al., 2023]. This advancement would overcome the limitations of fully connected networks, which are recognized for their challenges in handling problems with a high number of dimensions. Moving beyond mechanics, the proposed approach can also find its use across a broader spectrum of engineering and scientific domains.

Chapter 8

Concluding remarks

8.1 Summary and conclusions

This thesis presented novel deep-learning surrogate frameworks that address the challenges of accurate and fast simulations in solid mechanics. The proposed neural network frameworks are robust and scale efficiently with the dimensionality of the inputs. The proposed deterministic and Bayesian approaches have been trained using synthetic datasets, but they can straightforwardly assimilate experimental data. This capability makes our frameworks suitable for data-driven applications requiring fast response rates, for example, such as in patient-specific computer-aided surgery of soft human tissues.

In addition to utilising the existing neural network architectures, this thesis presented a novel geometric deep learning architecture called MAgNET. MAgNET enables efficient supervised learning on graph-structured data. MAgNET comprises two neural network operations: Multichannel Aggregation layer (MAg) and graph pooling/unpooling layers, which form a graph U-Net architecture capable of learning on large-dimension inputs/outputs. These newly proposed layers are compatible with state-of-the-art neural network layers and are useful in a wide range of scientific/engineering applications. Within the context of surrogate modeling for computational mechanics, MAgNET can handle arbitrary meshes, and it can capture nonlinear relationships in data, as demonstrated by quantitative cross-validation against ground-truth results obtained with FEM.

The thesis also utilised state-of-the-art attention-based architectures, such as transformers, for their uses in surrogate modeling of the mechanics of solids. We have presented a detailed performance analysis of Perceiver IO, an attention-based deep neural network, by comparing it with convolution and aggregation-based deep neural networks. Our comparison demonstrated

that the Perceiver IO network gave better predictions with fewer parameters when compared to MAgNET and CNN approaches. These results create a direct link to rapidly advancing research in ML and AI communities, which promises further advancements. The proposed frameworks can be used as surrogate models for non-linear computational models in mechanics and have the potential to accelerate computationally expensive accurate simulations of different engineering applications.

Apart from its predictive capabilities, the proposed frameworks focus on an important aspect, i.e. to provide reliable estimates of uncertainty. Mainly, this thesis presented two novel ideas based on Variational Bayes and Gaussian process formulations. Proposed approaches tackle the challenging problem of quantifying high-dimensional uncertainty on full-field solutions. Our analysis demonstrated a positive correlation between the predicted uncertainties and the prediction errors, which were computed as fitting errors to the FEM solution. Furthermore, we observed that the uncertainties quickly escalate in the extrapolated region, a property that we aimed to achieve, providing a means to trust the solution or not. Moreover, alongside accounting for model uncertainty, the proposed frameworks demonstrate an ability to effectively capture inherent data noises. Consequently, this thesis allows a significant advancement in establishing trustworthy and fast simulations in mechanics.

Overall, the contributions of this thesis provide important advancements in the fields of deep learning as well mechanics. The developed approaches have significant potential to be applied in a wide range of engineering applications. We believe that this work will serve as a reference for further developments in many emerging areas of research.

8.2 Future directions

There are numerous avenues for further research and expansion of the work presented in this thesis. One potential area for improvement is the integration of underlying physics into the learning process. While the current data-based frameworks can capture some of the underlying physics, we have observed localised errors while predicting physical quantities of interest. Incorporating these physical quantities of interest into the learning objective function could help to address this issue. The proposed frameworks can be straightforwardly extended to scalable Physics Informed Neural Network variants (such as physics-informed MAgNET).

Another promising direction for the extension of current work is to include path-dependent processes, such as elastoplasticity. This would require tracking the evolution of state variables in a time-stepping manner, which could be achieved using MAgNET as well attention-based

architectures like Perceiver IO. Attention-based approaches are capable of capturing long-range dependencies in sequential data, such as in the case of path and time-dependent problems.

Additionally, the Bayesian deep learning approach presented in Chapter 4 is only applicable to grid inputs, such as structured meshes. This concept can be directly incorporated into the MAgNET framework proposed in Chapter 5. This can be accomplished by performing local aggregations with probability distributions rather than discrete weights. A Bayesian MAgNET would be capable of tracking uncertainties inherent to neural network models as well as real-world data, for problems involving large-scale arbitrary meshes. However, we have already addressed this limitation through an another approach based on the proposed GP + autoencoder framework.

We believe that the generality of our neural network frameworks would support problems from different fields. They can handle problems with non-linear connections between variables and arbitrary divisions, especially in cases involving PDEs. The codes, datasets, and examples presented in this thesis are available open-access in respective GitHub and Zenodo repositories. We envision that the provided repositories will serve as a foundation for future developments and extensions in various emerging areas of research.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Diab W. Abueidda, Seid Koric, Nahil A. Sobh, and Huseyin Sehitoglu. Deep learning for plasticity and thermo-viscoplasticity. *International Journal of Plasticity*, 136:102852, 2021. ISSN 0749-6419. doi: https://doi.org/10.1016/j.ijplas.2020.102852. URL https://www.sciencedirect.com/science/article/pii/S0749641920302096.

Yinoussa Adagolodjo, Nicolas Golse, Eric Vibert, Michel De Mathelin, Stéphane Cotin, and Hadrien Courtecuisse. Marker-based registration for large deformations - application to open liver surgery. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4007–4012, 2018. doi: 10.1109/ICRA.2018.8462909.

Patricia Alcañiz, César Vivo de Catarina, Alessandro Gutiérrez, Jesús Pérez, Carlos Illana, Beatriz Pinar, and Miguel A. Otaduy. Soft-tissue simulation of the breast for intraoperative navigation and fusion of preoperative planning. *Frontiers in Bioengineering and Biotechnology*, 10, 2022. ISSN 2296-4185. URL https://www.frontiersin.org/articles/10.3389/fbioe.2022.976328.

P.E. Allier, L. Chamoin, and P. Ladevèze. Proper generalized decomposition computational methods on a benchmark problem: introducing a new strategy based on constitutive relation error minimization. *Advanced Modeling and Simulation in Engineering Sciences*, 2:17, 2015. doi: 10.1186/s40323-015-0038-4.

Jasem Almotiri, Khaled Elleithy, and Abdelrahman Elleithy. Comparison of autoencoder and principal component analysis followed by neural network for e-learning using handwritten recognition. In *2017 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pages 1–5, 2017. doi: 10.1109/LISAT.2017.8001963.

P.D. Arendt, D.W. Apley, and W. Chen. Quantification of model uncertainty: Calibration, model discrepancy, and identifiability. *Journal of Mechanical Design*, 134(10):100908, 2012.

Faisal As'ad, Philip Avery, and Charbel Farhat. A mechanics-informed artificial neural network approach in data-driven constitutive modeling. 01 2022. doi: 10.2514/6.2022-0100.

Gianmarco Aversano, Aurélie Bellemans, Zhiyi Li, Axel Coussement, Olivier Gicquel, and Alessandro Parente. Application of reduced-order models based on pca & kriging for the development of digital twins of reacting flow applications. *Computers & Chemical Engineering*, 121:422–441, 2019. ISSN 0098-1354. doi: https://doi.org/10.f1016/j.compchemeng.2018.09.022. URL https://www.sciencedirect.com/science/article/pii/S0098135418305891.

Roland Can Aydin, Fabian Albert Braeu, and Christian Johannes Cyron. General multi-fidelity framework for training artificial neural networks with computational models. *Frontiers in Materials*, 6:61, 2019.

Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in Neural Information Processing Systems*, 33:12449–12460, 2020.

Juan M. Barrios and Pablo E. Romero. Decision tree methods for predicting surface roughness in fused deposition modeling parts. *Materials*, 12(16), 2019. ISSN 1996-1944. doi: 10.3390/ma12162574. URL https://www.mdpi.com/1996-1944/12/16/2574.

Shivam Barwey, Varun Shankar, Venkatasubramanian Viswanathan, and Romit Maulik. Multi-scale graph neural network autoencoders for interpretable scientific machine learning, 2023.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

M. J. Bayarri, J. O. Berger, R. Paulo, J. Sacks, J. A. Cafeo, J. Cavendish, C. H. Lin, and J. Tu. A framework for validation of computer models. *Technometrics*, 49(2):138–154, 2007.

S. Bhattacharjee and K. Matouš. A nonlinear manifold-based reduced order model for multiscale analysis of heterogeneous hyperelastic materials. *Journal of Computational Physics*, 313:635–653, 2016. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2016.01.040. URL https://www.sciencedirect.com/science/article/pii/S0021999116001194.

Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling, 2019. URL https://arxiv.org/abs/1907.00481.

Jonas Biehler, Michael W. Gee, and Wolfgang A. Wall. Towards efficient uncertainty quantification in complex and large-scale biomechanical problems based on a bayesian multi-fidelity scheme. *Biomechanics and Modeling in Mechanobiology*, 14(3):489–513, September 2014. doi: 10.1007/s10237-014-0618-0. URL https://doi.org/10.1007/s10237-014-0618-0.

Nolan Black and Ahmad R. Najafi. Learning finite element convergence with the multi-fidelity graph neural network. *Computer Methods in Applied Mechanics and Engineering*, 397:115120, 2022. ISSN 0045-7825. URL https://www.sciencedirect.com/science/article/pii/S004578252200305X.

C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural network. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1613–1622, Lille, France, 07–09 Jul 2015. PMLR.

Frederic E Bock, Roland C Aydin, Christian J Cyron, Norbert Huber, Surya R Kalidindi, and Benjamin Klusemann. A review of the application of machine learning and data mining approaches in continuum materials mechanics. *Frontiers in Materials*, 6:110, 2019.

G.F. Bomarito, T.S. Townsend, K.M. Stewart, K.V. Esham, J.M. Emery, and J.D. Hochhalter. Development of interpretable, data-driven plasticity models with symbolic regression. *Computers &amp Structures*, 252:106557, August 2021. doi: 10.1016/j.compstruc.2021.106557. URL https://doi.org/10.1016/j.compstruc.2021.106557.

S. C. Brenner and L. R. Scott. *Finite Element Multigrid Methods*, pages 155–173. Springer New York, New York, NY, 2008. ISBN 978-0-387-75934-0. doi: 10.1007/978-0-387-75934-0_7.

Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017. doi: 10.1109/MSP.2017.2693418.

Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Wessel Bruinsma, Eric Perim, William Tebbutt, Scott Hosking, Arno Solin, and Richard Turner. Scalable exact inference in multi-output Gaussian processes. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1190–1201. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/bruinsma20a.html.

Jean-Nicolas Brunet, Andrea Mendizabal, Antoine Petit, Nicolas Golse, Eric Vibert, and Stéphane Cotin. Physics-based deep neural network for augmented reality during liver surgery. In Dinggang Shen, Tianming Liu, Terry M. Peters, Lawrence H. Staib, Caroline Essert, Sean Zhou, Pew-Thian Yap, and Ali Khan, editors, *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019*, pages 137–145, Cham, 2019. Springer International Publishing. ISBN 978-3-030-32254-0.

H. P. Bui, S. Tomar, H. Courtecuisse, S. Cotin, and S. P. A. Bordas. Real-time error control for surgical simulation. *IEEE Transactions on Biomedical Engineering*, 65(3):596–607, 2018. doi: 10.1109/TBME.2017.2695587.

Keith T Butler, Daniel W Davies, Hugh Cartwright, Olexandr Isayev, and Aron Walsh. Machine learning for molecular and materials science. *Nature*, 559(7715):547–555, 2018.

Chen Cai, Dingkang Wang, and Yusu Wang. Graph coarsening with neural networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=uxpzitPEooJ.

German Capuano and Julian J. Rimoli. Smart finite elements: A novel machine learning application. *Computer Methods in Applied Mechanics and Engineering*, 345:363–381, 2019. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2018.10.046. URL https://www.sciencedirect.com/science/article/pii/S0045782518305541.

B. P. Carlin and T. A. Louis. Bayes and empirical bayes methods for data analysis. *Statistics and Computing*, 1997. doi: https://doi.org/10.1023/A:1018577817064.

Eleni N. Chatzi, Andrew W. Smyth, and Sami F. Masri. Experimental application of on-line parametric identification for nonlinear hysteretic systems with model uncertainty. *Structural Safety*, 32(5):326–337, 2010. ISSN 0167-4730. doi: https://doi.org/10.1016/j.strusafe.2010.03.008.

URL https://www.sciencedirect.com/science/article/pii/S0167473010000275. Probabilistic Methods for Modeling, Simulation and Optimization of Engineering Structures under Uncertainty in honor of Jim Beck's 60th Birthday.

Alvin I Chen, Max L Balter, Timothy J Maguire, and Martin L Yarmush. Deep learning robotic guidance for autonomous vascular access. *Nature Machine Intelligence*, 2(2):104–115, 2020a.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1725–1735. PMLR, 13–18 Jul 2020b. URL https://proceedings.mlr.press/v119/chen20v.html.

HeeSun Choi, Cindy Crump, Christian Duriez, Asher Elmquist, Gregory Hager, David Han, Frank Hearl, Jessica Hodgins, Abhinandan Jain, Frederick Leve, et al. On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward. *Proceedings of the National Academy of Sciences*, 118(1):e1907856118, 2021.

F. Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

Kamal Choudhary, Brian DeCost, Chi Chen, Anubhav Jain, Francesca Tavazza, Ryan Cohn, Cheol Woo Park, Alok Choudhary, Ankit Agrawal, Simon JL Billinge, et al. Recent advances and applications of deep learning methods in materials science. *npj Computational Materials*, 8(1):1–26, 2022.

Özgün Çiçek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: learning dense volumetric segmentation from sparse annotation. In *International conference on medical image computing and computer-assisted intervention*, pages 424–432. Springer, 2016.

S. Cotin, H. Delingette, and N. Ayache. Real-time elastic deformations of soft tissues for surgery simulation. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):62–73, 1999. doi: 10.1109/2945.764872.

Stéphane Cotin, Hervé Delingette, and Nicholas Ayache. A Hybrid Elastic Model allowing Real-Time Cutting, Deformations and Force-Feedback for Surgery Training and Simulation. *The Visual Computer*, 16(8):437–452, 2000. doi: 10.1007/PL00007215. URL https://inria.hal.science/inria-00615105.

H. Courtecuisse, J. Allard, P. Kerfriden, S.P.A. Bordas, S. Cotin, and C. Duriez. Real-time simulation of contact and cutting of heterogeneous soft-tissues. *Medical Image Analysis*, 18 (2):394–410, 2014a. ISSN 1361-8415. doi: https://doi.org/10.1016/j.media.2013.11.001. URL https://www.sciencedirect.com/science/article/pii/S1361841513001692.

Hadrien Courtecuisse, Jérémie Allard, Pierre Kerfriden, Stéphane PA Bordas, Stéphane Cotin, and Christian Duriez. Real-time simulation of contact and cutting of heterogeneous soft-tissues. *Medical image analysis*, 18(2):394–410, 2014b.

Thomas Daniel, Fabien Casenave, Nissrine Akkari, and David Ryckelynck. Model order reduction assisted by deep neural networks (rom-net). *Advanced Modeling and Simulation in Engineering Sciences*, 7(1):1–27, 2020.

Marco De Vivo, Matteo Masetti, Giovanni Bottegoni, and Andrea Cavalli. Role of molecular dynamics and related methods in drug discovery. *Journal of Medicinal Chemistry*, 59(9): 4035–4061, 2016. doi: 10.1021/acs.jmedchem.5b01684. URL https://doi.org/10.1021/acs.jmedchem.5b01684. PMID: 26807648.

H. Delingette, S. Cotin, and N. Ayache. A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation. In *Proceedings Computer Animation 1999*, pages 70–81, 1999. doi: 10.1109/CA.1999.781200.

Cyrill Dennler, David E. Bauer, Anne-Gita Scheibler, José Spirig, Tobias Götschi, Philipp Fürnstahl, and Mazda Farshad. Augmented reality in the operating room: a clinical feasibility study. *BMC Musculoskeletal Disorders*, 22(1), May 2021. doi: 10.1186/s12891-021-04339-w. URL https://doi.org/10.1186/s12891-021-04339-w.

Saurabh Deshpande, Jakub Lengiewicz, and Stéphane P.A. Bordas. Probabilistic deep learning for real-time large deformation simulations. *Computer Methods in Applied Mechanics and Engineering*, 398:115307, 2022. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2022.115307. URL https://www.sciencedirect.com/science/article/pii/S004578252200411X.

Saurabh Deshpande, Stéphane Bordas, and Jakub Lengiewicz. MAgNET: A Graph U-Net Architecture for Mesh-Based Simulations [Supplementary data], March 2023a. URL https://doi.org/10.5281/zenodo.7784804.

Saurabh Deshpande, Stéphane P. A. Bordas, and Jakub Lengiewicz. Magnet: A graph u-net architecture for mesh-based simulations, 2023b. URL https://arxiv.org/abs/2211.00713.

Saurabh Deshpande, Raúl I. Sosa, Stéphane P. A. Bordas, and Jakub Lengiewicz. Convolution, aggregation and attention based deep neural networks for accelerating simulations in mechanics. *Frontiers in Materials*, 10, 2023c. doi: 10.3389/fmats.2023.1128954. URL https://doi.org/10.3389/fmats.2023.1128954.

Saurabh Deshpande, Raul Ian Sosa, Stéphane Bordas, and Jakub Lengiewicz. Convolution, aggregation and attention based deep neural networks for accelerating simulations in mechanics [Dataset], January 2023d. URL https://doi.org/10.5281/zenodo.7585319.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

J. V. Dillon, I. Langmore, D. Tran, E. Brevdo, S. Vasudevan, D. Moore, B. Patton, A. Alemi, M. D. Hoffman, and R. A. Saurous. Tensorflow distributions. *CoRR*, abs/1711.10604, 2017. URL http://arxiv.org/abs/1711.10604.

Chensen Ding, Hussein Rappel, and Tim Dodwell. Full-field order-reduced gaussian process emulators for nonlinear probabilistic mechanics. *Computer Methods in Applied Mechanics and Engineering*, 405:115855, 2023. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2022.115855. URL https://www.sciencedirect.com/science/article/pii/S0045782522008118.

Stefan Doll and Karl Schweizerhof. On the development of volumetric strain energy functions. *J. Appl. Mech.*, 67(1):17–21, 2000.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

O. Duerr, B. Sick, and E. Murina. *Probabilistic Deep Learning: With Python, Keras and TensorFlow Probability*. Manning Publications, 2020. ISBN 9781617296079. URL https://books.google.co.in/books?id=-bYCEAAAQBAJ.

George El Haber, Jonathan Viquerat, Aurelien Larcher, David Ryckelynck, Jose Alves, Aakash Patil, and Elie Hachem. Deep learning model to assist multiphysics conjugate problems. *Physics of Fluids*, 34(1):015131, 2022. doi: 10.1063/5.0077723. URL https://doi.org/10.1063/5.0077723.

Aflah Elouneg, Audrey Bertin, Quentin Lucot, Vincent Tissot, Emmanuelle Jacquet, Jérôme Chambert, and Arnaud Lejeune. In vivo skin anisotropy dataset from annular suction test. *Data in Brief*, 40:107835, 2022. ISSN 2352-3409. doi: https://doi.org/10.1016/j.dib.2022.107835. URL https://www.sciencedirect.com/science/article/pii/S2352340922000476.

C. Farhat and F. Roux. A method of finite element tearing and interconnecting and its parallel solution algorithm. *International Journal for Numerical Methods in Engineering*, 32(6):1205–1227, 1991. doi: https://doi.org/10.1002/nme.1620320604. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.1620320604.

Sofia Farina, Susanne Claus, Jack S Hale, Alexander Skupin, and Stéphane PA Bordas. A cut finite element method for spatially resolved energy metabolism models in complex neuro-cell morphologies with minimal remeshing. *Advanced Modeling and Simulation in Engineering Sciences*, 8:1–32, 2021. doi: https://doi.org/10.1186/s40323-021-00191-8.

Di Feng, Lars Rosenbaum, and Klaus Dietmayer. Towards safe autonomous driving: Capture uncertainty in the deep neural network for lidar 3d vehicle detection. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3266–3273, 2018. doi: 10.1109/ITSC.2018.8569814.

Moritz Flaschel, Siddhant Kumar, and Laura De Lorenzis. Unsupervised discovery of interpretable hyperelastic constitutive laws. *Computer Methods in Applied Mechanics and Engineering*, 381:113852, 2021. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2021.113852. URL https://www.sciencedirect.com/science/article/pii/S0045782521001894.

Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2000.

Stefania Fresca, Luca Dede', and Andrea Manzoni. A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized pdes. *Journal of Scientific Computing*, 87:1–36, 2021.

Stefania Fresca, Giorgio Gobat, Patrick Fedeli, Attilio Frangi, and Andrea Manzoni. Deep learning-based reduced order models for the real-time simulation of the nonlinear dynamics of microstructures. *International Journal for Numerical Methods in Engineering*, 123(20):4749–4777, 2022.

Richard A. Friesner. ab initio quantum chemistry: Methodology and applications. *Proceedings of the National Academy of Sciences*, 102(19):6648–6653, 2005. doi: 10.1073/pnas.0408036102. URL https://www.pnas.org/doi/abs/10.1073/pnas.0408036102.

Y. Gal. Uncertainty in deep learning. 2016.

Han Gao, Luning Sun, and Jian-Xun Wang. Phygeonet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain. *Journal of Computational Physics*, 428:110079, 2021. ISSN 0021-9991. URL https://www.sciencedirect.com/science/article/pii/S0021999120308536.

Han Gao, Matthew J. Zahr, and Jian-Xun Wang. Physics-informed graph neural galerkin networks: A unified framework for solving pde-governed forward and inverse problems.

Computer Methods in Applied Mechanics and Engineering, 390:114502, 2022. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2021.114502. URL https://www.sciencedirect.com/science/article/pii/S0045782521007076.

Hongyang Gao and Shuiwang Ji. Graph u-net, 2019. URL https://openreview.net/forum?id=HJePRoAct7.

T. Gawlikowski, C.R.N. Tassi, M Ali, L. Lee, M. Humt, J. Feng, A. Kruspe, R. Triebel, P. Jung, R. Roscher, M. Shahzad, W. Yang, R. Bamler, and X. X. Zhu. A survey of uncertainty in deep neural networks, 2021.

A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian data analysis*. Chapman & Hall/CRC Texts in Statistical Science. Chapman & Hall/CRC, 2003. ISBN 9781584883883.

Nicholas Geneva and Nicholas Zabaras. Transformers for modeling physical systems. *Neural Networks*, 146:272–289, February 2022. doi: 10.1016/j.neunet.2021.11.022. URL https://doi.org/10.1016/j.neunet.2021.11.022.

C. Geuzaine and J. Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79:1309 – 1331, 09 2009. doi: 10.1002/nme.2579.

Torkan Gholamalizadeh, Faezeh Moshfeghifar, Zachary Ferguson, Teseo Schneider, Daniele Panozzo, Sune Darkner, Masrour Makaremi, François Chan, Peter Lampel Søndergaard, and Kenny Erleben. Open-full-jaw: An open-access dataset and pipeline for finite element models of human jaw. *Computer Methods and Programs in Biomedicine*, 224:107009, 2022. ISSN 0169-2607. doi: https://doi.org/10.1016/j.cmpb.2022.107009. URL https://www.sciencedirect.com/science/article/pii/S0169260722003911.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL https://proceedings.mlr.press/v9/glorot10a.html.

O. Goury and C. Duriez. Fast, generic and reliable control and simulation of soft robots using model order reduction. *IEEE Transactions on Robotics*, 34(6):1565 – 1576, 2018a. doi: 10.1109/TRO.2018.2861900. URL https://hal.archives-ouvertes.fr/hal-01834483.

Olivier Goury and Christian Duriez. Fast, generic, and reliable control and simulation of soft robots using model order reduction. *IEEE Transactions on Robotics*, 34(6):1565–1576, 2018b.

Lorenzo Grassi, Enrico Schileo, Christelle Boichon, Marco Viceconti, and Fulvia Taddei. Comprehensive evaluation of pca-based finite element modelling of the human femur. *Medical engineering & physics*, 36, 08 2014. doi: 10.1016/j.medengphy.2014.06.021.

A. Graves. Practical variational inference for neural networks. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL https://proceedings.neurips.cc/paper/2011/file/7eb3c8be3d411e8ebfab08eba5f49632-Paper.pdf.

Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 481–490, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939738. URL https://doi.org/10.1145/2939672.2939738.

Ehsan Haghighat, Maziar Raissi, Adrian Moure, Hector Gomez, and Ruben Juanes. A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 379:113741, 2021. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2021.113741. URL https://www.sciencedirect.com/science/article/pii/S0045782521000773.

R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or. Meshcnn: A network with an edge. 38(4), 2019. ISSN 0730-0301. doi: 10.1145/3306346.3322959. URL https://doi.org/10.1145/3306346.3322959.

R. Hassanian, H. Myneni, Á. Helgadóttir, and M. Riedel. Deciphering the dynamics of distorted turbulent flows: Lagrangian particle tracking and chaos prediction through transformer-based deep learning models. *Physics of Fluids*, 35(7), July 2023. doi: 10.1063/5.0157897. URL https://doi.org/10.1063/5.0157897.

P. Hauseux, J.S. Hale, and S.P.A. Bordas. Accelerating Monte Carlo estimation with derivatives of high-level finite element models. *Computer Methods in Applied Mechanics and Engineering*, 318:917–936, 2017a.

P. Hauseux, J. S. Hale, S. Cotin, and S.P.A. Bordas. Quantifying the uncertainty in a hyperelastic soft tissue model with stochastic parameters. *Applied Mathematical Modelling*, 62:86–102, 2018. ISSN 0307-904X. doi: https://doi.org/10.1016/j.apm.2018.04.021. URL https://www.sciencedirect.com/science/article/pii/S0307904X18302063.

Paul Hauseux, Jack S. Hale, and Stéphane P.A. Bordas. Accelerating monte carlo estimation with derivatives of high-level finite element models. *Computer Methods in Applied Mechanics and Engineering*, 318:917–936, May 2017b. doi: 10.1016/j.cma.2017.01.041. URL https://doi.org/10.1016/j.cma.2017.01.041.

Paul Hauseux, Thanh-Tung Nguyen, Alberto Ambrosetti, Katerine Saleme Ruiz, Stéphane P. A. Bordas, and Alexandre Tkatchenko. From quantum to continuum mechanics in the delamination of atomically-thin layers from substrates. *Nature Communications*, 11(1), April 2020. doi: 10.1038/s41467-020-15480-w. URL https://doi.org/10.1038/s41467-020-15480-w.

J. He and J. Xu. Mgnet: A unified framework of multigrid and convolutional neural network. *Science china mathematics*, 62(7):1331–1354, 2019.

Alexander Henkes, Henning Wessels, and Rolf Mahnken. Physics informed neural networks for continuum micromechanics. *Computer Methods in Applied Mechanics and Engineering*, 393:114790, 2022. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2022.114790. URL https://www.sciencedirect.com/science/article/pii/S0045782522001268.

Romain Hennequin, Anis Khlif, Felix Voituret, and Manuel Moussallam. Spleeter: a fast and efficient music source separation tool with pre-trained models. *Journal of Open Source Software*, 5:2154, 06 2020. doi: 10.21105/joss.02154.

G.E. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science (New York, N.Y.)*, 313:504–7, 08 2006. doi: 10.1126/science.1127647.

Gerhard A Holzapfel. Nonlinear solid mechanics: a continuum approach for engineering science, 2002.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(91)90009-T. URL https://www.sciencedirect.com/science/article/pii/089360809190009T.

Daniel Z. Huang, Kailai Xu, Charbel Farhat, and Eric Darve. Learning constitutive relations from indirect observations using deep neural networks. *Journal of Computational Physics*, 416:109491, 2020. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2020.109491. URL https://www.sciencedirect.com/science/article/pii/S0021999120302655.

Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods. *Machine Learning*, 110(3):457–506, March 2021. doi: 10.1007/s10994-021-05946-3. URL https://doi.org/10.1007/s10994-021-05946-3.

A. Courville I. Goodfellow, Y. Bengio. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. http://www.deeplearningbook.org.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 02 2015.

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, Olivier J Henaff, Matthew Botvinick, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver IO: A general architecture for structured inputs & outputs. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=fILj7WpI-g.

Chintan Jansari, Stéphane P.A. Bordas, and Elena Atroshchenko. Design of metamaterial-based heat manipulators by isogeometric shape optimization. *International Journal of Heat and Mass Transfer*, 196:123201, 2022. ISSN 0017-9310. doi: https://doi.org/10.1016/j.ijheatmasstransfer.2022.123201. URL https://www.sciencedirect.com/science/article/pii/S0017931022006718.

Dipendra Jha, Logan Ward, Arindam Paul, Wei-keng Liao, Alok Choudhary, Chris Wolverton, and Ankit Agrawal. Elemnet: Deep learning the chemistry of materials from only elemental composition. *Scientific reports*, 8(1):1–13, 2018.

Carl Jidling. Strain field modelling using gaussian processes, 2017. ISSN 1401-5757.

Stian F Johnsen, Zeike A Taylor, Matthew J Clarkson, John Hipwell, Marc Modat, Bjoern Eiben, Lianghao Han, Yipeng Hu, Thomy Mertzanidou, David J Hawkes, et al. Niftysim: A gpu-based nonlinear finite element package for simulation of soft tissue biomechanics. *International journal of computer assisted radiology and surgery*, 10:1077–1095, 2015.

A. Kendall and Y. Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5580–5590, Red Hook, NY, USA, 2017a. Curran Associates Inc. ISBN 9781510860964.

A. Kendall and Y. Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5580–5590, Red Hook, NY, USA, 2017b. Curran Associates Inc. ISBN 9781510860964.

M.C. Kennedy and A. O'Hagan. Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(3):425–464, 2001. ISSN 1467-9868.

P. Kerfriden, P. Gosselet, S. Adhikari, and S.P.A. Bordas. Bridging proper orthogonal decomposition methods and augmented newton–krylov algorithms: An adaptive model order reduction for highly nonlinear mechanical problems. *Computer Methods in Applied Mechanics and Engineering*, 200(5):850–866, 2011a. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2010.10.009. URL https://www.sciencedirect.com/science/article/pii/S0045782510002872.

P. Kerfriden, P. Gosselet, S. Adhikari, and S.P.A. Bordas. Bridging proper orthogonal decomposition methods and augmented newton–krylov algorithms: An adaptive model order reduction for highly nonlinear mechanical problems. *Computer Methods in Applied Mechanics and Engineering*, 200(5):850–866, 2011b. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2010.10.009. URL https://www.sciencedirect.com/science/article/pii/S0045782510002872.

P. Kerfriden, O. Goury, T. Rabczuk, and S. Bordas. A partitioned model order reduction approach to rationalise computational expenses in multiscale fracture mechanics. *Computer Methods Appl Mech Eng*, pages 169–188, 08 2012. doi: https://doi.org/10.1016/j.cma.2012.12.004.

Gaëtan Kerschen, J.-C Golinval, ALEXANDER VAKAKIS, and LAWRENCE BERGMAN. The method of proper orthogonal decomposition for dynamical characterization and order reduction of mechanical systems: An overview. *Nonlinear Dynamics*, 41:147–169, 08 2005. doi: 10.1007/s11071-005-2803-2.

Moon ki Choi, Marco Pasetto, Zhaoxiang Shen, Ellad B. Tadmor, and David Kamensky. Atomistically-informed continuum modeling and isogeometric analysis of 2d materials over holey substrates. *Journal of the Mechanics and Physics of Solids*, 170:105100, 2023. ISSN 0022-5096. doi: https://doi.org/10.1016/j.jmps.2022.105100. URL https://www.sciencedirect.com/science/article/pii/S0022509622002770.

Yowhan Kim, Samarth Mishra, SouYoung Jin, Rameswar Panda, Hilde Kuehne, Leonid Karlinsky, Venkatesh Saligrama, Kate Saenko, Aude Oliva, and Rogerio Feris. How transferable are video representations based on synthetic data? In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL https://openreview.net/forum?id=lRUCfzs5Hzg.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

D. P. Kingma, T. Salimans, and M. Welling. Variational dropout and the local reparameterization trick. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/bc7316929fe1545bf0b98d114ee3ecb8-Paper.pdf.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv*, 2014. doi: 10.48550/ARXIV.1412.6980. URL https://arxiv.org/abs/1412.6980.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

T. Kirchdoerfer and M. Ortiz. Data-driven computational mechanics. *Computer Methods in Applied Mechanics and Engineering*, 304:81–101, 2016. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2016.02.001. URL https://www.sciencedirect.com/science/article/pii/S0045782516300238.

Armen Der Kiureghian and Ove Ditlevsen. Aleatory or epistemic? does it matter? *Structural Safety*, 31(2):105–112, March 2009. doi: 10.1016/j.strusafe.2008.06.020. URL https://doi.org/10.1016/j.strusafe.2008.06.020.

Dominik K. Klein, Rogelio Ortigosa, Jesús Martínez-Frutos, and Oliver Weeger. Finite electroelasticity with physics-augmented neural networks. *Computer Methods in Applied Mechanics and Engineering*, 400:115501, 2022. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2022.115501. URL https://www.sciencedirect.com/science/article/pii/S004578252200514X.

Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021. doi: 10.1073/pnas.2101784118. URL https://www.pnas.org/doi/abs/10.1073/pnas.2101784118.

J. Korelc. Multi-language and multi-environment generation of nonlinear finite element codes. *Engineering with Computers*, 18:312–327, 01 2002. doi: 10.1007/s003660200028.

P. S. Koutsourelakis. A multi-resolution, non-parametric, Bayesian framework for identification of spatially-varying model parameters. *Journal of Computational Physics*, 228(17):6184–6211, 2009. ISSN 0021-9991.

V. Krokos, V. Bui Xuan, S. P. A. Bordas, P. Young, and P. Kerfriden. A bayesian multiscale cnn framework to predict local stress fields in structures with microscale features. *Computational Mechanics*, 69:733–766, 05 2022a. doi: 10.1007/s00466-021-02112-3.

Vasilis Krokos, Stéphane P. A. Bordas, and Pierre Kerfriden. A graph-based probabilistic geometric deep learning framework with online physics-based corrections to predict the criticality of defects in porous materials, 2022b. URL https://arxiv.org/abs/2205.06562.

I.E. Lagaris, A. Likas, and D.I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998. doi: 10.1109/72.712178.

I.E. Lagaris, A.C. Likas, and D.G. Papageorgiou. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, 11(5):1041–1049, 2000. doi: 10.1109/72.870037.

T. Lavigne, A. Mazier, A. Perney, S.P.A. Bordas, F. Hild, and J. Lengiewicz. Digital volume correlation for large deformations of soft tissues: Pipeline and proof of concept for the application to breast ex vivo deformations. *Journal of the Mechanical Behavior of Biomedical Materials*, page 105490, 2022. ISSN 1751-6161. doi: https://doi.org/10.1016/j.jmbbm.2022.105490. URL https://www.sciencedirect.com/science/article/pii/S1751616122003952.

T. Lavigne, S.P.A. Bordas, and J. Lengiewicz. Identification of material parameters and traction field for soft bodies in contact. *Computer Methods in Applied Mechanics and Engineering*, 406:115889, 2023. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2023.115889. URL https://www.sciencedirect.com/science/article/pii/S0045782523000129.

Kody Law, Andrew Stuart, and Kostas Zygalakis. Data assimilation. *Cham, Switzerland: Springer*, 214:52, 2015.

Thi-Thu-Huong Le, Hyoeun Kang, and Howon Kim. Towards incompressible laminar flow estimation based on interpolated feature generation and deep learning. *Sustainability*, 14(19), 2022. ISSN 2071-1050. doi: 10.3390/su141911996. URL https://www.mdpi.com/2071-1050/14/19/11996.

Tuan Anh Le, Atilim Giineş Baydin, Robert Zinkov, and Frank Wood. Using synthetic data to train neural networks is model-based reasoning. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 3514–3521. IEEE, 2017.

Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling, 2019. URL https://arxiv.org/abs/1904.08082.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=c8P9NQVtmnO.

Cheng-Yuan Liou, Wei-Chen Cheng, Jiun-Wei Liou, and Daw-Ran Liou. Autoencoder for words. *Neurocomputing*, 139:84–96, 2014.

Wing Kam Liu, George Karniadakis, Shaoqiang Tang, and Julien Yvonnet. A computational mechanics special issue on: data-driven modeling and simulation—theory, methods, and applications. *Computational Mechanics*, 64(2):275–277, June 2019. doi: 10.1007/s00466-019-01741-z. URL https://doi.org/10.1007/s00466-019-01741-z.

Yue Liu, Tianlu Zhao, Wangwei Ju, and Siqi Shi. Materials discovery and design using machine learning. *Journal of Materiomics*, 3(3):159–177, 2017. ISSN 2352-8478. doi: https://doi.org/10.1016/j.jmat.2017.08.002. URL https://www.sciencedirect.com/science/article/pii/S2352847817300515. High-throughput Experimental and Modeling Research toward Advanced Batteries.

D. Lorente, F. Martínez-Martínez, M.J. Rupérez, M.A. Lago, M. Martínez-Sober, P. Escandell-Montero, J.M. Martínez-Martínez, S. Martínez-Sanchis, A.J. Serrano-López, C. Monserrat, and J.D. Martín-Guerrero. A framework for modelling the biomechanical behaviour of the human liver during breathing in real time using machine learning. *Expert Systems with Applications*, 71:342–357, 2017. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2016.11.037. URL https://www.sciencedirect.com/science/article/pii/S0957417416306728.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv*, 2017. doi: 10.48550/ARXIV.1711.05101. URL https://arxiv.org/abs/1711.05101.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3:218–229, 06 2021. doi: 10.1038/s42256-021-00302-5.

Enxhell Luzhnica, Ben Day, and Pietro Lio'. Clique pooling for graph classification, 2019. URL https://arxiv.org/abs/1904.00374.

Zhiping Mao, Ameya D. Jagtap, and George Em Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2019.112789. URL https://www.sciencedirect.com/science/article/pii/S0045782519306814.

D. Marinkovic and M. Zehn. Survey of finite element method-based real-time simulations. *Applied Sciences*, 9(14), 2019. ISSN 2076-3417. doi: 10.3390/app9142775. URL https://www.mdpi.com/2076-3417/9/14/2775.

Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *Artificial Neural Networks and Machine Learning–ICANN 2011: 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part I 21*, pages 52–59. Springer, 2011.

H.G. Matthies. Stochastic finite elements: Computational approaches to stochastic partial differential equations. *ZAMM*, 88(11):849–873, November 2008. doi: 10.1002/zamm.200800095. URL https://doi.org/10.1002/zamm.200800095.

Arnaud Mazier and Stéphane P. A. Bordas. Breast simulation pipeline: from medical imaging to patient-specific simulations. 2023.

Arnaud Mazier, Sophie Ribes, Benjamin Gilles, and Stéphane P.A. Bordas. A rigged model of the breast for preoperative surgical planning. *Journal of Biomechanics*, 128:110645, 2021. ISSN 0021-9290. doi: https://doi.org/10.1016/j.jbiomech.2021.110645. URL https://www.sciencedirect.com/science/article/pii/S0021929021004140.

Arnaud Mazier, Sidaty El Hadramy, Jean-Nicolas Brunet, Jack S. Hale, Stéphane Cotin, and Stéphane P. A. Bordas. Sonics: Develop intuition on biomechanical systems through interactive error controlled simulations. 2022.

R. McAllister, Y. Gal, A. Kendall, M. v. d. Wilk, A. Shah, R. Cipolla, and A. Weller. Concrete problems for autonomous vehicle safety: Advantages of bayesian deep learning. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 4745–4753, 2017. doi: 10.24963/ijcai.2017/661. URL https://doi.org/10.24963/ijcai.2017/661.

K. McFall and J. Mahan. Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions. *IEEE transactions on neural networks*, 20(8):1221—1233, August 2009. ISSN 1045-9227. doi: 10.1109/tnn.2009.2020735. URL https://doi.org/10.1109/TNN.2009.2020735.

A. Mendizabal, P. Márquez-Neila, and S. Cotin. Simulation of hyperelastic materials in real-time using deep learning. *Medical Image Analysis*, 59:101569, 10 2019a. doi: 10.1016/j.media.2019.101569.

A. Mendizabal, P. Márquez-Neila, and S. Cotin. Simulation of hyperelastic materials in real-time using deep learning. *Medical Image Analysis*, 59:101569, 10 2019b. doi: 10.1016/j.media.2019.101569.

Jaber Rezaei Mianroodi, Nima H Siboni, and Dierk Raabe. Teaching solid mechanics to artificial intelligence—a fast solver for heterogeneous materials. *Npj Computational Materials*, 7(1):1–10, 2021.

Francisco J. Montáns, Francisco Chinesta, Rafael Gómez-Bombarelli, and J. Nathan Kutz. Data-driven modeling and learning in science and engineering. *Comptes Rendus Mécanique*, 347(11):845–855, 2019. ISSN 1631-0721. doi: https://doi.org/10.1016/j.crme.2019.11.009. URL https://www.sciencedirect.com/science/article/pii/S1631072119301809. Data-Based Engineering Science and Technology.

M. Mozaffar, R. Bostanabad, W. Chen, K. Ehmann, J. Cao, and M. A. Bessa. Deep learning predicts path-dependent plasticity. *Proceedings of the National Academy of Sciences*, 116 (52):26414–26420, December 2019. doi: 10.1073/pnas.1911815116. URL https://doi.org/10.1073/pnas.1911815116.

Takaaki Murata, Kai Fukami, and Koji Fukagata. Nonlinear mode decomposition with convolutional neural networks for fluid dynamics. *Journal of Fluid Mechanics*, 882:A13, 2020.

R.M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag, Berlin, Heidelberg, 1996. ISBN 0387947248.

Stefanos Nikolopoulos, Ioannis Kalogeris, and Vissarion Papadopoulos. Non-intrusive surrogate modeling for parametrized time-dependent partial differential equations using convolutional autoencoders. *Engineering Applications of Artificial Intelligence*, 109:104652, March 2022. doi: 10.1016/j.engappai.2021.104652. URL https://doi.org/10.1016/j.engappai.2021.104652.

S. Niroomandi, I. Alfaro, E. Cueto, and F. Chinesta. Model order reduction for hyperelastic materials. *International Journal for Numerical Methods in Engineering*, 81:1180 – 1206, 01 2009. doi: 10.1002/nme.2733.

Octavi Obiols-Sales, Abhinav Vishnu, Nicholas Malaya, and Aparna Chandramowliswharan. Cfdnet: A deep learning-based accelerator for fluid simulations. ICS '20, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379830. doi: 10.1145/3392717.3392772. URL https://doi.org/10.1145/3392717.3392772.

Alban Odot, Ryadh Haferssas, and Stéphane Cotin. Deepphysics: a physics aware deep learning framework for real-time simulation, 2021. URL https://arxiv.org/abs/2109.09491.

R. W. Ogden. *Non-linear elastic deformations*. Dover Publications, 2005.

Atsuya Oishi and Genki Yagawa. Computational mechanics enhanced by deep learning. *Computer Methods in Applied Mechanics and Engineering*, 327:327–351, 2017. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2017.08.040. URL https://www.sciencedirect.com/science/article/pii/S0045782517306199. Advances in Computational Mechanics and Scientific Computation—the Cutting Edge.

Pranshu Pant, Ruchit Doshi, Pranav Bahl, and Amir Barati Farimani. Deep learning for reduced order modelling and efficient temporal evolution of fluid simulations. *Physics of Fluids*, 33(10), 2021.

Paris Papavasileiou, Eleni D. Koronaki, Gabriele Pozzetti, Martin Kathrein, Christoph Czettl, Andreas G. Boudouvis, T.J. Mountziaris, and Stéphane P.A. Bordas. An efficient chemistry-enhanced cfd model for the investigation of the rate-limiting mechanisms in industrial chemical vapor deposition reactors. *Chemical Engineering Research and Design*, 186:314–325, 2022. ISSN 0263-8762. doi: https://doi.org/10.1016/j.cherd.2022.08.005. URL https://www.sciencedirect.com/science/article/pii/S0263876222004087.

Paris Papavasileiou, Eleni D. Koronaki, Gabriele Pozzetti, Martin Kathrein, Christoph Czettl, Andreas G. Boudouvis, and Stéphane P.A. Bordas. Equation-based and data-driven modeling strategies for industrial coating processes. *Computers in Industry*, 149: 103938, 2023. ISSN 0166-3615. doi: https://doi.org/10.1016/j.compind.2023.103938. URL https://www.sciencedirect.com/science/article/pii/S016636152300088X.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct): 2825–2830, 2011.

Oscar J. Pellicer-Valero, María José Rupérez, Sandra Martínez-Sanchis, and José D. Martín-Guerrero. Real-time biomechanical modeling of the liver using machine learning models trained on finite element method simulations. *Expert Systems with Applications*, 143:113083, 2020. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2019.113083. URL https://www.sciencedirect.com/science/article/pii/S0957417419308000.

T. Pfaff, M. Fortunato, A.S. Gonzalez, and P. Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=roNqYL0_XP.

Micha Pfeiffer, Carina Riediger, Jürgen Weitz, and Stefanie Speidel. Learning soft tissue behavior of organs for surgical navigation with convolutional neural networks. *International Journal of Computer Assisted Radiology and Surgery*, 14(7):1147–1155, April 2019. doi: 10.1007/s11548-019-01965-7. URL https://doi.org/10.1007/s11548-019-01965-7.

Benoît Piranda, Paweł Chodkiewicz, Paweł Hołobut, Stéphane P. A. Bordas, Julien Bourgeois, and Jakub Lengiewicz. Distributed prediction of unsafe reconfiguration scenarios of modular robotic programmable matter. *IEEE Transactions on Robotics*, 37(6):2226–2233, 2021. doi: 10.1109/TRO.2021.3074085.

Rosalie Plantefeve, Igor Peterlik, Nazim Haouchine, and Stéphane Cotin. Patient-specific biomechanical modeling for guidance during minimally-invasive hepatic surgery. *Annals of biomedical engineering*, 44:139–153, 2016.

Dario Poloni, Daniele Oboe, Claudio Sbarufatti, and Marco Giglio. Towards a stochastic inverse finite element method: A gaussian process strain extrapolation. *Mechanical Systems and Signal Processing*, 189:110056, 2023. ISSN 0888-3270. doi: https://doi.org/10.1016/j.ymssp.2022.110056. URL https://www.sciencedirect.com/science/article/pii/S0888327022011244.

M. Raissi, P. Perdikaris, and G.E. Karniadakis. 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, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

Chengping Rao and Yang Liu. Three-dimensional convolutional neural network (3d-cnn) for heterogeneous material homogenization. *Computational Materials Science*, 184:109850, 11 2020. doi: 10.1016/j.commatsci.2020.109850.

H. Rappel, L. A. A. Beex, L. Noels, and S. P. A. Bordas. Identifying elastoplastic parameters with Bayes' theorem considering output error, input error and model uncertainty. *Probabilistic Engineering Mechanics*, 55:28–41, 2018.

H. Rappel, L. Wu, L. Noels, and L. A. A. Beex. A Bayesian Framework to Identify Random Parameter Fields Based on the Copula Theorem and Gaussian Fields: Application to Polycrystalline Materials. *Journal of Applied Mechanics*, 86(12):121009, 10 2019.

H. Rappel, L. A. A. Beex, J. S. Hale, L. Noels, and S. P. A Bordas. A tutorial on bayesian inference to identify material parameters in solid mechanics. *Archives of Computational Methods in Engineering*, 27, 2020. doi: https://doi.org/10.1007/s11831-018-09311-x.

Hussein Rappel, Mark Girolami, and Lars A.A. Beex. Intercorrelated random fields with bounds and the bayesian identification of their parameters: Application to linear elastic struts and fibers. *International Journal for Numerical Methods in Engineering*, 123(15):3418–3463, 2022.

C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.

Carl Edward Rasmussen, Christopher K Williams, et al. Gaussian processes for machine learning, vol. 1, 2006.

G Thippa Reddy, M Praveen Kumar Reddy, Kuruva Lakshmanna, Rajesh Kaluri, Dharmendra Singh Rajput, Gautam Srivastava, and Thar Baker. Analysis of dimensionality reduction techniques on big data. *Ieee Access*, 8:54776–54788, 2020.

Xinlei Ren, Xu Zhang, Lianwu Chen, Xiguang Zheng, Chen Zhang, Liang Guo, and Bing Yu. A causal u-net based neural beamforming network for real-time multi-channel speech enhancement. pages 1832–1836, 08 2021. doi: 10.21437/Interspeech.2021-1457.

S. Rogers and M. Girolami. *A first course in machine learning, second edition*. Chapman & Hall/CRC, 2nd edition, 2016. ISBN 1498738486, 9781498738484.

O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. URL http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a. (available on arXiv:1505.04597 [cs.CV]).

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

James C Rosser Jr, Michinori Murayama, and Nick H Gabriel. Minimally invasive surgical training solutions for the twenty-first century. *Surgical Clinics of North America*, 80(5): 1607–1624, 2000.

Samuel H. Rudy, Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3(4):e1602614, 2017. doi: 10.1126/sciadv.1602614. URL https://www.science.org/doi/abs/10.1126/sciadv.1602614.

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

Gundula Runge, Mats Wiese, and Annika Raatz. Fem-based training of artificial neural networks for modular soft robots. In *2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 385–392, 2017. doi: 10.1109/ROBIO.2017.8324448.

Matthias Rupp, Alexandre Tkatchenko, Klaus-Robert Müller, and O. Anatole von Lilienfeld. Fast and accurate modeling of molecular atomization energies with machine learning. *Phys. Rev. Lett.*, 108:058301, Jan 2012. doi: 10.1103/PhysRevLett.108.058301. URL https://link.aps.org/doi/10.1103/PhysRevLett.108.058301.

Daniela Rus and Michael T Tolley. Design, fabrication and control of soft robots. *Nature*, 521 (7553):467–475, 2015.

E. Samaniego, C. Anitescu, S. Goswami, V.M. Nguyen-Thanh, H. Guo, K. Hamdia, X. Zhuang, and T. Rabczuk. An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications. *Computer Methods in Applied Mechanics and Engineering*, 362:112790, 2020. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2019.112790. URL https://www.sciencedirect.com/science/article/pii/S0045782519306826.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W Battaglia. Learning to simulate complex physics with graph networks. *Learning to simulate complex physics with graph networks*, page 8459 – 8468, 2020. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85094802982&partnerID=40&md5=57a515ee79e6915a2266fa3f2bc4870c. Cited by: 53.

Carlos Sánchez-Sánchez and Dario Izzo. Real-time optimal control via deep neural networks: Study on landing problems. *Journal of Guidance, Control, and Dynamics*, 41(5):1122–1135, May 2018. doi: 10.2514/1.g002357. URL https://doi.org/10.2514/1.g002357.

Sudip K. Sarker and Charles Vincent. Errors in surgery. *International Journal of Surgery*, 3(1):75–81, 2005. ISSN 1743-9191. doi: https://doi.org/10.1016/j.ijsu.2005.04.003. URL https://www.sciencedirect.com/science/article/pii/S1743919105000257.

Gabriel R Schleder, Antonio CM Padilha, Carlos Mera Acosta, Marcio Costa, and Adalberto Fazzio. From DFT to machine learning: recent approaches to materials science–a review. *Journal of Physics: Materials*, 2(3):032001, 2019.

Jonathan Schmidt, Mário RG Marques, Silvana Botti, and Miguel AL Marques. Recent advances and applications of machine learning in solid-state materials science. *npj Computational Materials*, 5(1):1–36, 2019.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

Kristof Schütt, Pieter-Jan Kindermans, Huziel Enoc Sauceda Felix, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/303ed4c69846ab36c2904d3ba8573050-Paper.pdf.

Kristof T Schütt, Farhad Arbabzadah, Stefan Chmiela, Klaus R Müller, and Alexandre Tkatchenko. Quantum-chemical insights from deep tensor neural networks. *Nature communications*, 8(1):1–8, 2017.

Sina Shafaei, Stefan Kugele, Mohd Hafeez Osman, and Alois Knoll. Uncertainty in machine learning: A safety perspective on autonomous driving. In *Developments in Language Theory*, pages 458–464. Springer International Publishing, 2018. doi: 10.1007/978-3-319-99229-7_39. URL https://doi.org/10.1007/978-3-319-99229-7_39.

Alok Sharma, Edwin Vans, Daichi Shigemizu, Keith A Boroevich, and Tatsuhiko Tsunoda. Deepinsight: A methodology to transform a non-image data to an image for convolution neural network architecture. *Scientific reports*, 9(1):11399, 2019.

Krushna Shinde, Vincent Itier, José Mennesson, Dmytro Vasiukov, and Modesar Shakoor. Dimensionality reduction through convolutional autoencoders for fracture patterns prediction. *Applied Mathematical Modelling*, 114:94–113, 2023. ISSN 0307-904X. doi: https://doi.org/10.1016/j.apm.2022.09.034. URL https://www.sciencedirect.com/science/article/pii/S0307904X22004541.

Ajay Shrestha and Ausif Mahmood. Review of deep learning algorithms and architectures. *IEEE Access*, 7:53040–53065, 2019. doi: 10.1109/ACCESS.2019.2912200.

JC Simo and RL Taylor. Penalty function formulations for incompressible nonlinear elastostatics. *Computer Methods in Applied Mechanics and Engineering*, 35(1):107–118, 1982.

Laurent Stainier, Adrien Leygue, and Michael Ortiz. Model-free data-driven methods in mechanics: material data identification and solvers. *Computational Mechanics*, 64(2): 381–393, jun 2019. doi: 10.1007/s00466-019-01731-1. URL https://doi.org/10.1007%2Fs00466-019-01731-1.

George Stefanou. The stochastic finite element method: Past, present and future. *Computer Methods in Applied Mechanics and Engineering*, 198(9-12):1031–1051, February 2009. doi: 10.1016/j.cma.2008.11.007. URL https://doi.org/10.1016/j.cma.2008.11.007.

Sebastian Strönisch, Marcus Meyer, and Christoph Lehmann. Flow field prediction on large variable sized 2d point clouds with graph convolution. In *Proceedings of the Platform for Advanced Scientific Computing Conference*, PASC '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450394109. doi: 10.1145/3539781.3539789. URL https://doi.org/10.1145/3539781.3539789.

Pratik Suchde and Jörg Kuhnert. A meshfree generalized finite difference method for surface pdes. *Computers & Mathematics with Applications*, 78(8):2789–2805, 2019. ISSN 0898-1221. doi: https://doi.org/10.1016/j.camwa.2019.04.030. URL https://www.sciencedirect.com/science/article/pii/S0898122119302469.

Hugo Talbot, Federico Spadoni, Christian Duriez, Maxime Sermesant, Mark O'Neill, Pierre Jaïs, Stéphane Cotin, and Hervé Delingette. Interactive training system for interventional electrocardiology procedures. *Medical Image Analysis*, 35:225–237, 2017.

Prakash Thakolkaran, Akshay Joshi, Yiwen Zheng, Moritz Flaschel, Laura De Lorenzis, and Siddhant Kumar. Nn-euclid: Deep-learning hyperelasticity without stress data. *Journal of the Mechanics and Physics of Solids*, 169:105076, 2022. ISSN 0022-5096. doi: https://doi.org/10.1016/j.jmps.2022.105076. URL https://www.sciencedirect.com/science/article/pii/S0022509622002538.

Tijmen Tieleman, Geoffrey Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2): 26–31, 2012.

Oliver T. Unke, Stefan Chmiela, Huziel E. Sauceda, Michael Gastegger, Igor Poltavsky, Kristof T. Schütt, Alexandre Tkatchenko, and Klaus-Robert Müller. Machine learning force fields. *Chemical Reviews*, 121(16):10142–10186, March 2021. doi: 10.1021/acs.chemrev.0c01111. URL https://doi.org/10.1021/acs.chemrev.0c01111.

Stephane Urcun, Pierre-Yves Rohan, Wafa Skalli, Stéphane Bordas, and Giuseppe Sciumè. Digital twinning of cellular capsule technology: Emerging outcomes from the perspective of porous media mechanics. *PLOS ONE*, 16:e0254512, 07 2021. doi: 10.1371/journal.pone.0254512.

S. Varrette, P. Bouvry, H. Cartiaux, and F. Georgatos. Management of an academic hpc cluster: The ul experience. 07 2014. doi: 10.1109/HPCSim.2014.6903792.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017. URL https://arxiv.org/abs/1710.10903.

S. Vijayaraghavan, L. Wu, L. Noels, S. P. A. Bordas, S. Natarajan, and L. A. A. Beex. Neural-network acceleration of projection-based model-order-reduction for finite plasticity: Application to rves, 2021a. URL https://arxiv.org/abs/2109.07747.

S. Vijayaraghavan, L. Wu, L. Noels, S. P. A. Bordas, S. Natarajan, and L. A. A. Beex. Neural-network acceleration of projection-based model-order-reduction for finite plasticity: Application to RVEs. *arXiv*, 2021b. doi: 10.48550/ARXIV.2109.07747. URL https://arxiv.org/abs/2109.07747.

M. Vladimirova, J. Verbeek, P. Mesejo, and J. Arbel. Understanding priors in Bayesian neural networks at the unit level. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6458–6467. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/vladimirova19a.html.

Nikolaos N. Vlassis, Ran Ma, and WaiChing Sun. Geometric deep learning for computational mechanics part i: anisotropic hyperelasticity. *Computer Methods in Applied Mechanics and Engineering*, 371:113299, 2020. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2020.113299. URL https://www.sciencedirect.com/science/article/pii/S0045782520304849.

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

F. Wang, A. Eljarrat, J. Müller, T. Henninen, R. Erni, and C. Koch. Multi-resolution convolutional neural networks for inverse problems. *Scientific Reports*, 10:5730, 03 2020a. doi: 10.1038/s41598-020-62484-z.

F. Wang, A. Eljarrat, J. Müller, T. Henninen, R. Erni, and C. Koch. Multi-resolution convolutional neural networks for inverse problems. *Scientific Reports*, 10:5730, 03 2020b. doi: 10.1038/s41598-020-62484-z.

Yasi Wang, Hongxun Yao, and Sicheng Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, 2016.

Zilong Wang and Young-Jin Cha. Unsupervised deep learning approach using a deep auto-encoder with a one-class support vector machine to detect damage. *Structural Health Monitoring*, 20(1):406–425, 2021. doi: 10.1177/1475921720934051. URL https://doi.org/10.1177/1475921720934051.

Asiri Umenga Weerasuriya, Xuelin Zhang, Bin Lu, Kam Tim Tse, and CH Liu. A gaussian process-based emulator for modeling pedestrian-level wind field. *Building and Environment*, 188:107500, 2021.

Gege Wen, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, and Sally M. Benson. U-fno—an enhanced fourier neural operator-based deep-learning model for multiphase flow. *Advances in Water Resources*, 163:104180, 2022. ISSN 0309-1708. doi: https://doi.org/10.1016/j.advwatres.2022.104180. URL https://www.sciencedirect.com/science/article/pii/S0309170822000562.

Y. Wen, P. Vicol, J. Ba, D. Tran, and R. Grosse. Flipout: Efficient pseudo-independent weight perturbations on mini-batches, 2018.

D Wirtz, N Karajan, and B Haasdonk. Surrogate modeling of multiscale models using kernel methods. *International Journal for Numerical Methods in Engineering*, 101(1):1–28, 2015.

J. Wu, R. Westermann, and C. Dick. A survey of physically based simulation of cuts in deformable bodies. *Computer Graphics Forum*, 34, 03 2015. doi: 10.1111/cgf.12528.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021. doi: 10.1109/TNNLS.2020.2978386.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR, 2015.

Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition, 2018. URL https://arxiv.org/abs/1801.07455.

Andriy Zakutayev, Nick Wunder, Marcus Schwarting, John D Perkins, Robert White, Kristin Munch, William Tumas, and Caleb Phillips. An open experimental database for exploring inorganic materials. *Scientific data*, 5(1):1–12, 2018.

Milad Zeraatpisheh, Stephane P.A. Bordas, and Lars A.A. Beex. Bayesian model uncertainty quantification for hyperelastic soft tissue models. *Data-Centric Engineering*, 2:e9, 2021. doi: 10.1017/dce.2021.9.

Wenbo Zhang, David S. Li, Tan Bui-Thanh, and Michael S. Sacks. Simulation of the 3d hyperelastic behavior of ventricular myocardium using a finite-element based neural-network approach. *Computer Methods in Applied Mechanics and Engineering*, 394:114871, 2022. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2022.114871. URL https://www.sciencedirect.com/science/article/pii/S0045782522001724.

Long Zhao, Xi Peng, Yu Tian, Mubbasir Kapadia, and Dimitris N. Metaxas. Semantic graph convolutional networks for 3d human pose regression. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2019. doi: 10.1109/cvpr.2019.00354. URL https://doi.org/10.1109%2Fcvpr.2019.00354.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020. ISSN 2666-6510. doi: https://doi.org/10.1016/j.aiopen.2021.01.001. URL https://www.sciencedirect.com/science/article/pii/S2666651021000012.

O.C. Zienkiewicz and R.L. Taylor. The finite element method, ; volume 2: solid and fluid mechanics, dynamics and non-linearity. 1991.

Anina Šarkić Glumac, Onkar Jadhav, Vladimir Despotović, Bert Blocken, and Stephane P.A. Bordas. A multi-fidelity wind surface pressure assessment via machine learning: A high-rise building case. *Building and Environment*, 234:110135, 2023. ISSN 0360-1323. doi: https://doi.org/10.1016/j.buildenv.2023.110135. URL https://www.sciencedirect.com/science/article/pii/S0360132323001622.